Controller Synthesis of Discrete Event Systems via Planning

Keywords:Non-classical logics for KR, Knowledge-based Software Engineering, Uncertainty in AI

Abstract

We show how AI automated planning techniques can be leveraged effectively to solve control problems of Discrete Event Systems. To do so, we first propose a careful (but simple) encoding of the DES controller synthesis problem into a planning problem that provably captures the compositional and reactive nature of DES specifications. We then report on experimental results comparing planning techniques under our encoding with existing synthesis tools for DES. The results show that the planning approach outperforms the controller synthesis tools, but also suggest that compositional analyses are more effective in some settings.

1 Introduction

Both Artificial Intelligence (AI) *automated planning* [Ghallab *et al.*, 2004; Geffner and Bonet, 2013] and *controller synthesis* of Discrete Event Systems (DES) [Ramadge and Wonham, 1987; Cassandras and Lafortune, 2006] look for an orderly combination of actions/events guaranteeing a given goal. Arising from different communities, they consider distinct perspectives on representational and computational aspects. In this work, we contribute to the recent efforts in relating the two fields (e.g., [Patrizi *et al.*, 2013; Sardina and D'Ippolito, 2015b; Camacho *et al.*, 2016]). Concretely, we show how to leverage on planning techniques to tackle DES synthesis problems. We then empirically demonstrate that the planning-based approach performs better than existing controller synthesis tools in many cases.

DES are discrete-state dynamical systems that react to the occurrence of diverse events. DES arise in many domains including, robotics, logistics, manufacturing, and networks. The behavior of DES is usually captured with state machines that update their state following a labeled transition relation. Controller synthesis for DES aims at controlling such systems to achieve certain guarantees. This is done by deploying a so-called "supervisor" controller that is able to *disable* the controllable events while monitoring the uncontrollable ones. Importantly, for engineering's reasons, the overall system is typically modeled by the *parallel composition* of multiple interacting components [Ramadge and Wonham, 1989; Pnueli and Rosner, 1990; Flordal *et al.*, 2007]. Furthermore, the various components usually include uncontrollable events (i.e., events that cannot be directly disabled) and shared events (i.e., events that synchronize multiple components). Process calculi, such as Communicating Sequential Processes (CSP) [Hoare, 1978], are often used to describe such components in a high-level succinct language.

Planning stems from a different tradition, namely, Knowledge Representation (KR) within AI. As such, the work there has special focus on adequate representations (of the dynamic system being modeled) and algorithms that can exploit such representations. A planning problem is specified by describing the preconditions and effects of actions, together with the goal to be achieved. Based on KR reasoning about action languages, powerful effective techniques have been developed over the years [Geffner and Bonet, 2013]. Unlike that in DES, the work in planning has been oriented mainly towards *non-reactive* environments and with no support for compositionality (and hence of synchronizing events).

Both, planning and control problems, are specified using compact descriptions. Their semantics are based on sorts of transition systems that are often exponential with respect to the size of such descriptions, due to the unfolding of the factored state representation in planning and the parallel composition in control. The state explosion problem has been dealt with in both disciplines using different approaches (e.g., compositional analysis, heuristic search).

In this paper, we provide evidence that planning can be a competitive computational approach for the synthesis of controllers for DES. To do so, we propose a way to compile a DES controller synthesis task into a (non-deterministic) planning task. To meet the correct execution semantics, the encoding needs to *(a)* capture the concurrent semantics of parallel composition (avoiding the construction of the exponential model); *(b)* realize the synchronization among components; and *(c)* account for uncontrollable events. We then carry out experiments over six classical control problems, three of them from the 9th International Workshop on Discrete Event Systems. The results show that using state-of-the-art planning techniques with our encoding outperforms existing DES synthesis tools. Nevertheless, they also suggest that investigating compositional analysis within planning frameworks may bring significant benefits. We hope our work will contribute to the awareness and cross-fertilization among the two fields.

2 Preliminaries

2.1 Automated Planning

Automated Planning is a model-based approach to the synthesis of plans involving the execution of actions (also called *operators*) that bring about a given *goal* in a domain. Importantly, the dynamics of the domain is specified with a *factorized* representation [Ghallab *et al.*, 2004; Bonet and Geffner, 2001] using appropriate languages. We shall focus here on the Fully Observable Non-deterministic (FOND) [Rintanen, 2008] variant, which extends classical planning with non-deterministic effects.

Definition 1 (FOND Planning). A *FOND planning problem* is a tuple $\mathcal{P} = \langle F, I, O, G \rangle$, where F stands for the problem *fluents* (i.e., boolean state propositions whose value changes due to action execution), $I \subseteq F$ encodes the initial *state*, O is a set of operators, and G is a set of literals from F (i.e., f or $\neg f$) defining the target *goal* to be achieved.

An *operator* is a pair $o = \langle Pre(o), \mathit{Eff}(o) \rangle$, where $Pre(o)$ is a boolean formula over F describing the *preconditions* of the operator, and $\mathit{Eff}(o) = e_1 \mid \cdots \mid e_n$, with $n \geq 1$, is the (non-deterministic) *effect* of o, where each e_i is a conjunction of conditional deterministic effects. A *conditional effect* has the form $C \Rightarrow E$, where C is a boolean formula over F and E is a set (conjunction) of *literals* over F. When $n = 1$ the action's effects are said to be deterministic. ■

Whenever an operator o with effects $\mathit{Eff}(o)=e_1|\cdots|e_n$ is executed, one of the e_i effects will ensue non-deterministically, that is, without the control of the executor.[1] In turn, a conditional effect $C \Rightarrow E$ states that when condition C holds – in the state in which the action is being executed – the set of literals E ought to hold in the successor state (and everything else remains static, thus addressing the frame problem). With this understanding, it is possible to define a function $Succ(o, s)$ denoting the *possible successor states* when operator o is executed in a state s [Rintanen, 2003]. A state is a subset of F (or conjunction of fluent atoms) representing those fluents that are true (in the state).

Rephrasing [Muise *et al.*, 2012], a solution to a FOND planning task is a *policy* $\pi : 2^F \mapsto 2^O$ that maps state s to a set of appropriate actions $\pi(s)$ such that the agent eventually reaches the goal. A policy is *closed* if it returns an action for every non-goal state potentially reached by following the policy. Then, a *strong plan* is a closed policy that achieves the goal and never visits the same state twice [Cimatti *et al.*, 2003]. A a strong plan guarantees the goal in a bounded *finite* number of steps. For the sake of this paper, we shall focus on strong plans, though other solutions exist for FOND planning and could be considered too.[2]

We close by noting that FOND problems can be specified in the Planning Domain Definition Language (PDDL) [McDermott *et al.*, 1998; Gerevini *et al.*, 2006], the de-facto standard language for specifying planning problems. Besides its convenience in terms of modeling, the language is often exploited by the actual planning algorithms, for example, for the automatic extraction of domain-independent heuristics.

Example. The following is a fragment of the PDDL model of the action of flipping a coin:

```
(:action flip
  :parameters (?c - coin)
  :precondition (holding ?c)
  :effect (and
    (oneof (heads ?c) (not (heads ?c)))
    (not (holding ?c)) ))

(:action pickup
  :parameters (?c - coin)
  :precondition (not (holding ?c))
  :effect (holding ?c) )
```

In words, flipping a coin c is possible if the agent is holding it and its effects may result in heads or tails (i.e., not heads) non-deterministically. In any case, the agent does not hold the coin anymore once it flips until it picks it up again. □

2.2 Controller Synthesis

Controller synthesis for DES focuses on a component interaction model, based on events [Ramadge and Wonham, 1989]. In particular, we address control problems for behavior models expressed as Label Transition Systems (LTS) and parallel composition defined broadly as synchronous product. That is, given a model of the assumed behavior of the environment (also called the *plant*), we look for an operational behavior model of a controller (also called *supervisor*) such that, when enacted in a consistent environment, the goal is guaranteed.

Definition 2 (Labeled Transition Systems). An LTS is a tuple $T{=}(S_T, A_T, \rightarrow_T, t_0)$, where S_T is a finite set of states, A_T is its alphabet, $\rightarrow_T \subseteq (S_T \times A_T \times S_T)$ is a transition relation, and $t_0 \in S_T$ is the initial state. ■

A complex environment E can be described by means of the parallel composition of simpler components $E_0, \ldots, E_n$.

Definition 3 (Parallel Composition). The *parallel composition* ($\|$) of two LTSs T and Q is a symmetric operator that yields an LTS $T\|Q = (S_T \times S_Q, A_T \cup A_Q, \rightarrow_{T\|Q}, \langle t_0, q_0 \rangle)$, where $\rightarrow_{T\|Q}$ is the smallest relation that satisfies:

$$\frac{t \xrightarrow{\ell}_T t'}{\langle t, q\rangle \xrightarrow{\ell}_{T\|Q} \langle t', q\rangle}\ \ell \in A_T \setminus A_Q \qquad \frac{q \xrightarrow{\ell}_Q q'}{\langle t, q\rangle \xrightarrow{\ell}_{T\|Q} \langle t, q'\rangle}\ \ell \in A_Q \setminus A_T$$

$$\frac{t \xrightarrow{\ell}_T t' \quad q \xrightarrow{\ell}_Q q'}{\langle t, q\rangle \xrightarrow{\ell}_{T\|Q} \langle t', q'\rangle}\ \ell \in A_T \cap A_Q$$

■

Note that the states in $E_0\| \ldots \|E_n$ can grow exponentially, as it contains the cross-product of the states of $E_0, \ldots, E_n$. Furthermore, note that the last rule realizes the synchronization between components, that is, it enforces the synchronous execution of shared events in the LTSs that have the events in their alphabets. Nonetheless, some controller synthesis techniques avoid the blow-up produced by the parallel composition by reasoning directly on the individual components, but still taking synchronization into account.

[1] This formalization of actions corresponds to the 1ND Normal Form [Rintanen, 2003] with no nested conditional effects, and to the usual `(oneof e1...en)` PDDL clauses [Gerevini *et al.*, 2006].

[2] E.g., a *strong cyclic plan* is a closed policy that achieves the goal and every reachable state can reach the goal using the policy.

Usually control goals are written in Linear Temporal Logic (LTL) [Keller, 1976]. However, the work in planning (particularly in FOND) deals mainly with reachability objectives. Thus, we identify and focus on two common types of goals that have a correspondence in the planning framework, namely *safety* (i.e., maintenance) and *co-safety* (i.e., achievement). These goals state properties related to the possible traces (i.e., a sequence of labeled transitions) of an LTS.

Definition 4 (Safety). We say that an LTS T satisfies a *safety* goal $G_\square$ given as a set of labels, if and only if a label of $G_\square$ never occurs in a trace of T. In other words, the events in $G_\square$ are never executed. ■

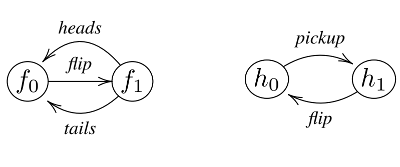
Definition 5 (Co-Safety). We say that an LTS T satisfies a *co-safety* goal $G_\Diamond$ given as a set of labels, if and only if in every trace of T there is at least one occurrence of a label of $G_\Diamond$. Or in other words, at least one event in $G_\Diamond$ is guaranteed to be executed eventually. ■

Given a partition of the alphabet of an environment E in controllable and uncontrollable events ($A_E = A_C \dot\cup A_U$), we look for a controller M that achieves a goal G by disabling some of the controllable events in E, while monitoring uncontrollable events.

Definition 6 (Control Problem). A *control problem* is a tuple $\mathcal{E}{=}(E, G, A_C)$, with $E{=}(S_E, A_E, \rightarrow_E, e_0)$ an LTS resulting from the parallel composition of LTSs $E_0 \| \ldots \| E_n$ where each $E_i{=}(S_{E_i}, A_{E_i}, \rightarrow_{E_i}, e_0^i)$, a control goal $G{=}(G_\square, G_\Diamond)$ with safety and co-safety subgoals, and $A_C \subseteq A_E$ a set of controllable events. ■

A solution for a control problem is a component that restricts the behavior of the environment only by disabling controllable events and, in doing so, it guarantees the goal specifications. In other words, the controller is itself an LTS M such that when composed in parallel with the environment E, it does not block uncontrollable events and satisfies the goal G, denoted $E \| M \models G$. When considering safety and co-safety goals, $G_\square$ and $G_\Diamond$ respectively, no trace of $M \| E$ contains safety violations in $G_\square$ and all traces must contain a co-safety goal in $G_\Diamond$.

Example. The following picture depicts two LTSs that captures the behavior of picking up and flipping a coin.



(a) LTS F (b) LTS H

Figure 1: LTSs modeling the flipping of a coin

After the controllable event *flip*, LTS F evolves into a state in which one of the uncontrollable events *heads* or *tails* will occur. Note that *flip* is shared with LTS H, and hence synchronizes. Thus, for *flip* to be executable H has to be in a state that allows it (i.e., after a *pickup* event occurs). □

3 Control via Planning

In this section we develop a translation from a control problem (Def. 6) to a planning problem (Def. 1). The intuition behind the translation is that one can model the behavior of any state machine within a planning domain. The main ingredients of our encoding are:

1. The behavior (i.e., state transitions) of each component in the control problem is modeled *separately*, thus avoiding the computation of the parallel composition.
2. Synchronization of components is modeled explicitly in the planning domain. Roughly speaking, the encoding forces the planner to take auxiliary steps to "aggregate" the locally available events of various components into a globally synchronizing event.
3. Each controllable event ℓ is mapped to an operator ℓ available to the planner at a special "selection phase."
4. Each uncontrollable event ℓ is mapped to an operator ℓ that the planner is forced to take after its selection at a distinguished "non-deterministic choice phase."
5. Unsuitable uncontrollable events for the current situation could be selected in the non-deterministic choice phase. In such a case we default then the choice of an uncontrollable event to the planner. This does not grant more control to the planner, as it still ought to consider (and resolve) every possible non-deterministic choice.

From now on, consider a control problem $\mathcal{E} = (E, G, A_C)$, with *(i)* an LTS $E = (S_E, A_E, \rightarrow_E, e_0)$ resulting from the parallel composition of components $E_0 \| \ldots \| E_n$ where $E_i = (S_{E_i}, A_{E_i}, \rightarrow_{E_i}, e_0^i)$, and with initial state $e_0 = \langle e_0^0, \ldots, e_0^n \rangle$; *(ii)* the control goal $G = (G_\square, G_\Diamond)$ with safety and co-safety subgoals; *(iii)* the controllable and uncontrollable events $A_C \subseteq A_E$ and $A_U = A_E \backslash A_C$, respectively.

We next encode $\mathcal{E}$ into a suitable FOND planning problem $\mathcal{P}_\mathcal{E} = \langle F, I, O, G \rangle$ as follows.

Fluents. The set of fluents of $\mathcal{P}_\mathcal{E}$ is defined as $F = \mathit{At} \cup \mathit{Ready} \cup \mathit{Enabled} \cup \mathit{Inprogress} \cup \mathit{Status}$, where:

- $\mathit{At} = \{\mathit{at}(e_j, E_i) \mid e_j \in S_{E_i}\}$, indicating if component E_i is at state e_j;
- $\mathit{Ready} = \{\mathit{ready}(\ell, E_i) \mid \ell \in A_{E_i}\}$, stating if event ℓ is locally available in component E_i. While this can be inferred from a disjunction of relevant *at* fluents, its explicit representation allows for a simpler model requiring no disjunctive preconditions;
- $\mathit{Enabled} = \{\mathit{enabled}(\ell) \mid \ell \in A_{E_0} \cup \ldots \cup A_{E_n}\}$, indicating whether event ℓ is *ready* in every component containing it (i.e., it synchronizes as per Def. 3). This information could be inferred from the conjunction of *ready* values, but it allows us to model the effects synchronization compactly and explicitly;
- $\mathit{Inprogress} = \{\mathit{inprogress}(\ell) \mid \ell \in A_U\}$, indicating that an uncontrollable event ℓ has been chosen for execution at a non-deterministic choice phase; and

- *Status* $= \{$*setup*, *synch*, *step*, *wild*, *busy*, *goal*, *error*$\}$, encoding the various phases the planner goes through at every "reasoning cycle," in which fluents *ready* and *enabled* are updated between the selection of actions, until a *goal* or an *error* (i.e. a state where a safety violation occurred) is reached.

The logical flow of the different phases is depicted in Figure 2 and is central to the encoding. In the initial phase the planner resets auxiliary fluents. In the second phase locally available events are collected and stored in auxiliary fluents. The third phase captures the effects of synchronization enabling globally synchronizing events and turning on the *wild* fluent that states if at least one uncontrollable event is enabled. The final phase is divided in two, depending if there are uncontrollable events enabled or not. In the former, the uncontrollable event to execute is chosen non-deterministically, while in the latter the planner can choose from any of the enabled controllable events. Finally the phase cycle starts over.

Initial and Goal States. The initial planning state $I = \{at(e_0^i, E_i) \mid 0 \leq i \leq n\}$ captures the initial state of each component. In turn, the goal specification is simply $G = \{goal, \neg error\}$, that is any state where *goal* fluent holds true and the *error* fluent does not.

Operators. Set $O = A_E \cup \{\mathsf{reset}, \mathsf{setR}, \mathsf{setE}, \mathsf{pickU}\}$ is the collection of planning operators, where:

- Operator setR "re-configures" *ready* fluents to capture which events are locally available in which components:

$$\begin{aligned} Pre(\mathsf{setR}) &= \{setup\}; \\ Eff(\mathsf{setR}) &= \{\neg setup, synch\} \cup \\ &\quad \{at(e, E_i) \Rightarrow ready(\ell, E_i) \mid e \xrightarrow{\ell}_{E_i} e'\}. \end{aligned}$$

- Operator setE sets *enabled* fluents when an event is *ready* at every component containing it:

$$\begin{aligned} Pre(\mathsf{setE}) &= \{synch\}; \\ Eff(\mathsf{setE}) &= \{\neg synch, step\} \cup \\ &\quad \{\{ready(\ell, E_i) \mid \ell \in A_{E_i}\} \Rightarrow \\ &\quad\ \{enabled(\ell)\} \cup \{wild \mid \ell \in A_U\} \mid \ell \in A\}. \end{aligned}$$

  This is a key operator in that it is the one responsible of realizing the synchronization among all components sharing an event. Thus an event is enabled if *all* the components sharing the event are in states where the event is *ready*. Furthermore, the operator sets the *wild* fluent if at least one uncontrollable event becomes enabled (the fluent could be inferred from the disjunction of the *enabled* fluents corresponding to uncontrollable events, but this allows for a simpler model requiring no disjunctive preconditions).

- Operator reset sets all *ready* and *enabled* fluents false:

$$\begin{aligned} Pre(\mathsf{reset}) &= \{\neg setup, \neg synch, \neg step, \neg busy, \neg error\}; \\ Eff(\mathsf{reset}) &= \{\neg wild, \neg goal, setup\} \cup \\ &\quad \{\neg enabled(\ell) \mid \ell \in A\} \cup \\ &\quad \{\neg ready(\ell, E_i) \mid \ell \in A_{E_i}\}. \end{aligned}$$

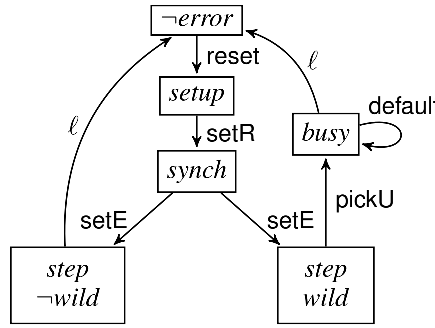


Figure 2: Flow of operators enforced by the encoding (boxes represent the fragments of the operators' preconditions that enforce the order while arrows represent operators).

- Operator pickU non-deterministically chooses an uncontrollable event from set $A_U = \{\ell_0, \cdots, l_k\}$:

$$\begin{aligned} Pre(\mathsf{pickU}) &= \{step, wild\}; \\ Eff(\mathsf{pickU}) &= \{\neg step, busy\} \cup \\ &\quad \{enabled(\ell_0) \Rightarrow inprogress(\ell_0) \mid \ldots \mid \\ &\quad\ enabled(\ell_k) \Rightarrow inprogress(\ell_k)\}. \end{aligned}$$

  Note that if the selected event is enabled then it is set to in-progress (i.e. forcing the planner to execute said event) and nothing changes otherwise.

- Operator default allows the planner to select an uncontrollable event (by setting all the enabled uncontrollable events to in-progress) if pickU picks a non-enabled event:

$$\begin{aligned} Pre(\mathsf{default}) &= \{busy\} \cup \{\neg inprogress(\ell) \mid \ell \in A_U\}; \\ Eff(\mathsf{default}) &= \{enabled(\ell) \Rightarrow inprogress(\ell) \mid \ell \in A_U\}. \end{aligned}$$

- Each operator $\ell \in A_E$ representing an event in the control problem – controllable or uncontrollable – updates the components as per their corresponding LTS transition relation. An uncontrollable event can be executed only if previously selected by the pickU operator (i.e. $inprogress(\ell)$ is true), while *enabled* controllable events can be selected by the planner when the *step* fluent holds.

$$\begin{aligned} Pre(\ell) &= \{step, \neg wild, enabled(\ell) \mid \ell \in A_C\} \cup \\ &\quad \{busy, inprogress(\ell) \mid \ell \in A_U\}; \\ Eff(\ell) &= \{\neg step, \neg busy\} \cup \\ &\quad \{goal \mid \ell \in G_\Diamond\} \cup \{error \mid \ell \in G_\Box\} \cup \\ &\quad \{at(e, E_i) \Rightarrow \{\neg at(e, E_i), at(e', E_i)\} \mid \\ &\quad\ e \xrightarrow{\ell}_{E_i} e' \wedge e \neq e'\}. \end{aligned}$$

It is not difficult to see that the complexity of the above translation is $\mathrm{O}\big((\sum_{i=0}^{n} |A_{E_i}|)(\sum_{i=0}^{n} |S_{E_i}|)^2\big)$, since:

1. the fluents consists of the union between states in S_{E_i} for all $0 \leq i \leq n$ and labels of A_E; and
2. in order to generate the operators each transition needs to be considered once (the transitions cannot surpass the connection of every state by every event).

The following result states that the encoding is indeed correct, in that it fully captures the control problem of interest.

Theorem 1 (Correctness). *Let $\mathcal{E}$ be a control problem and $\mathcal{P}_{\mathcal{E}}$ its corresponding planning problem as per the above encoding. Then, there exists a controller solution M for $\mathcal{E}$* if and only if *there exists a strong plan π for $\mathcal{P}_{\mathcal{E}}$. In addition, M can be constructed from π in linear time.*

PROOF SKETCH. We prove this by showing an isomorphism between the semantic models induced by the problems $\mathcal{E}$ and $\mathcal{P}_{\mathcal{E}}$. In [Geffner and Bonet, 2013] the semantics of planning problems are captured with state models. Whereas in [Piterman *et al.*, 2006] game structures are used to capture the semantics of control problems. Both approaches share common characteristics that we take into account to ground a unified formulation for the semantic models. The existence of an isomorphism follows from the fact that the semantic models are given by the same transition relation.[3]

Example. Consider LTSs F and H in Fig 1, which capture the flipping of a coin, a fragment of their translation to PDDL is as follows:

```
(:action setR
  :precondition (setup)
  :effect (and (not setup) (synch)
    (when (at f0 F) (ready flip F))
    (when (at f1 F) (ready heads F))
    (when (at f1 F) (ready tails F))
    (when (at h0 H) (ready pickup H))
    (when (at h1 H) (ready flip H)) ))

(:action setE
  :precondition (synch)
  :effect (and (not synch) step
    (when (ready tails F) (and (enabled tails) wild))
    (when (ready heads F) (and (enabled heads) wild))
    (when (ready pickup H) (enabled pickup))
    (when (and (ready flip F) (ready flip H))
      (enabled flip))) ))

(:action pickU
  :precondition (step wild)
  :effect (and (not step) busy
    (oneof (when (enabled tails) (inprogress tails))
           (when (enabled heads) (inprogress heads)) )))

(:action flip
  :precondition (and step (not wild) (enabled flip))
  :effect (and (not step)
    (when (at f0 F) (and (not (at f0 F)) (at f1 F)))
    (when (at h1 H) (and (not (at h1 H)) (at h0 H))) ))

(:action heads
  :precondition (and busy (inprogress heads))
  :effect (and (not busy)
    (when (at f1 F) (and (not (at f1 F)) (at f0 F))) ))
```

Note that non-deterministic choice is constrained to action pickU. Actions setR and setE set the auxiliary fluents *ready* and *enabled* respectively. Event *heads* updates only the local state of LTS F, while shared event *flip* updates the local state of both LTSs (i.e., capturing the effects of synchronization).□

4 Evaluation

In this section we report on an evaluation of our translation. The aim of the evaluation is to validate whether the translation successfully allows to leverage on advances in planning to solve control problems and to compare the performance of techniques from both disciplines. For this we selected a benchmark of three classical control problems presented in the 9th International Workshop on Discrete Event Systems, and we enriched it with three new problems inspired in the control literature. The six problems can be scaled up in two different directions, namely the number of intervening components (n) and the number of states per component (k).

We now briefly describe the problems in the benchmark:

- TL (Transfer Line): One of the most traditional examples in controller synthesis. The TL consists of n machines $M(1), \ldots, M(n)$ connected by n buffers $B(1), \ldots, B(n)$ with finite capacity k and ending in a special machine called Test Unit. The goal of the problem is to output a processed element avoiding overflows.
- DP (Dinning Philosophers): The classical problem where n philosophers with n forks sit around a table sharing one fork with each adjacent philosopher. The goal is to control the access to the forks avoiding a deadlock and allowing each philosopher to eat at least once while performing k intermediate etiquette steps.
- CM (Cat and Mouse): n cats and n mice are placed in opposite ends of a corridor divided in $5k$ cells. They move taking turns one cell at a time. The goal of the problem is to control the mice in order to reach the center of the corridor while avoiding sharing a cell with a cat.
- TA (Travel Agency): A travel agency agency receives requests for vacation packages and to fulfill them it interacts with n different web-services, which after k selection steps may offer a reservation. The goal is to orchestrate the web-services to provide a vacation package when possible, avoiding to pay for incomplete packages.
- BW (Bidding Workflow): A company evaluates n projects in order to decide which ones to bid for. For this, a document describing the project needs to be accepted by up to k teams with different specializations. The goal is to synthesize a workflow that evaluates all documents.
- AT (Air-traffic management): An airport control tower receives requests from n planes wishing to land. The tower needs to signal them if it is safe to approach the ramp or at which of k spaces they must perform holding maneuvers. The goal is to control the air traffic guaranteeing that all the planes eventually land safely.

For each problem we vary the parameters n and k independently between 1 and 6. Hence, the evaluation considers the execution of 36 tests per case study, totaling 216 tests.

In this evaluation we consider the following four tools:

From Control:

1. MTSA [D'Ippolito *et al.*, 2008], implementing monolithic explicit state representation.
2. Supremica [Mohajerani *et al.*, 2011], implementing compositional explicit state representation.

From Planning:

3. MBP [Cimatti *et al.*, 2003], implementing monolithic symbolic state representation with BDDs.
4. PRP [Muise *et al.*, 2014], implementing heuristic on-the-fly exploration over a explicit state representation.

[3] Proof technicalities and detailed evaluation results can be found in https://www.dropbox.com/s/nk489uop8e6kow8/CSPA.pdf

	MTSA		PRP		SUP		MBP	
	S	T	S	T	S	T	S	T
TL	19	6.21	**36**	0.15	20	1.46	30	49.54
DP	25	26.91	**36**	0.73	34	50.01	24	52.33
CM	16	6.69	12	48.3	**19**	33.25	11	27.21
TA	17	11.71	**31**	61.32	13	3.5	12	5.2
BW	18	16.24	17	38.01	**18**	5.31	15	51.11
AT	34	15.51	**35**	26.75	27	34.3	22	38.35
Total	129	83.28	**167**	175.26	131	127.83	114	223.73

Table 1: Results (S stands for successes and T for total time)

In order to be able to include MBP in the evaluation we had to reimplement a pre-processing step that translates PDDL files to SMV, since the original implementation fails in all but the simpler cases. We believe that the error is caused by the size of the generated file. Despite doing this we are still following our translation, but instead of targeting PDDL we target SMV directly.

In Table 1 we show the results of the evaluation. Due to lack of space, we only report on the totals of solved cases (i.e., not time outs, out of memory or other failures) and the total execution time (sum of times of the successful runs) in minutes. We do not report the time required by the translation because it is negligible for all cases. The experiments were run on a desktop computer with an Intel i7-3770, 8GB of RAM, and a time out of 30 minutes.

From the results we can make the following observations:

1. PRP having as input the PDDL file obtained from our translation and relying on heuristic search, solves more instances than the other tools and usually in less time.
2. All the tools are able to consistently solve more instances of the TL, DP and AT problems, while CM and BW prove to be particularly challenging.
3. Supremica, relying on a compositional analysis, performs better than the other tools on CM and BW.
4. Despite relying on a explicit representation MTSA is able to solve many instances.
5. Despite relying on a symbolic representation we do not observe an advantage in using MBP over the other tools.

5 Related Work

To the best of our knowledge our translation is the first attempt to reduce a control problem into planning exploiting the compositional aspects of the control specification. However, there are numerous reductions from decision problems to planning. Despite being different from ours some shared characteristics with other reductions can be found.

In [Fritz *et al.*, 2008] a reduction from ConGolog, a logical programming language for agents, to situation calculus is presented. This translation shares some characteristics with ours, namely the schematization of different phases in the execution semantics of the respective models. However, their translation relies on building a monolithic Petri-Net, while we explicitly encode the synchronization mechanism.

In [Sohrabi *et al.*, 2010] a reduction from diagnosis to planning is presented. The focus of the paper is on the proper characterization of diagnoses and their relation to planning. Despite working on automata they follow a monolithic approach, that is, they consider a single component instead of the composition of multiple synchronizing components. Variations of the translation presented herein could extend said work by allowing to efficiently encode the diagnosis of componentized domains in planning.

In [Sardina and D'Ippolito, 2015a] an inverse reduction, that is from planning to control, is presented in order to explicitly characterize fairness assumptions. However, the reduction passes through the underlying semantic model and thus it can incur in an exponential cost. The bidirectional existing reductions highlight the similarities between the disciplines. Still we believe that, for practical purposes, compositional reductions are required.

Reductions from the LTL synthesis problem to FOND planning are presented in [Patrizi *et al.*, 2013] and [Camacho *et al.*, 2016]. The former considers deterministic and non-deterministic actions for a limited form of LTL goals. The latter improves on this approach by dropping the restriction on the LTL goals and attaining greater efficiency. Despite sharing some methodological similarities with our translation, none of these approaches consider compositional representations as the ones typically used in DES, nor compared results with tools from this field. Hence, the advantages of the compositional analyses had gone unnoticed.

6 Conclusions and future work

In this paper we present a translation from discrete event control problems to planning. The translation is polynomial with respect to the size of a compact input specification since it takes advantage of the compositional aspects of the control specification. Applying the translation we compared different tools and found that it effectively allows leveraging the advances in planning to solve control problems.

Since the results of the automatic translation are structured differently and are usually larger than manually written specifications, the risk of using the translation is that it may hinder the planners. We have found that this varies from planner to planner depending on how they process the input files. While PRP worked flawlessly, the translation from PDDL to SMV performed by MBP failed and we had to reimplement it.

Despite the fact that PRP – using heuristic search – achieved the best results, Supremica – using a compositional approach – also performed well and was able to solve problem instances that the other tools failed to handle. This raises the question of whether these techniques can be combined for greater efficacy. We believe this question may not have arisen in the planning community due to the lack of primitives for compositional description, whereas in control the formalisms used do not seem amenable for heuristic search. The combination of techniques from both areas shows promise and we intend to work on it in the future.

References

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning and control in artificial intelligence: A unifying perspective. *Applied Intelligence*, 14(3):237–252, 2001.

[Camacho *et al.*, 2016] Alberto Camacho, Eleni Triantafillou, Christian J. Muise, Jorge A. Baier, and Sheila A. McIlraith. Non-deterministic planning with temporally extended goals: Completing the story for finite and infinite LTL. In *Proceedings of the Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KNOWPROS)*, 2016.

[Cassandras and Lafortune, 2006] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer, Secaucus, NJ, USA, 2006.

[Cimatti *et al.*, 2003] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, 147, 2003.

[D'Ippolito *et al.*, 2008] N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel. MTSA: The Modal Transition System Analyser. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, ASE, 2008.

[Flordal *et al.*, 2007] Hugo Flordal, Robi Malik, Martin Fabian, and Knut Åkesson. Compositional Synthesis of Maximally Permissive Supervisors Using Supervision Equivalence. *Discrete Event Dynamic Systems*, 17(4):475–504, 2007.

[Fritz *et al.*, 2008] Christian Fritz, Jorge A Baier, and Sheila A McIlraith. ConGolog, Sin Trans: Compiling ConGolog into Basic Action Theories for Planning and Beyond. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, KR, pages 600–610, 2008.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[Gerevini *et al.*, 2006] Alfonso Gerevini, Blai Bonet, and Bob Givan, editors. *Booklet of 4th International Planning Competition*, Lake District, UK, 2006.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc., May 2004.

[Hoare, 1978] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21, 1978.

[Keller, 1976] Robert M. Keller. Formal Verification of Parallel Programs. *Communications of the ACM*, 19(7), 1976.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, A. Howe, Craig A. Knoblock, A. Ram, M. Veloso, Daniel S. Weld, and David E. Wilkins. PDDL—The planning domain definition language. Technical Report DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, Connecticut, 1998.

[Mohajerani *et al.*, 2011] Sahar Mohajerani, Robi Malik, Simon Ware, and Martin Fabian. Compositional synthesis of discrete event systems using synthesis abstraction. In *Control and Decision Conference*, CCDC, 2011.

[Muise *et al.*, 2012] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 172–180, 2012.

[Muise *et al.*, 2014] Christian J Muise, Sheila A McIlraith, and Vaishak Belle. Non-Deterministic Planning With Conditional Effects. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling*, ICAPS, pages 370–374, 2014.

[Patrizi *et al.*, 2013] Fabio Patrizi, Nir Lipovetzky, and Hector Geffner. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[Piterman *et al.*, 2006] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) Designs. In *Proceedings of the 7th International Conference on Verification, Model Checking and Abstract Interpretation*, volume 3855, 2006.

[Pnueli and Rosner, 1990] A. Pnueli and R. Rosner. Distributed Reactive Systems Are Hard to Synthesize. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, volume 2 of *SFCS*, pages 746–757, 1990.

[Ramadge and Wonham, 1987] P. J. Ramadge and W. M. Wonham. Supervisory Control of a Class of Discrete Event Processes. *SIAM Journal on Control and Optimization*, 25, 1987.

[Ramadge and Wonham, 1989] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77, 1989.

[Rintanen, 2003] Jussi Rintanen. Expressive equivalence of formalisms for planning with sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 185–194, 2003.

[Rintanen, 2008] Jussi Rintanen. Regression for classical and nondeterministic planning. In *Proceedings of the European Conference in Artificial Intelligence (ECAI)*, pages 568–572, 2008.

[Sardina and D'Ippolito, 2015a] Sebastian Sardina and Nicolas D'Ippolito. Towards Fully Observable Non-deterministic Planning As Assumption-based Automatic Synthesis. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI, pages 3200–3206, 2015.

[Sardina and D'Ippolito, 2015b] Sebastian Sardina and Nicolas D'Ippolito. Towards fully observable non-deterministic planning as assumption-based reactive synthesis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3200–3206, 2015.

[Sohrabi *et al.*, 2010] Shirin Sohrabi, Jorge A Baier, and Sheila A McIlraith. Diagnosis as Planning Revisited. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning*, KR, pages 26–36, 2010.