LTL$_f$ Goal-oriented Service Composition - Supplementary Material

Primary Keywords: *(4) Theory; (8) Knowledge Representation/Engineering*

Proofs of Section "Solution Technique"

Proposition 1. *$a_0 \dots a_{m-1} \in \mathcal{L}(\mathcal{A}_\varphi)$ iff $(a_0, q_1) \dots (a_{m-1}, q_m) \in \mathcal{L}(\mathcal{A}_{\mathsf{act}})$, for some $q_1 \dots q_m$.*

Proof. By definition, $a_0 \dots a_{m-1} \in \mathcal{L}(\mathcal{A}_\varphi)$ iff there exist a run $r = q_0, q_1 \dots q_m$ s.t. for $1 \leq k \leq m$, $\delta(q_{k-1}, a_k) = q_k$ and $q_m \in F$. Consider the word $w' = (a_0, q_1) \dots (a_{m-1}, q_m)$. By construction of $\mathcal{A}_{\mathsf{act}}$, w' induces a run $r^d = r$. Since $q_m \in F$ by assumption, r^d is an accepting run for $\mathcal{A}_{\mathsf{act}}$, and therefore $w' \in \mathcal{L}(\mathcal{A}_{\mathsf{act}})$ is accepted. The other direction follows by construction because, if $(a_0, q_1) \dots (a_{m-1}, q_m) \in \mathcal{L}(\mathcal{A}_{\mathsf{act}})$, then by construction $q_1 \dots q_m$ is a run of $\mathcal{A}_\varphi$ over word $a_0 \dots a_{m-1}$, and since $q_m \in F$ by assumption $a_0 \dots a_{m-1} \in \mathcal{L}(\mathcal{A}_\varphi)$. $\square$

Proposition 2. *Let h be a history with $\mathcal{C}$ and φ be a specification. Then, h is successful iff there exist a word $w \in A'^*$ such that $h = \tau_{\varphi,\mathcal{C}}(w)$ and $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$.*

Proof. We prove both directions of the equivalence separately.

($\Leftarrow$) Let w be a word $w = (a_0, q_1, o_0, \sigma_{o_0}) \dots (a_{m-1}, q_m, o_{m-1}, \sigma_{o_{m-1}}) \in A'^*$ Assume $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. We have that the induced run $r = (q_0, \sigma_{10} \dots \sigma_{n0}), \dots, (q_m, \sigma_{1m} \dots \sigma_{nm})$ over $\mathcal{A}_{\varphi,\mathcal{C}}$ is accepting. This means that $q_m \in F$ and $\sigma_{im} \in F_i$ for all i. Now consider the history $h = \tau_{\varphi,\mathcal{C}}(w) = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots (\sigma_{1m} \dots \sigma_{nm})$. For every $0 \leq k < m$, we have by definition of δ' that $\delta_i(\sigma_{i,k}, a_k) = \sigma_{i,k+1}$ if $i = o_k$, and $\sigma_{i,k} = \sigma_{i,k+1}$ otherwise. Moreover, $o_k \in \{1 \dots n\}$, $a_k \in A$ and $\sigma_{ik} \in \Sigma_i$, for all $i \in \{1, \dots, n\}$ and $0 \leq k < m$. Therefore, h is a valid history. Moreover, h is a successful history because $q_m \in F$ iff $a_0, \dots, a_{m-1} \models \varphi$, and $\sigma_{im} \in F_i$ for all i.

($\Rightarrow$) Assume that $h = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots, (\sigma_{1m} \dots \sigma_{nm})$ is a successful history with $\mathcal{C}$ Then we both have that (*i*) $\mathsf{actions}(h) = a_0, \dots, a_{m-1} \models \varphi$ and (*ii*) $\sigma_{im} \in F_i$ for all i. By construction of the NFA $\mathcal{A}_\varphi$, proposition (*i*) implies that there exists a run $q_0, \dots, q_m$ where $q_m \in F$. Moreover, let $\sigma_1, \dots, \sigma_m$ be the sequence of service states s.t. $\sigma_k = \sigma_{i,k}$ (where $i = o_k$), for $1 \leq k \leq m$. Now consider the sequence $w = (a_0, q_1, o_0, \sigma_1), \dots, (a_{m-1}, q_m, o_{m-1}, \sigma_m)$. By construction, we have $\tau_{\varphi,\mathcal{C}}(w) = h$. Moreover, from proposition (*i*) and proposition (*ii*), it follows that $w \in \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. This concludes the proof. $\square$

DFA *games* are games between two players, here called respectively the *environment* and the *controller*, that are specified by a DFA. We have a set of $\mathcal{X}$ of *uncontrollable symbols*, which are under the control of the environment, and a set $\mathcal{Y}$ of *controllable symbols*, which are under the control of the controller. A *round* of the game consists of both the controller and the environment choosing the symbols they control. A (complete) *play* is a word in $\mathcal{X} \times \mathcal{Y}$ describing how the controller and environment set their propositions at each round till the game stops. The *specification* of the game is given by a DFA $\mathcal{G}$ whose alphabet is $\mathcal{X} \times \mathcal{Y}$. A play is *winning* for the controller if such a play leads from the initial to a final state. A *strategy* for the controller is a function $f : \mathcal{X}^* \rightarrow \mathcal{Y}$ that, given a history of choices from the environment, decides which symbols $\mathcal{Y}$ to pick next. In this context, a history has the form $w = (X_0, Y_0) \cdots (X_{m-1}, Y_{m-1})$. Let us denote by $w_{\mathcal{X}}|_k$ the sequence projected only on $\mathcal{X}$ and truncated at the k-th element (included), i.e., $w_{\mathcal{X}}|_k = X_0 \cdots X_k$. A *winning strategy* is a strategy $f : \mathcal{X}^* \rightarrow \mathcal{Y}$ such that for all sequences $w = (X_0, Y_0) \cdots (X_{m-1}, Y_{m-1})$ with $Y_i = f(w_{\mathcal{X}} \,|_k)$, we have that w leads to a final state of our DFA game. The *realizability* problem consists of checking whether there exists a *winning strategy*. The *synthesis* problem amounts to actually computing such a strategy.

We now give a sound and complete technique to solve realizability for DFA games. We start by defining the *controllable preimage* $PreC(\mathcal{E})$ of a set $\mathcal{E}$ of states $\mathcal{E}$ of $\mathcal{G}$ as the set of states s such that there exists a choice for symbols $\mathcal{Y}$ such that for all choices of symbols $\mathcal{X}$, game $\mathcal{G}$ progresses to states in $\mathcal{E}$. Formally, $PreC(\mathcal{E}) = \{s \in S \mid \exists Y \in \mathcal{Y}.\forall X \in \mathcal{X}.\delta(s, (X, Y)) \in \mathcal{E}\}$. Using such a notion, we define the set $Win(\mathcal{G})$ of winning states of a DFA game $\mathcal{G}$, i.e., the set formed by the states from which the controller can win the DFA game $\mathcal{G}$. Specifically, we define $Win(G)$ as a least-fixpoint, making use of approximates $Win_k(\mathcal{G})$ denoting all states where the controller wins in at most k steps: (1) $Win_0(\mathcal{G}) = F$ (the final states of $\mathcal{G}$); and (2) $Win_{k+1}(\mathcal{G}) = Win_k(\mathcal{G}) \cup PreC(Win_k(\mathcal{G}))$. Then, $Win(\mathcal{G}) = \bigcup_k Win_k(\mathcal{G})$. Notice that computing $Win(\mathcal{G})$ requires linear time in the number of states in $\mathcal{G}$.

Indeed, after at most a linear number of steps $Win_{k+1}(\mathcal{G}) = Win_k(\mathcal{G}) = Win(\mathcal{G})$. It can be shown that a DFA game $\mathcal{G}$ admits a winning strategy iff $s_0 \in Win(\mathcal{G})$ (De Giacomo and Vardi 2015).

The resulting strategy is a transducer $T = (\mathcal{X} \times \mathcal{Y}, Q', q'_0, \delta_T, \theta_T)$, defined as follows: $\mathcal{X} \times \mathcal{Y}$ is the input alphabet, Q' is the set of states, q'_0 is the initial state, $\delta_T : Q' \times \mathcal{X} \to Q'$ is the transition function such that $\delta_T(q, X) = \delta'(q, (X, \theta(q))$, and $\theta_T : Q \to \mathcal{Y}$ is the output function defined as $\theta_T(q) = Y$ such that if $q \in Win_{i+1}(\mathcal{G}) \setminus Win_i(\mathcal{G})$ then $\forall X.\delta(q, (X, Y)) \in Win_i(\mathcal{G})$ (De Giacomo and Vardi 2015).

Theorem 1. *The* realizability of service composition for LTL_f goals *with community $\mathcal{C}$ for the satisfaction of an LTL_f goal specification φ can be solved by checking whether $q'_0 \in Win(\mathcal{A}_{\varphi,\mathcal{C}})$.*

Proof. The proof is rather technical, therefore we first give an intuitive explanation. Soundness can be proved by induction on the maximum number of steps i for which the controller wins the DFA game from q'_0, building γ in a backward fashion such that it chooses $(a_k, o_k) \in A'$ that allows forcing the win in the DFA game (which exists by assumption). Completeness can be shown by contradiction, assuming that there exists an orchestrator γ that realizes φ with community $\mathcal{C}$, but that $q'_0 \notin Win(\mathcal{A}_{\varphi,\mathcal{C}})$; the latter implies that we can build an arbitrarily long history that is not successful, by definition of winning region, contradicting that γ realizes φ.

We now provide the full proof of the claim by separately proving the soundness and completeness of our approach.

Soundness. Assume $q'_0 = (q_0, \sigma_{10} \dots \sigma_{n0}) \in Win_m(\mathcal{A}_{\varphi,\mathcal{C}})$, i.e. the controller can win in at most m steps; we aim to show that there exists an orchestrator γ that realizes φ, i.e. all executions t are finite and successful.

We prove it by induction on the maximum number of steps i for which the controller wins the DFA game from q'_0, building γ in a backward fashion.

Base case ($k = 0$): assume $q'_0 \in F'$, i.e. q'_0 is already a goal state. Then, the problem is trivially realizable since the goal specification is already satisfied (the execution $t = (\sigma_{10} \dots \sigma_{n0})$ is a successful execution for any γ).

Inductive case: assume the claim holds for every state $q'_k \in Q'$ that can reach an accepting state in at most k steps, i.e. $q'_k = (q_k, \sigma_{1k} \dots \sigma_{nk}) \in Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$, and let γ_k be the orchestrator that realizes the goal specification starting from such states. Let $\Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}}) = Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}}) \setminus Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$. Consider a state $q'_{k+1} = (q_{k+1}, \sigma_{1,k+1} \dots \sigma_{n,k+1}) \in \Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}})$. By construction, $q'_{k+1} \in PreC(Win_k(\mathcal{A}_{\varphi,\mathcal{C}}))$. This means that there exist a controllable symbol $Y \in \mathcal{Y}$ such that for all uncontrollable symbols $X \in \mathcal{X}$ we can reach a state in $Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$, i.e. $\delta(q'_{k+1}, (X, Y)) = q'_k \in Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$. Let γ_{k+1} be the orchestrator that behaves as γ_k in states in $Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$, and $\gamma_{k+1}((\sigma'_{1,k+1} \dots \sigma'_{n,k+1})) = (a_{k+1}, o_{k+1})$, where $Y = (a_{k+1}, q'_{k+2}, o_{k+1})$, for every $\sigma'_{1,k+1} \dots \sigma'_{n,k+1}$ such that $(q'_{k+1}, \sigma'_{1,k+1} \dots \sigma'_{n,k+1}) \in \Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}})$. By the inductive hypothesis, we have that all finite executions t_k of γ_k, starting from some $(\sigma_{1k} \dots \sigma_{nk}) \in \Delta Win_k(\mathcal{A}_{\varphi,\mathcal{C}})$, are successful finite executions. To prove our claim, we only need to show that all t_{k+1}, starting from some state in $\Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}})$, are also successful executions. If $|t_{k+1}| \leq k$, e.g. when the adversary behaves cooperatively, then it holds by inductive hypothesis. For executions of length $k+1$, this is the case because, by construction, we have $t_{k+1} = \sigma'_{z,k+1}, (a_{k+1}, o_{k+1}), t'$ for some execution t' of γ_k, some $(q'_{k+1}, \sigma'_{z,k+1}) \in \Delta Win_{k+1}(\mathcal{A}_{\varphi,\mathcal{C}})$, and $(a_{k+1}, o_{k+1}) = \gamma_{k+1}(\sigma'_{z,k+1})$. In other words, t_{k+1} is a valid finite execution of γ_{k+1}, and moreover, it is successful since t' is successful. Finally, we have that γ_m, by induction, is an orchestrator that can force the win of the game from q'_0.

Completeness. By contradiction, assume there exists an orchestrator γ that realizes φ with community $\mathcal{C}$, but that $q'_0 \notin Win(\mathcal{A}_{\varphi,\mathcal{C}})$. If $q'_0 \notin Win(\mathcal{A}_{\varphi,\mathcal{C}})$, then it means, by definition of $Win(\mathcal{A}_{\varphi,\mathcal{C}})$ and $PreC$, that for all $Y \in \mathcal{Y}$, there exist $X \in \mathcal{X}$ such that the successor state q'_1 is not in $Win(\mathcal{A}_{\varphi,\mathcal{C}})$. Therefore, we can recursively generate an arbitrarily long word $w = (X_0, Y_0) \dots (X_{m-1}, Y_{m-1})$, for any choice of $Y_0 \dots Y_{m-1}$, such that $w \notin \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$. Now consider the execution $t = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), (\sigma_{11} \dots \sigma_{n1}) \dots (a_{m-1}, o_{m-1}), (\sigma_{1m} \dots \sigma_{nm})$, built as follows: for all $0 \leq k \leq m-1$, $(a_k, o_k) = \gamma((\sigma_{10} \dots \sigma_{n0}) \dots (\sigma_{1k} \dots \sigma_{nk}))$, and for any $Y_k = (a_k, q_{k+1}, o_k)$, take $X_k = \sigma$ such that $w = (Y_0, X_0) \dots (Y_k, X_k) \notin \mathcal{L}(\mathcal{A}_{\varphi,\mathcal{C}})$ and $\delta_{o_k}(\sigma_{o_k,k}, a_k) = \sigma$, and set $\sigma_{o_k,k} = \sigma$. In other words, we consider any valid execution t of γ where the successor state of the chosen service is taken according to the winning environment strategy (which is losing for the agent). By construction, t is an infinite execution of γ. Moreover, no prefix of t it is not successful, because w is not accepted by $\mathcal{A}_{\varphi,\mathcal{C}}$ and, by construction of $\mathcal{A}_{\varphi,\mathcal{C}}$, this means that either $\mathsf{actions}(h) \not\models \varphi$, or for some service $\mathcal{S}_i$, $\sigma_{ik} \notin F_i$. Since we assumed that γ realizes φ with community $\mathcal{C}$, but we can construct an execution t of γ that is not successful, we reached a contradiction. $\square$

Theorem 2. *Let the composition problem be $\langle \varphi, \mathcal{C} \rangle$, and let the transducer T be a winning strategy over the game arena $\mathcal{A}_{\varphi,\mathcal{C}}$. Let γ_T be the orchestrator extracted from T, as defined above. Then, γ_T realizes φ with community $\mathcal{C}$.*

Proof. By definition, γ_T realizes φ with $\mathcal{C}$ if all its executions are finite successful traces. By contradiction, assume this is not the case, and that there exists an infinite execution $t = (\sigma_{10} \dots \sigma_{n0}), (a_0, o_0), \dots$, for which all finite prefixes t' of t are such that $\mathsf{actions}(t') \not\models \varphi$ and $last(\mathsf{states}(t')) \notin F_1 \times \cdots \times F_n$. Since the set of states $\Sigma_1 \times \cdots \times \Sigma_n$ is finite, it must be the case that a service configuration $\sigma_1, \dots \sigma_n$ is visited more than once in t. Consider the corresponding infinite trace produced by the strategy implemented by T while playing the game $\mathcal{A}_{\varphi,\mathcal{C}}$: $\tau = (q_0, \sigma_{10} \dots \sigma_{n0}), (a_0, q_1, o_0, \sigma_{o_0}), \dots$. By construction of $\mathcal{A}_{\varphi,\mathcal{C}}$, it follows that there exists an environment move X that forces the agent to loop infinitely and never reach the goal states in $\mathcal{A}_{\varphi,\mathcal{C}}$. But this contradicts the fact that T is a winning strategy that forces the game to reach an accepting state. $\square$

Implementation and Applications

This section describes our software prototype to solve the composition problem and the tests we executed on industrial case studies taken from the literature.

The tool. Our tool takes in input a list of services (in explicit representation) and a LTL_f goal specification and computes a PDDL specification (i.e. domain and problem files) of the corresponding FOND planning task, using the technique formalized in the previous section.

First, we construct $\mathcal{D}_\mathcal{C}$ and $\Gamma_\mathcal{C}$ in PDDL form. The PDDL domain file represents the nondeterministic behaviour of the services. One of the challenges we encountered was that some planning systems do not support the `when` expression with complex effect types, such as `oneof`; this prevented us from specifying the transitions as a list of `when` expressions, one for each possible starting state, each followed by a `oneof` expression that includes all the possible successors. To workaround this issue, given an action $\langle a, i\rangle$ of $\mathcal{D}_\mathcal{C}$, we defined a PDDL operator $\langle a, i, \sigma_{ij}\rangle$, one for each possible starting state $\sigma_{ij} \in \Sigma_i$ of service i; In this way, we can use the `oneof` effect without nesting it into a `when` expression. Then, to include the on-the-fly evaluation of the LTL_f goal specification φ, we rely on the (Torres and Baier 2015)'s translator (in the following, denoted with TB), implemented in SWI-Prolog, for what concerns the LTL_f encoding in the planning problem.

The encoding of the goal formula in PDDL follows the syntax supported by the TB translator, which we recall now. The goal LTL_f formula φ is encoded in PDDL using the following translation function $\mathcal{T}$:

- $\mathcal{T}(a)$ = `(a)`: the atomic proposition is encoded as an atomic PDDL predicate. E.g. the action $\mathit{move\_up}$ is translated into the atomic ground predicate `(move_up)`;
- $\mathcal{T}(\neg\varphi) =$ `(not` $\mathcal{T}(\varphi)$`)`;
- $\mathcal{T}(\varphi_1 \wedge \varphi_2) =$ `(and` $\mathcal{T}(\varphi_1)\,\mathcal{T}(\varphi_2)$`)`;
- $\mathcal{T}(\circ\varphi) =$ `(next` $\mathcal{T}(\varphi)$`)`;
- $\mathcal{T}(\bullet\varphi) =$ `(weaknext` $\mathcal{T}(\varphi)$`)`;
- $\mathcal{T}(\varphi_1 \mathcal{U} \varphi_2) =$ `(until` $\mathcal{T}(\varphi_1)\,\mathcal{T}(\varphi_2)$`)`;
- $\mathcal{T}(\varphi_1 \mathcal{R} \varphi_2) =$ `(release` $\mathcal{T}(\varphi_1)\,\mathcal{T}(\varphi_2)$`)`;
- $\mathcal{T}(\Diamond\varphi) =$ `(eventually` $\mathcal{T}(\varphi)$`)`;
- $\mathcal{T}(\Box\varphi) =$ `(always` $\mathcal{T}(\varphi)$`)`.

The final `(:goal)` section includes, in conjunction: (*i*) the goal specified by the TB encoding, and (*ii*) the formula that specifies the accepting configuration for all services. The PDDL formula for the accepting service configuration has the form:

```
(and
  (or (curstate_s1 σ11)(curstate_s1 σ12)...)
  ...
  (or (curstate_sn σn1)(curstate_sn σn2)...)
)
```

For all services $\mathcal{S}_i$ with $i = 1 \ldots n$, and $\sigma_{ij} \in F_i$. Intuitively, the formula captures the condition that for each service, it holds that in the current planning state each service is in either one of its final states. Note that we could have encoded the final acceptance condition by means of the formula $\Diamond(\phi \wedge \bullet \mathit{true})$, where ϕ is a propositional formula, which is the formula that is accepting whenever, in the current state of the trace, ϕ is true. However, this would have burdened the TB translator with a larger LTL_f formula, ending up in enlarging the overhead of the encoding (i.e. more sync actions, more NFA states, etc.).

The TB translator supports four modes: Simple, OSA, PG, and OSA+PG, where Simple is the "naive" translation (cfr. Torres and Baier 2015, Section 4) and OSA, OSA+PG are two optimizations called "Order for Synchronization Action" and "Positive Goals" (cfr. *ibid.*, Section 4.3). OSA+PG is the combination of OSA and PG.

Case Studies. To test our tool, we considered case studies inspired by the literature on service composition applied to the Smart Manufacturing and Digital Twins industry.

Electric Motor (EM). We consider a simplified version of the electric motor assembly, proposed in the context of Digital Twins composition for Smart Manufacturing (De Giacomo et al. 2023). We consider the production process of an electric motor widely used in various applications such as industrial machinery, electric vehicles, household appliances, and many others (De Giacomo et al. 2023). To function properly, electric motors require certain materials that possess specific electrical and magnetic properties. Therefore, before the manufacturing processes start, the raw materials (i.e., copper, steel, aluminium, magnets, insulation materials, bearings) must be extracted and refined to obtain essential metals and polymers for electric motor parts manufacturing. When the materials are in the manufacturing facility, the effective manufacturing process can start. For the sake of brevity, in the following, we focus on the main aspects of the manufacturing process, skipping the provisioning, but the formalization can be easily extended to cover more details.

The main components of an electric motor are the stator, the rotor, and, in the case of alternate current motors with direct current power (e.g., in the case of electric cars), the inverter. These three components are built or retrieved in any order, but the final assembly step must have all the previous components available. Moreover, after the assembly step, it is required that at least one test between an electric test and a full static test must be performed. This goal is captured by the following LTL_f constraints:

$$
\begin{aligned}
&\Diamond(\mathit{assembleMotor}) && (1)\\
&\wedge (\neg \mathit{assembleMotor}\, \mathcal{U}\, \mathit{buildStator}) && (2)\\
&\wedge (\neg \mathit{assembleMotor}\, \mathcal{U}\, \mathit{buildRotor}) && (3)\\
&\wedge (\neg \mathit{assembleMotor}\, \mathcal{U}\, \mathit{buildInverter}) && (4)\\
&\wedge \Diamond(\mathit{staticTest} \vee \mathit{electricTest}) && (5)\\
&\wedge (\neg \mathit{electricTest}\, \mathcal{U}\, \mathit{assembleMotor}) && (6)\\
&\wedge (\neg \mathit{staticTest}\, \mathcal{U}\, \mathit{assembleMotor}) && (7)
\end{aligned}
$$

The $\mathcal{U}$-formulas (2), (3) and (4) prevent the assembly step

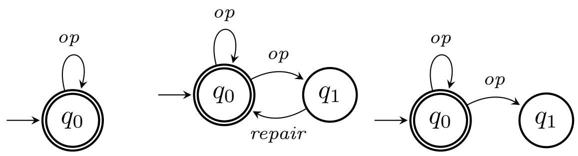


Figure 1: The infallible, breakable and irreparable services templates, respectively.

before all the components are available, and the final goal is specified by $\Diamond(\mathit{assembleMotor})$ (1). Formula (5) specifies that either $\mathit{staticTest}$ or $\mathit{electricTest}$ must be executed. Finally, formulas (6) and (7) specify that the tests must be executed after the assembly step. Note that we could omit the DECLARE conditions in the formula since they are forced at the semantic level. In this scenario, the services can be considered machines that produce specific components or human operators that perform the tests manually. We consider two types of services: *infallible* and *breakable* (Figure 1). The former has only one accepting state and supports one operation op; the latter, when executing the operation, can nondeterministically go into a "broken" state, from which a repair action is required to make it available again. In our experiments, we will have exactly one service for each process action.

Chip Production (CP). Here we consider a smart factory scenario in which the goal is to produce chips (Monti et al. 2023). In our simplified setting, the goal specification consists of a sequence of operations to be performed: cleaning the silicon wafers, thin film deposition, resist coating, etc. We consider three variants of this scenario, one for each service type. In particular, in the first variant, all services are of type infallible; in the second variant, they are of type breakable; and in the third variant, the services are all of type *irreparable*, i.e. they are like the breakable services except that they cannot be repaired. The goal formula has the form: The LTL$_f$ goal specification is a sequential goal with the following actions: $\mathit{cleaning}, \mathit{filmDeposition}$, $\mathit{resistCoating}$, $\mathit{exposure}$, $\mathit{development}$, $\mathit{etching}$, $\mathit{impuritiesImplantation}$, $\mathit{activation}$, $\mathit{resistStripping}$, $\mathit{assembly}, \mathit{testing}$, and $\mathit{packaging}$. The formula has the following form:

$$
\begin{aligned}
&\Diamond(\mathit{cleaning} \wedge \neg \mathit{filmDeposition} \wedge \neg \cdots \\
&\ \Diamond(\mathit{filmDeposition} \wedge \neg \cdots \\
&\ \ \Diamond(\mathit{resistCoating} \wedge \neg \cdots \\
&\ \ \ \Diamond(\mathit{exposure} \wedge \neg \cdots \\
&\ \ \ \ \Diamond(\mathit{development} \wedge \neg \cdots \\
&\ \ \ \ \ \Diamond(\mathit{etching} \wedge \neg \cdots \\
&\ \ \ \ \ \ \Diamond(\mathit{impuritiesImplantation} \wedge \neg \cdots \\
&\ \ \ \ \ \ \ \Diamond(\mathit{activation} \wedge \neg \cdots \\
&\ \ \ \ \ \ \ \ \Diamond(\mathit{resistStripping} \wedge \neg \cdots \\
&\ \ \ \ \ \ \ \ \ \Diamond(\mathit{assembly} \wedge \neg \cdots \\
&\ \ \ \ \ \ \ \ \ \ \Diamond(\mathit{testing} \wedge \neg \cdots \\
&\ \ \ \ \ \ \ \ \ \ \ \Diamond(\mathit{packaging} \wedge \neg \cdots))))))))))))
\end{aligned}
$$

Note that at each step we negate the presence of all the other Also, in this scenario, for all variants, we will have exactly one service for each process action. The sequence is iteratively truncated to evaluate the scalability of the planners with such instances.

Evaluation. We evaluated the MyND Planner (Mattmüller 2013), combined with the h_{ff} and $h_{\max}$ heuristics, over the PDDL files produced by our tool and the 4 available encoding of the TB translator. The metrics we considered are: pre-processing time, i.e., translation and SAS computation (TT), planning time (PT), the number of nodes expanded during the search (EN), and the policy size (PS). As benchmarks, we considered: e_i, with $i = 0, \ldots, 6$, are instances of the electric motor scenario with i builder services breakable and $6 - i$ infallible; c_i are the instances on the chip production scenario, with $i = 1, \ldots, 12$ being the length of the sequence of operations, with all services infallible; cn_i as c_i but with all services breakable; and cu_i as c_i but with all services irreparable. We set a timeout of 1000 seconds ($\approx$ 15 minutes).

Platform. The experiments have been run on an Ubuntu 22.04 machine, endowed with 12th Gen Intel(R) Core(TM) i7-1260P, with 16 CPU threads (12 cores) and 64GB of RAM. The JVM version is 14 for compatibility with MyND. The maximum RAM allocated for the JVM was 16GB.

Results. The results of our evaluation on all benchmarks, both using $h_{\max}$ and h_{ff}, are shown in Table 1, 2, 3 and 4. For better readability, we also show the plots for the PT metric for each scenario: Figure 2a, 2b, 2c, and 2d.

Regarding the Electic Motor scenario (Table 1), we observe that the PT of the OSA encoding is generally lower than the others, and in particular $h_{\max}$ is slightly better in PT than h_{ff}. The Simple encoding has comparable but slightly higher PT. The PG encoding has the PT higher than the PT in the Simple and OSA case, for all instances, sometimes higher by a factor of 4-5. The OSA+PG encoding was considerably worse than the other since the evaluation on many instances reached the timeout. Regarding the EN and PS metrics, the Simple encoding often gave a smaller number of expanded nodes and a smaller policy size, respectively, than the others. The TT was always lower than one second in all instances of this scenario.

In the deterministic Chip Production scenario (Table 2), we have that the OSA encoding, with respect to the PT metric, is better than the others, with no strict dominance between $h_{\max}$ and h_{ff}. The other encodings, from better to worse, were OSA+PG, PG and Simple for the h_{ff} heuristic, and Simple, PG and OSA+PG for the $h_{\max}$ heuristic. Regarding the EN metric, we observe mixed results: no approach dominates the other in all cases, although we observe that the EN in the OSA experiments for large instances was smaller than the other cases, while the PG encoding has a smaller number of EN for smaller instances. Finally, the PS metric was the smallest with the PG encoding for both heuristics.

In the nondeterministic Chip Production scenario (Table 3), the performances were quite worse than the deterministic case for all encodings and heuristics; the executions from cn8 on timed out. We noticed a certain advantage of using h_{ff} with the PG encoding, and $h_{\max}$ with the OSA encoding.

In the unsolvable Chip Production scenario (Table 4), the OSA was considerably better than the other encodings, with comparable performances between $h_{\max}$ and h_{ff}. For all Chip production variants, the TT metric was always lower than 1 second for Simple and OSA, and for PG and OSA+PG it was increasing with the length of the formula by a factor of 2 at each increment.

Electric motor scenario

	Simple				OSA				PG				OSA+PG			
	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS
$h_{\max}$																
e0	**0.1209**	15.327	31083	69	0.1286	**9.393**	**20476**	180	0.1474	16.689	38752	**66**	0.1335	248.28	766448	171
e1	**0.1128**	15.831	41783	109	0.1204	**14.992**	**29878**	270	0.1367	24.389	60349	**104**	0.1329	765.812	1499975	261
e2	**0.1165**	18.34	**65967**	**121**	0.1178	**16.134**	72043	331	0.1411	53.388	129246	209	0.1619	—	—	—
e3	**0.1001**	48.522	211105	**158**	0.1225	**19.077**	**150636**	601	0.141	108.081	210708	302	0.1411	—	—	—
e4	**0.1002**	69.688	247760	391	0.1441	**20.083**	**155162**	722	0.1713	94.505	247368	**269**	0.1432	—	—	—
e5	**0.1004**	27.17	**95087**	**265**	0.1272	**20.958**	161513	722	0.1525	94.714	247393	269	0.1435	—	—	—
e6	**0.1059**	**35.16**	**133480**	**244**	0.131	38.321	241633	963	0.1732	98.925	248018	274	0.1558	—	—	—
h_{ff}																
e0	**0.1157**	12.096	**15863**	69	0.1382	**9.423**	20047	180	0.1235	18.496	34196	**66**	0.1306	263.119	766551	171
e1	**0.1133**	**14.028**	**27876**	**98**	0.131	15.388	42631	271	0.1333	31.682	65180	108	0.195	743.99	1472364	231
e2	0.1164	20.893	**73624**	**124**	0.1386	**18.215**	87854	482	0.1323	41.886	102093	161	**0.1119**	—	—	—
e3	**0.1129**	31.031	**103592**	**151**	0.135	**23.953**	179959	903	0.1277	100.598	150256	334	0.1156	—	—	—
e4	**0.1159**	63.091	**145993**	**226**	0.134	**28.263**	209560	1294	0.142	152.991	174310	474	0.1218	—	—	—
e5	**0.1219**	77.981	256946	**399**	0.1411	**30.147**	219430	1294	0.1336	154.35	**174436**	474	0.1222	—	—	—
e6	**0.1198**	**55.656**	**136193**	**175**	0.1409	55.746	290153	2307	0.1382	196.674	199953	631	0.1202	—	—	—

Table 1: Evaluation metrics over the Electric Motor scenario, using MyND with $h_{\max}$ and h_{ff} heuristics.

Chip Production scenario (deterministic)

	Simple				OSA				PG				OSA+PG			
	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS
$h_{\max}$																
c1	**0.0782**	**0.101**	14	12	0.0821	2.899	18	18	0.0863	0.123	**9**	**9**	0.0997	3.796	17	15
c2	**0.1014**	**5.387**	68	24	0.1056	5.635	96	48	0.1099	6.694	**59**	**21**	0.1087	8.734	166	44
c3	**0.1169**	**7.922**	205	40	0.1191	9.434	353	96	0.1339	8.512	**194**	**37**	0.1352	9.21	918	91
c4	0.1416	13.906	530	60	**0.1408**	14.001	895	170	0.1685	**12.02**	**508**	**57**	0.1778	14.644	3650	166
c5	**0.1432**	16.808	1279	84	0.1578	18.951	2186	276	0.2473	19.446	**1253**	**81**	0.2517	**15.415**	10866	272
c6	**0.1464**	24.73	2988	112	0.1898	**18.498**	4508	420	0.395	23.075	**2955**	**109**	0.3824	21.181	25186	415
c7	**0.1785**	31.907	6857	144	0.2342	**26.286**	8821	608	0.6999	31.288	**6830**	**141**	0.6851	27.604	49079	600
c8	**0.2599**	46.196	15542	180	0.2852	**17.158**	**14872**	846	1.335	44.04	15509	**177**	1.3415	46.031	96456	843
c9	**0.324**	358.285	34867	220	0.3392	**44.613**	**25333**	1140	2.7121	68.722	34820	**217**	2.5954	71.578	160717	1129
c10	**0.3786**	—	—	—	0.4823	**38.293**	**38875**	1496	4.9345	219.81	77447	**261**	4.9841	121.717	282685	1492
c11	**0.6279**	—	—	—	0.7819	**37.336**	**59949**	1920	9.1422	—	—	—	8.1315	255.152	456012	**1906**
c12	**0.8098**	—	—	—	0.9784	**44.558**	**87472**	**2418**	—	—	—	—	—	—	—	—
h_{ff}																
c1	**0.0809**	**0.095**	13	12	0.0822	2.314	18	18	0.1018	0.123	**9**	**9**	0.1012	3.261	19	15
c2	0.1071	**5.393**	63	24	0.1089	5.861	102	48	**0.107**	6.505	**38**	**21**	0.116	8.679	110	44
c3	**0.1166**	**7.886**	195	40	0.1232	8.686	365	96	0.1227	8.911	**145**	**37**	0.1376	9.03	679	91
c4	**0.1311**	13.788	506	60	0.1434	13.375	936	170	0.1602	**12.233**	**426**	**57**	0.1713	14.97	3645	166
c5	**0.151**	**16.866**	**666**	84	0.1637	18.314	2186	276	0.2421	18.176	1088	**81**	0.2361	18.019	11569	272
c6	**0.1543**	24.478	**1332**	112	0.1872	**17.98**	4586	420	0.3807	22.758	2665	**109**	0.386	43.771	28600	415
c7	**0.1793**	30.635	**2017**	144	0.2209	**24.704**	8853	608	0.6707	32.741	6321	**141**	0.7167	158.009	57782	600
c8	**0.221**	33.814	**1910**	180	0.2784	**17.287**	14894	846	1.3139	49.424	14599	**177**	1.3502	772.85	114243	843
c9	**0.3058**	87.657	33855	220	0.3666	**45.414**	**25484**	1140	2.8215	106.995	33132	**217**	2.594	—	—	—
c10	**0.3808**	162.768	58684	264	0.5243	**43.201**	**38910**	1496	4.9545	492.077	74182	**261**	4.9146	—	—	—
c11	**0.6136**	611.468	116315	**312**	0.8102	**40.923**	**60285**	1920	8.3054	—	—	—	9.6083	—	—	—
c12	**0.8079**	69.42	**6185**	**364**	0.9845	**52.27**	87694	2418	—	—	—	—	—	—	—	—

Table 2: Evaluation metrics over the Chip Production scenario, using MyND with $h_{\max}$ and h_{ff} heuristics.

Chip Production scenario (nondeterministic)

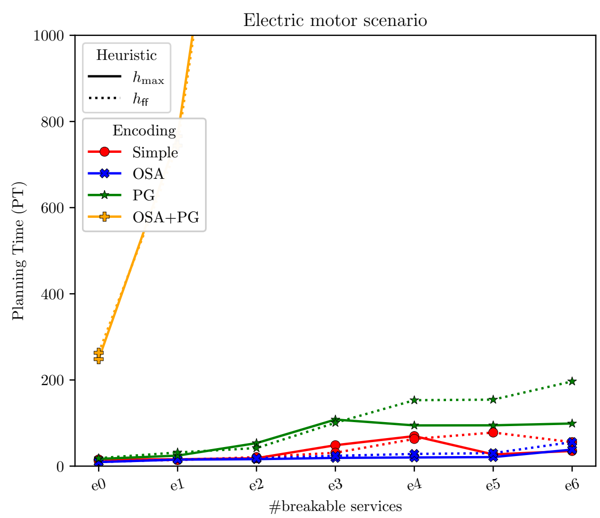
	Simple				OSA				PG				OSA+PG			
	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS
$h_{\max}$																
cn1	0.1007	**0.191**	23	19	0.1031	3.839	28	28	**0.1004**	0.267	**17**	**15**	0.1053	5.971	29	24
cn2	**0.1051**	**6.822**	**254**	64	0.1309	9.994	268	97	0.1309	7.874	259	**58**	0.1232	9.252	704	124
cn3	0.1604	12.961	**1044**	**134**	0.1932	**9.34**	1607	217	**0.1475**	9.999	1110	143	0.168	12.59	3666	211
cn4	**0.1541**	18.008	6902	296	0.1584	**9.479**	14084	547	0.1822	13.294	**4569**	**240**	0.2052	12.321	38205	950
cn5	**0.1542**	20.166	**17091**	**416**	0.1605	**17.559**	27271	783	0.2568	22.554	18097	884	0.2523	108.134	345650	1376
cn6	0.2235	98.258	83019	**661**	**0.2137**	**60.737**	204543	3122	0.4269	70.874	**68624**	862	0.4109	222.071	474593	2871
cn7	**0.1957**	—	—	—	0.2394	**466.19**	736566	3725	0.7276	734.046	**249752**	**2301**	0.8348	—	—	—
cn8	**0.2034**	—	—	—	0.2464	—	—	—	1.1816	—	—	—	1.1961	—	—	—
cn9	**0.2842**	—	—	—	0.3125	—	—	—	2.8293	—	—	—	2.3139	—	—	—
cn10	**0.4129**	—	—	—	0.4446	—	—	—	5.5838	—	—	—	4.4328	—	—	—
cn11	**0.683**	—	—	—	0.7297	—	—	—	—	—	—	—	8.3411	—	—	—
cn12	0.9034	—	—	—	**0.8845**	—	—	—	—	—	—	—	—	—	—	—
h_{ff}																
cn1	0.1047	**0.185**	20	19	**0.102**	3.798	28	28	0.1028	0.246	**16**	**15**	0.1025	5.461	29	24
cn2	**0.1155**	7.525	236	64	0.1224	10.009	318	97	0.1194	**6.999**	**99**	**58**	0.1374	10.07	397	124
cn3	0.1502	12.675	1321	174	0.1516	**8.37**	2280	339	**0.1478**	10.51	**511**	**164**	0.1691	13.725	5620	403
cn4	**0.1453**	16.249	6311	428	0.1557	**9.398**	10004	1097	0.1736	13.005	**2240**	**412**	0.1686	19.813	34055	1238
cn5	**0.1469**	**20.081**	32406	1011	0.1656	20.559	37620	3148	0.2425	21.052	**8961**	**976**	0.253	151.588	139295	3318
cn6	**0.1731**	51.953	145182	**282**	0.2885	62.837	133207	8800	0.3863	**41.794**	**35282**	2236	0.4306	—	—	—
cn7	**0.1585**	423.237	547586	**485**	0.223	307.16	420330	22489	0.7753	**216.97**	**139259**	5016	0.6069	—	—	—
cn8	**0.2156**	—	—	—	0.4938	—	—	—	1.1469	—	—	—	2.3898	—	—	—
cn9	**0.2807**	—	—	—	0.6468	—	—	—	2.3548	—	—	—	4.6995	—	—	—
cn10	**0.4334**	—	—	—	0.9253	—	—	—	4.6169	—	—	—	7.3848	—	—	—
cn11	**0.6717**	—	—	—	1.4502	—	—	—	8.2353	—	—	—	—	—	—	—
cn12	**0.8957**	—	—	—	1.7755	—	—	—	—	—	—	—	—	—	—	—

Table 3: Evaluation metrics over the Chip Production scenario (nondeterministic), using MyND with $h_{\max}$ and h_{ff} heuristics.

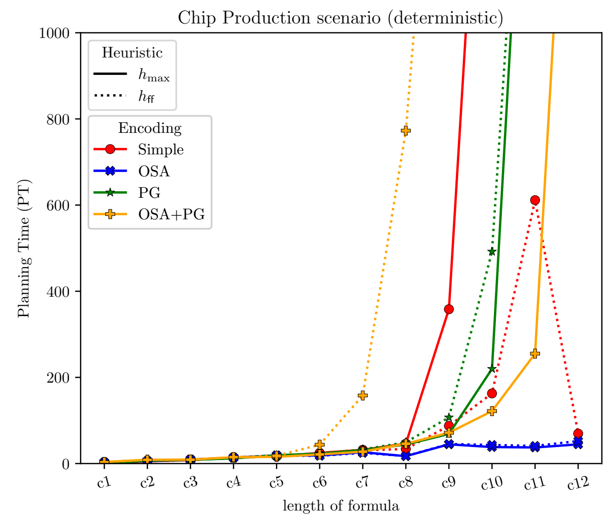
Chip Production scenario (unsolvable)

	Simple				OSA				PG				OSA+PG			
	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS	TT	PT	EN	PS
$h_{\max}$																
cu1	0.0815	**0.138**	N/A	N/A	**0.0808**	2.274	N/A	N/A	0.0825	0.263	N/A	N/A	0.0856	2.274	N/A	N/A
cu2	**0.0868**	5.67	N/A	N/A	0.1128	6.11	N/A	N/A	0.1234	8.511	N/A	N/A	0.1138	**5.515**	N/A	N/A
cu3	0.1308	8.275	N/A	N/A	**0.1289**	12.498	N/A	N/A	0.1334	8.685	N/A	N/A	0.1344	**6.4**	N/A	N/A
cu4	0.147	11.891	N/A	N/A	**0.1438**	12.431	N/A	N/A	0.1676	14.005	N/A	N/A	0.1791	**10.672**	N/A	N/A
cu5	**0.1454**	19.967	N/A	N/A	0.1643	18.282	N/A	N/A	0.2414	**17.811**	N/A	N/A	0.2591	19.81	N/A	N/A
cu6	**0.1508**	25.994	N/A	N/A	0.1959	**9.483**	N/A	N/A	0.3856	22.212	N/A	N/A	0.3862	12.053	N/A	N/A
cu7	**0.1661**	33.268	N/A	N/A	0.2245	25.889	N/A	N/A	0.6819	31.839	N/A	N/A	0.6761	**23.122**	N/A	N/A
cu8	**0.249**	47.649	N/A	N/A	0.2604	**37.509**	N/A	N/A	1.3084	43.185	N/A	N/A	1.3242	39.476	N/A	N/A
cu9	**0.3033**	300.892	N/A	N/A	0.3391	**47.825**	N/A	N/A	2.6881	68.552	N/A	N/A	2.6304	74.131	N/A	N/A
cu10	**0.3736**	—	N/A	N/A	0.4997	**29.513**	N/A	N/A	4.9509	228.683	N/A	N/A	5.3943	172.825	N/A	N/A
cu11	**0.747**	—	N/A	N/A	0.8026	**41.257**	N/A	N/A	9.6311	—	N/A	N/A	9.1791	—	N/A	N/A
cu12	1.4366	—	N/A	N/A	**1.3352**	**84.423**	N/A	N/A	—	—	N/A	N/A	—	—	N/A	N/A
h_{ff}																
cu1	**0.0804**	**0.18**	N/A	N/A	0.0825	2.478	N/A	N/A	0.083	0.277	N/A	N/A	0.0818	2.997	N/A	N/A
cu2	**0.0844**	**5.607**	N/A	N/A	0.1262	6.363	N/A	N/A	0.1121	6.3	N/A	N/A	0.1226	6.904	N/A	N/A
cu3	**0.1154**	8.139	N/A	N/A	0.125	12.306	N/A	N/A	0.1288	9.308	N/A	N/A	0.1506	**6.346**	N/A	N/A
cu4	**0.1362**	11.869	N/A	N/A	0.1414	12.801	N/A	N/A	0.1682	13.606	N/A	N/A	0.1805	**11.768**	N/A	N/A
cu5	**0.1554**	20.156	N/A	N/A	0.1626	**18.573**	N/A	N/A	0.2483	19.296	N/A	N/A	0.2338	20.661	N/A	N/A
cu6	**0.159**	25.96	N/A	N/A	0.1861	**9.582**	N/A	N/A	0.3819	22.5	N/A	N/A	0.3785	29.088	N/A	N/A
cu7	**0.1845**	32.298	N/A	N/A	0.2171	**27.397**	N/A	N/A	0.6625	31.1	N/A	N/A	0.679	126.688	N/A	N/A
cu8	**0.2218**	39.144	N/A	N/A	0.261	**37.752**	N/A	N/A	1.3137	50.682	N/A	N/A	1.3392	612.567	N/A	N/A
cu9	**0.3244**	82.661	N/A	N/A	0.3365	**49.188**	N/A	N/A	2.7307	113.082	N/A	N/A	2.6442	—	N/A	N/A
cu10	**0.3806**	173.005	N/A	N/A	0.4965	**38.134**	N/A	N/A	5.0478	598.16	N/A	N/A	—	—	N/A	N/A
cu11	0.8236	736.008	N/A	N/A	**0.8122**	**46.241**	N/A	N/A	8.7799	—	N/A	N/A	8.9338	—	N/A	N/A
cu12	1.6415	404.779	N/A	N/A	**0.9672**	**71.658**	N/A	N/A	—	—	N/A	N/A	—	—	N/A	N/A

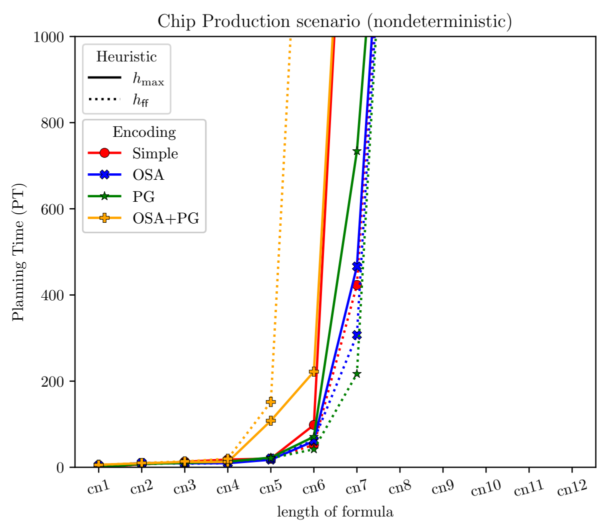
Table 4: Evaluation metrics over the Chip Production scenario (unsolvable), using MyND with $h_{\max}$ and h_{ff} heuristics.



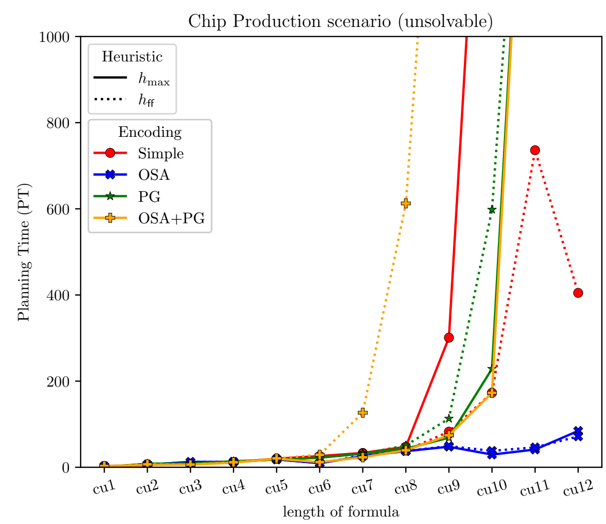
(a) Planning time (PT) metric for the Electric Motor scenario.



(b) Planning time (PT) metric for the Chip Production scenario (deterministic).



(c) Planning time (PT) metric for the Chip Production scenario (nondeterministic).



(d) Planning time (PT) metric for the Chip Production scenario (unsolvable).

References

De Giacomo, G.; Favorito, M.; Leotta, F.; Mecella, M.; Monti, F.; and Silo, L. 2023. AIDA: A Tool for Resiliency in Smart Manufacturing. In *CAiSE*.

De Giacomo, G.; Favorito, M.; Leotta, F.; Mecella, M.; Monti, F.; and Silo, L. 2023. AIDA: A Tool for Resiliency in Smart Manufacturing. In *CAiSE Forum*, volume 477 of *Lecture Notes in Business Information Processing*, 112–120. Springer.

De Giacomo, G.; and Vardi, M. Y. 2015. Synthesis for LTL and LDL on Finite Traces. In *IJCAI*.

Mattmüller, R. 2013. *Informed progression search for fully observable nondeterministic planning = Informierte Vorwärtssuche für nichtdeterministisches Planen unter vollständiger Beobachtbarkeit*. Ph.D. thesis.

Monti, F.; Silo, L.; Leotta, F.; and Mecella, M. 2023. On the Suitability of AI for Service-based Adaptive Supply Chains in Smart Manufacturing. In *ICWS*.

Torres, J.; and Baier, J. A. 2015. Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *IJCAI*.