On the Expressive Power of Planning Formalisms in Conjunction with LTL

Songtuan Lin, Pascal Bercher

School of Computing, College of Engineering and Computer Science
The Australian National University
{songtuan.lin, pascal.bercher}@anu.edu.au

Abstract

Linear Temporal Logic (LTL) has been widely employed in various planning formalisms, e.g., in the $\mathcal{STRIPS}$ formalism, in order to specify constraints over state trajectories in a planning problem. In this paper, we investigate the expressive power of two planning formalisms in conjunction with LTL that are most commonly seen in non-hierarchical planning and hierarchical planning respectively, namely the $\mathcal{STRIPS}$ formalism and the Hierarchical Task Network ($\mathcal{HTN}$) formalism. We do so by interpreting the set of all solutions to a planning problem as a formal language and comparing it with other formal ones, e.g., star-free languages. Our results provide an in-depth insight into the theoretical properties of the investigated planning formalisms and henceforth explore the common structure shared by solutions to planning problems in certain planning formalisms.

Introduction

Linear Temporal Logic (LTL) (Pnueli 1977) is a powerful tool for model checking, e.g., hardware and software verification (Huth and Ryan 2000; Baier and Katoen 2008). Recently, it has also started being employed in the field of automatic planning due to its ability of imposing constraints over state trajectories in a planning problem. LTL is usually exploited to describe so called *temporal extended goals* (Bacchus and Kabanza 1996; De Giacomo and Vardi 1999) in a planning formalism which specify what properties must hold while a plan is executing, e.g., see the work by Fox and Long (2003) on how this feature is adapted in the Planning Domain Definition Language (PDDL) (Ghallab et al. 1998).

In this paper, we are concerned with the expressive power of the $\mathcal{STRIPS}$ planning formalism (Fikes and Nilsson 1971) and of the Hierarchical Task Network ($\mathcal{HTN}$) planning formalism (Erol, Hendler, and Nau 1996; Geier and Bercher 2011; Bercher, Alford, and Höller 2019) in conjunction with LTL. The former one is a widely used formalism for non-hierarchical planning and the latter for hierarchical planning. Although there are works toward the expressiveness of the $\mathcal{STRIPS}$ and $\mathcal{HTN}$ formalism (Höller et al. 2014, 2016) and of LTL (Thomas 1997; Diekert and Gastin 2008; De Giacomo and Vardi 2013), to our best knowledge, no efforts have been devoted to study their combinations. We follow the idea by Höller et al. (2014, 2016) by which the set of all solutions to a planning problem (in a certain formalism) is interpreted as a formal language. Just as Höller et al. (2014, 2016), we compare such a language to other formal languages, to establish the tight upper bound for the expressiveness of the respective formalism. We visualize the comparison results in Fig. 1 by placing the formal languages described by each planning formalism studied into the Chomsky hierarchy (Chomsky 1956).

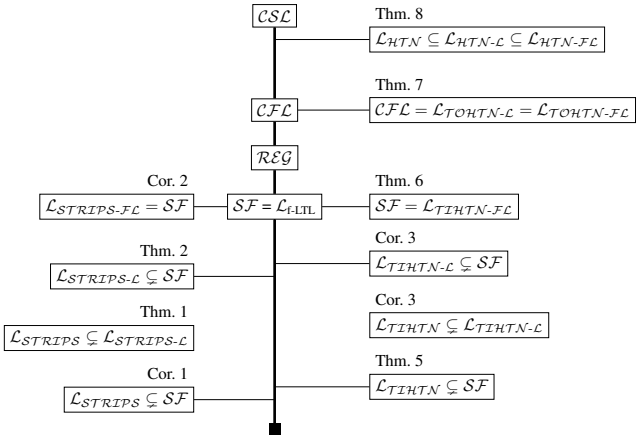

Figure 1: The positions of the languages of the planning formalisms studied in this paper in the Chomsky hierarchy. The notation $\mathcal{L}_X$ with X being a planning formalism refers to the formal languages described by the formalism X. $\mathcal{CSL}$, $\mathcal{CFL}$, $\mathcal{REG}$, and $\mathcal{SF}$ refer to context-sensitive, context-free, regular, and star-free languages respectively. $\mathcal{L}_{\text{f-LTL}}$ refers to the languages of *finite*-LTL. The languages of LTL are not included here, because those are equivalent to star-free languages of *infinite length*, whereas the languages listed here are all of *finite length*.

Another approach for evaluating and comparing the expressive power of different planning formalisms is to check whether there exists a polynomial reduction from an arbitrary planning problem $\mathcal{P}$ in one formalism into a planning problem $\mathcal{P}'$ in another formalism. Such reductions only require that $\mathcal{P}$ has a solution *iff* $\mathcal{P}'$ has one. Solutions to $\mathcal{P}$ and that to $\mathcal{P}'$ may thus have significantly different structures, or

some solutions might even be lost. In contrast, the evaluation approach used here by Höller et al. (2014, 2016) requires that a solution to a planning problem in one planning formalism must also be a solution (*resp.* a word) to a planning problem in another formalism (*resp.* another formal grammar). For more details about the comparison between these two evaluation approaches, see the work by Höller et al. (2016).

The motivation for our work is twofold. Firstly, it is to get an in-depth insight into the theoretical properties of the planning formalisms under investigation of which the expressiveness, i.e., the class of formal languages that can be described by a planning formalism, is the core aspect. More concretely, regarding the solution set of a planning problem as a language establishes a bridge from planning theory to formal language theory which itself has a connection to complexity theory (e.g., see the work by Hopcroft, Motwani, and Ullman (2007)) and logic (e.g., see the work by Thomas (1997), by Camacho et al. (2019), and by Pinchinat, Rubin, and Schwarzentruber (2022)). Additionally, expressiveness results can be exploited by transferring known properties of well-studied formal grammars to planning languages that are more expressive than them, which can further improve our understanding of those planning formalisms. Secondly, for practical implications, knowledge of expressiveness of each planning formalism helps choosing a suitable one to model a given problem at hand to make sure the problem can even be modeled in the first place. For instance, suppose that the solution set of a problem to be modeled is equivalent to a context-free language, then, by our result, we know that it cannot be modeled as a classical planning problem while preserving the same solution set.

Background

In this section, we present the concepts and notations that will be referred to throughout the paper. We start by giving a short review for formal languages. Afterwards, we will introduce the syntax and semantics of LTL as well as its variant finite-LTL (f-LTL). Lastly, we will formalize various planning formalisms including both hierarchical and non-hierarchical ones which incorporate LTL/f-LTL.

Formal Languages

The basis for defining formal languages is the concept of alphabets. An alphabet is a *finite* non-empty set whose elements are called symbols. Given an alphabet Σ, the notation Σ^i with $i \in \mathbb{N}_0$ ($\mathbb{N}_0 = \mathbb{N} \cup \{0\}$) refers to the set of sequences of length i which consist of symbols in Σ, i.e., $\Sigma^i = \{\langle a_1 \cdots a_i \rangle \mid 1 \leq j \leq i, a_j \in \Sigma\}$. The set Σ^* is defined as the union of Σ^i for all $i \geq 0$, i.e., $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$. An element $\omega \in \Sigma^*$ is called a word. Particularly, $\{\varepsilon\} = \Sigma^0$ where ε refers to the empty word.

A formal language over an alphabet Σ is a subset (finite or infinite) of Σ^*. One crucial class of languages which we will mostly deal with in this paper is star-free languages. A language over an alphabet Σ is star-free if it can be constructed in terms of the *languages* $\emptyset$ (the empty language), $\{\varepsilon\}$ (the language of the empty word), and all languages $\{a\}$ with $a \in \Sigma$ (the languages consisting of one word being a singleton symbol in Σ) by using concatenation and binary operations, i.e., union, intersection, and complement (with respect to Σ^*) finitely many times (Thomas 1997).

Beyond star-free languages, there are regular languages, context-free languages, context-sensitive languages, and recursively enumerable languages, each of which is a *strict* superset of previous ones. Those languages except star-free ones form the Chomsky hierarchy (Chomsky 1956), which has evolved as the standard measurement of the expressiveness of a language. For instance, by recognizing that star-free languages are a strict subset of regular languages (Mateescu and Salomaa 1997), we know that star-free languages have less expressive power than regular ones.

Note that apart from the formal languages described here which consist of words of *finite length*, there are languages where each word is of *infinite length*. Such languages are called ω-languages. Those are two disjoint classes of languages. In this paper, our primary concern is formal languages of finite words, because we will interpret the set of all solutions to a planning problem in a planning formalism as such a language, as mentioned in the introduction. However, we will also encounter ω-languages when we confront the language described by an LTL formula. Hence, we emphasize their difference here for clarity.

Linear Temporal Logic

Next we present the syntax and semantics of Linear Temporal Logic. We give the syntax of LTL in BNF, which is an adaption of the one given by Pnueli (1977) and is defined over a set of propositions P.

$$\varphi = \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \,\mathtt{U}\, \varphi_2$$

where $\top$ stands for *true*, $p \in P$ is a proposition, φ_1 and φ_2 are LTL formulae, $\bigcirc$ stands for *next* meaning informally that φ should be satisfied in the *next* state, and $\mathtt{U}$ stands for *until* meaning informally that φ_1 holds *until* φ_2 is satisfied.

The semantics of LTL is defined in terms of an *infinite* state trace $\pi = \langle s_1 s_2 \cdots \rangle$ where $s_i \subseteq P$ is called a state for each $i \geq 1$. We follow the convention to use π_i, $i \geq 1$, to refer to the subsequence $\langle s_i s_{i+1} \cdots \rangle$ of states. For $i \geq 1$,

- $\pi_i \vDash \top$,
- $\pi_i \vDash p$ *iff* $p \in s_i$,
- $\pi_i \vDash \neg\varphi$ *iff* $\pi_i \nvDash \varphi$,
- $\pi_i \vDash \bigcirc\varphi$ *iff* $\pi_{i+1} \vDash \varphi$,
- $\pi_i \vDash \varphi_1 \wedge \varphi_2$ *iff* $\pi_i \vDash \varphi_1 \wedge \pi_i \vDash \varphi_2$, and
- $\pi_i \vDash \varphi_1 \,\mathtt{U}\, \varphi_2$ *iff* there exists a $j \geq i$ such that $\pi_j \vDash \varphi_2$ and $\pi_k \vDash \varphi_1$ for all $i \leq k < j$.

Note that the set $\{\top, \neg, \bigcirc, \mathtt{U}\}$ of connectives is adequate for LTL (Huth and Ryan 2000) because others, i.e., $\bot$ (*false*), $\vee$ (logical or), $\diamond$ (eventually), and $\Box$ (always), can be formulated in terms of those:

- $\bot \equiv \neg\top$,
- $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$
- $\diamond\varphi \equiv \top \,\mathtt{U}\, \varphi$
- $\Box\varphi \equiv \neg(\diamond(\neg\varphi))$.

Intuitively, $\bot$ stands for *false*, $\vee$ is the *logical or* operator, $\diamond\varphi$ means that φ will *eventually* be satisfied, and $\Box\varphi$ means that φ is *always* satisfied.

The language $\mathcal{L}(\varphi)$ of an LTL formula φ over a proposition set P is defined over the alphabet $\Sigma_P = 2^P$ and consists of all infinite state sequences π such that $\pi \vDash \varphi$, i.e., $\mathcal{L}(\varphi) = \{\pi \mid \pi \vDash \varphi\}$ (Baier and Katoen 2008). Regarding a set Σ_P of *states* as an alphabet here might seem unusual, yet

this is well defined because any non-empty finite set can be seen as an alphabet. Clearly, $\mathcal{L}(\varphi)$ is an ω-*language*.

A variant of LTL called finite-LTL (f-LTL) which is tailored to *finite* state sequences are also introduced, e.g., see the works by Bienvenu, Fritz, and McIlraith (2006), by Baier and McIlraith (2006a, 2006b), and by De Giacomo and Vardi (2013). The syntax of f-LTL is identical to that of LTL. The semantics are defined over a *finite* state sequence $\pi = \langle s_1 \cdots s_n \rangle$ ($n \in \mathbb{N}$): For $1 \leq i \leq n$,

- $\pi_i \vDash \top$,
- $\pi_i \vDash p$ *iff* $p \in s_i$,
- $\pi_i \vDash \neg\varphi$ *iff* $\pi_i \nvDash \varphi$,
- $\pi_i \vDash \varphi_1 \wedge \varphi_2$ *iff* $\pi_i \vDash \varphi_1 \wedge \pi_i \vDash \varphi_2$,
- $\pi_i \vDash \bigcirc\varphi$ *iff* $i < n$ and $\pi_{i+1} \vDash \varphi$, and
- $\pi_i \vDash \varphi_1 \,\mathsf{U}\, \varphi_2$ *iff* there exists a j with $i \leq j \leq n$ such that $\pi_j \vDash \varphi_2$, and for each $i \leq k < j$, $\pi_k \vDash \varphi_1$.

The crucial difference between the semantics of f-LTL and that of LTL is that, apart from being given over a finite state trace and an infinite one, the operator $\bigcirc$ in f-LTL imposes an additional constraint that i must be strictly smaller than the length of the state trace. Such a constraint gives f-LTL the power to express that the state sequence has reached its *end*. The expression for this, written $\odot = \neg(\bigcirc\top)$, is satisfied *iff* $i = n$ (De Giacomo and Vardi 2013).

The formal language $\mathcal{L}(\varphi)$ of an f-LTL formula φ over a proposition set P is again defined over the alphabet $\Sigma_P = 2^P$ and $\mathcal{L}(\varphi) = \{\pi \mid \pi \vDash \varphi\}$ (De Giacomo and Vardi 2013). In contrast to LTL, $\mathcal{L}(\varphi)$ consists of words of *finite length*.

Non-hierarchical Planning Formalisms

We move on now to introduce the planning formalisms that will be studied in this paper. In this section, we focus on non-hierarchical planning formalisms. We begin by giving the definition of the $\mathcal{STRIPS}$ formalism.

Definition 1. A $\mathcal{STRIPS}$ planning problem $\mathcal{P}$ is a tuple $(\mathcal{F}, \mathcal{A}, \delta, s_I, g)$ where $\mathcal{F}$ is a set of propositions, $\mathcal{A}$ is a set of action names, $\delta : \mathcal{A} \mapsto 2^{\mathcal{F}} \times 2^{\mathcal{F}} \times 2^{\mathcal{F}}$ is a function mapping action names to their preconditions and effects, $s_I \in 2^{\mathcal{F}}$ is the initial state, and $g \subseteq \mathcal{F}$ is the goal description.

The purpose of introducing the function δ and emphasizing exclusively action names is to make the comparison with formal languages more straightforward for which we will have more discussion later. We use the term *action* and *action name* interchangeably throughout this paper to ease the notation. The function δ maps each action $a \in \mathcal{A}$ to its preconditions, add list and delete list, written $\delta(a) = (prec(a), add(a), del(a))$. The preconditions of an action determine whether it can be applied in some state. Formally, an action a is said to be applicable in a state $s \in 2^{\mathcal{F}}$ if $prec(a) \subseteq s$. The add list together with the delete list of an action is called the effects of this action, which specify how it changes a state once it is executed. A state s' is the consequence of applying an action a in a state s, written $s \rightarrow_a s'$, if a is applicable in s, and $s' = (s \backslash del(a)) \cup add(a)$. A state sequence $\pi = \langle s_0 \cdots s_n \rangle$ is the consequence of applying a sequence of actions $\overline{a} = \langle a_1 \cdots a_n \rangle$ in a state s if $s_0 = s$, and for each $1 \leq i \leq n$, $prec(a_i) \subseteq s_{i-1}$ and $s_i = (s_{i-1} \backslash del(a_i)) \cup add(a_i)$. We also write $s \rightarrow_{\overline{a}}^* s_n$ as an abbreviation of this process.

A solution to a planning problem $\mathcal{P} = (\mathcal{F}, \mathcal{A}, \delta, s_I, g)$ is an action sequence $\overline{a}$ such that $s_I \rightarrow_{\overline{a}}^* s'$ for some s' with $s' \in 2^{\mathcal{F}}$ and $g \subseteq s'$.

One way to incorporate LTL into the $\mathcal{STRIPS}$ formalism is to replace the goal description in a $\mathcal{STRIPS}$ planning problem with an LTL formula. Such goals are called temporally extended goals (TEGs) (Bacchus and Kabanza 1996; De Giacomo and Vardi 1999). We denote this extended planning formalism as $\mathcal{STRIPS}$-$\mathcal{L}$.

Definition 2. A $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem $\mathcal{P}$ is a tuple $(\mathcal{F}, \mathcal{A}, \delta, s_I, g)$ where $\mathcal{F}$, $\mathcal{A}$, δ, and s_I are defined in the same way as that in a $\mathcal{STRIPS}$ planning problem, and g is an LTL formula.

An action sequence $\overline{a}$ is a solution to a $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem $\mathcal{P}$ if the state sequence which is the consequence of applying $\overline{a}$ in the initial state of $\mathcal{P}$ satisfies the TEG, i.e., the LTL formula g. The problem that arises here is that the semantics of LTL are defined over infinite state sequences whereas a state sequence resulting from an action sequence is finite in standard planning processes. A common way to address this problem is to extend a state sequence in $\mathcal{STRIPS}$-$\mathcal{L}$ planning to an infinite one by repeating its last state indefinitely (Bacchus and Kabanza 1996; Edelkamp 2006). For instance, the state sequence $\langle s_1 \cdots s_n \rangle$ is extended to $\langle s_1 \cdots s_n s_n \cdots \rangle$.

A more sophisticated way to make the $\mathcal{STRIPS}$ formalism be able to employ TEGs without artificially extending state sequences is writing TEGs in f-LTL. We refer to this formalism as $\mathcal{STRIPS}$-$\mathcal{FL}$.

Definition 3. A $\mathcal{STRIPS}$-$\mathcal{FL}$ planning problem $\mathcal{P}$ is a tuple $(\mathcal{F}, \mathcal{A}, \delta, s_I, g)$ where $\mathcal{F}$, $\mathcal{A}$, δ, and s_I are defined in the same way as those in a $\mathcal{STRIPS}$ planning problem, and g is an f-LTL formula. Similarly, a solution to $\mathcal{P}$ is an action sequence which leads to a state sequence satisfying g.

Hierarchical Planning Formalisms

In this section, we introduce the hierarchical planning formalisms that will be studied in this paper. Hierarchical planning formalisms extend non-hierarchical ones in the way that a plan (i.e., an action sequence) is obtained from iteratively refining abstract tasks (also called compound tasks). We present here one standard hierarchical planning formalism called the Hierarchical Task Network ($\mathcal{HTN}$) formalism, which adapts the work by Geier and Bercher (2011) and by Bercher, Alford, and Höller (2019).

Definition 4. A task network tn is a tuple $(T, \prec, \alpha)$ where T is a set of symbols called task identifiers, $\prec \subseteq T \times T$ is a set of ordering constraints (i.e., partial order relations), and α is a function mapping each task identifier to its task name.

Two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ are called isomorphic, written $tn \cong tn'$, if there exists a bijective mapping $\varrho : T \mapsto T'$ such that for all $t_1, t_2 \in T$, $(t_1, t_2) \in \prec$ *iff* $(\varrho(t_1), \varrho(t_2)) \in \prec'$, and $\alpha(t_1) = \alpha'(\varrho(t_1))$.

A linearization of a task network $tn = (T, \prec, \alpha)$ is a total order of T which respects $\prec$. Note that a linearization $\overline{tn}$ of tn is a sequence of task identifiers. We write $\alpha(\overline{tn})$ to refer to

the sequence of task names represented by $\overline{tn}$. For instance, if $\overline{tn} = \langle t_1 \cdots t_n \rangle$, then $\alpha(\overline{tn}) = \langle \alpha(t_1) \cdots \alpha(t_n) \rangle$.

In $\mathcal{HTN}$ planning, task names are categorized as being compound task names and primitive task names. A primitive task name is identical to an action name defined in non-hierarchical planning formalisms, and hence we use the terms primitive task and action interchangeably. A compound task name c is refined into a task network tn by a method $m = (c, tn)$. Formally, the concept of task networks is employed in the $\mathcal{HTN}$ formalism as follows.

Definition 5. An $\mathcal{HTN}$ planning problem is a tuple $(\mathcal{D}, c_I, s_I, g)$. $\mathcal{D} = (\mathcal{F}, \mathcal{A}, \mathcal{C}, \mathcal{M}, \delta)$ is called the domain of the planning problem where $\mathcal{F}$ is a set of propositions, $\mathcal{A}$ is a set of action names, $\mathcal{C}$ is a set of compound task names, $\mathcal{M}$ is a set of methods, and δ is a function mapping each action name to its preconditions and effects, as defined in non-hierarchical formalisms. $c_I \in \mathcal{C}$ is the initial task name. $s_I \in 2^{\mathcal{F}}$ is the initial state. $g \subseteq \mathcal{F}$ is the goal description.

A task network $tn = (T, \prec, \alpha)$ is refined into another one $tn' = (T', \prec', \alpha')$ by a method $m = (c, tn_m)$, written $tn \rightarrow_m tn'$, if there exists a task identifier $t \in T$ with $\alpha(t) = c$, and there exists a task network $tn'_m = (T_m, \prec_m, \alpha_m)$ with $tn'_m \cong tn_m$ such that

- $T' = (T \backslash \{t\}) \cup T_m$,
- $\prec' = (\prec \cup \prec_m \cup \prec_X) \cap (T' \times T')$ in which $\prec_X = \{(t_1, t_2) \,|\, (t_1, t) \in \prec, t_2 \in T_m\} \cup \{(t_2, t_1) \,|\, (t, t_1) \in \prec, t_2 \in T_m\}$, and
- $\alpha' = (\alpha \backslash \{(t, c)\}) \cup \alpha_m$.

A task network tn is refined into another one tn' by a sequence of methods $\overline{m} = \langle m_1 \cdots m_n \rangle$ ($n \in \mathbb{N}_0$), written $tn \rightarrow^*_{\overline{m}} tn'$ if there exists a sequence of task networks $\langle tn_0 \cdots tn_n \rangle$ such that $tn_0 = tn$, $tn_n = tn'$, and for each $1 \leq i \leq n$, $tn_{i-1} \rightarrow_{m_i} tn_i$.

Given an $\mathcal{HTN}$ planning problem $\mathcal{P} = (\mathcal{D}, c_I, s_I, g)$ with $\mathcal{D} = (\mathcal{F}, \mathcal{A}, \mathcal{C}, \mathcal{M}, \delta)$, a task network $tn = (T, \prec, \alpha)$ is a solution to $\mathcal{P}$ if for all $t \in T$, $\alpha(t) \in \mathcal{A}$, tn can be refined from c_I (by some method sequence), and tn possesses a linearisation $\overline{tn}$ such that $s_I \rightarrow^*_{\overline{a}} s$ where $\overline{a} = \alpha(\overline{tn})$ for some $s \in 2^{\mathcal{F}}$ with $g \subseteq s$.

A variant of the $\mathcal{HTN}$ formalism which is widely used is called the Task Insertions Hierarchical Task Network formalism, denoted as $\mathcal{TIHTN}$ (Geier and Bercher 2011; Alford, Bercher, and Aha 2015). The only difference between the $\mathcal{TIHTN}$ formalism and the $\mathcal{HTN}$ formalism is that a solution task network tn to a $\mathcal{TIHTN}$ planning problem $\mathcal{P}$ does not have to be a refinement of the initial task c_I of $\mathcal{P}$, instead tn should be obtained from another task network tn', which is decomposed from c_I, by inserting arbitrary actions and ordering constraints to tn'.

To let the introduced hierarchical planning formalisms be able to express TEGs, we imitate how LTL and finite-LTL are incorporated into non-hierarchical planning formalisms. We use $(\mathcal{TI})\mathcal{HTN}$-$\mathcal{L}$, and $(\mathcal{TI})\mathcal{HTN}$-$\mathcal{FL}$ to refer to the formalisms in which the goal of a planning problem is described by an LTL and f-LTL formula respectively.

Formal Languages of Planning Formalisms

Having introduced all planning formalisms needed, we now define the formal languages of those formalisms. We follow the definition by Höller et al. (2014, 2016) by which the formal language of a planning problem in a non-hierarchical planning formalism is the set of all solutions to the problem.

Definition 6. The formal language $\mathcal{L}(\mathcal{P})$ of a planning problem $\mathcal{P}$ in a non-hierarchical planning formalism, i.e., $\mathcal{STRIPS}$, $\mathcal{STRIPS}$-$\mathcal{L}$, or $\mathcal{STRIPS}$-$\mathcal{FL}$, is the set of all solutions to $\mathcal{P}$, i.e., $\mathcal{L}(\mathcal{P}) = \{\omega \mid \omega \text{ is a solution to } \mathcal{P}\}$.

For hierarchical planning formalisms, Höller et al. (2014, 2016) defined that the language of a planning problem $\mathcal{P}$ in a hierarchical planning formalism is the set of all linearizations of all solutions to $\mathcal{P}$.

Definition 7. The formal language $\mathcal{L}(\mathcal{P})$ of a planning problem $\mathcal{P}$ in a hierarchical planning formalism, i.e., $(\mathcal{TI})\mathcal{HTN}$, $(\mathcal{TI})\mathcal{HTN}$-$\mathcal{L}$, or $(\mathcal{TI})\mathcal{HTN}$-$\mathcal{FL}$, is defined as follows: $\mathcal{L}(\mathcal{P}) = \{\omega \mid \omega = \alpha(\overline{tn}), \overline{tn} \in L(tn), tn \in \mathcal{S}(\mathcal{P})\}$, where $\mathcal{S}(\mathcal{P})$ is the set of all solution task networks to $\mathcal{P}$, and $L(tn)$ is the set of all linearizations of tn that is executable in the initial state.

The languages of a planning formalism is the set of all languages that can be described by planning problems in that formalism. Concretely, let X be one of $\mathcal{STRIPS}$, $\mathcal{STRIPS}$-$\mathcal{L}$, $\mathcal{STRIPS}$-$\mathcal{FL}$, $(\mathcal{TI})\mathcal{HTN}$, $(\mathcal{TI})\mathcal{HTN}$-$\mathcal{L}$, and $(\mathcal{TI})\mathcal{HTN}$-$\mathcal{FL}$, and $\mathcal{P}(X)$ be the set of all planning problems in the respective formalism, the languages $\mathcal{L}_X$ of the formalism X is defined as follows:

$$\mathcal{L}_X = \{\mathcal{L}(\mathcal{P}) \mid \mathcal{P} \in \mathcal{P}(X)\}$$

Note that the languages of any introduced planning formalism are of words of finite length, including those formalisms equipped with LTL formulae as TEGs. There, although we artificially extend a state sequence to infinite, the action sequence resulting in this state sequence remains finite.

We use the notation ω instead of $\overline{a}$ in the above definition to refer to an action sequence that is a solution to a planning problem $\mathcal{P}$ in order to emphasize that ω is now referred to a *word* in the language of $\mathcal{P}$ (ω is a standard notation used to denote a word in formal languages, see e.g., the work by Hopcroft, Motwani, and Ullman (2007)). Given a planning problem $\mathcal{P}$ in any introduced (hierarchical or non-hierarchical) planning formalism whose action set is $\mathcal{A}$, the presence of the function δ allows us to directly compare $\mathcal{L}(\mathcal{P})$ with some formal language defined over the alphabet Σ with $\Sigma = \mathcal{A}$. Such a treatment makes the comparison between the languages of a planning formalism and other formal languages more straightforward, and hence we can assess the expressiveness of a planning formalism by finding the position of the languages of this formalism in the Chomsky hierarchy, as proposed by Höller et al. (2014, 2016).

For the purpose of simplicity, we restrict that *all* planning problems in *all* formalisms which occur later in this paper are defined over the same action set $\mathcal{A}$, and all formal languages are defined over the alphabet Σ with $\Sigma = \mathcal{A}$.

Expressiveness of Planning Formalisms

We move on to study the expressiveness of the planning formalisms introduced, starting by non-hierarchical ones.

Expressiveness of Non-hierarchical Formalisms

We first present some existing results for the expressiveness of the $\mathcal{STRIPS}$ formalism. Höller et al. (2016) have shown that the languages of the $\mathcal{STRIPS}$ formalism are a strict subset of regular languages $\mathcal{REG}$. They explored this result by exploiting the proposition which we write down as follows because we will require it later on.

Proposition 1 (Höller et al. (2016, Prop. 1))**.** *Let s and s' be two states and a be an action, if $s \rightarrow_{\overline{a}}^{*} s'$ for $\overline{a} = \langle a\, a \rangle$, then for any $\overline{a}' = \langle a \dots a \rangle$ with $|\overline{a}'| > 0$ where $|\overline{a}'|$ is the length of $\overline{a}'$, $s \rightarrow_{\overline{a}'}^{*} s'$.*

However, Höller et al. (2016) did not give a tight bound for the expressiveness of the $\mathcal{STRIPS}$ formalism. Here, we tighten their result by showing that the languages of the $\mathcal{STRIPS}$ formalism is a strict subset of star-free languages $\mathcal{SF}$. Our justification is based on an important property of star-free languages, that is, every star-free language is *non-counting* and every regular language that is non-counting is also star-free Mateescu and Salomaa (1997).

Definition 8. A language $\mathcal{L}$ over the alphabet Σ is non-counting if there exists an $n_0 \in \mathbb{N}$ such that for every $n \geq n_0$ and for any $u, v, w \in \Sigma^*$, $uv^n w \in \mathcal{L}$ *iff* $uv^{n+1}w \in \mathcal{L}$.

The notation v^n refers to the word consisting of n consecutive v. We show that, by using this property, every language described by a $\mathcal{STRIPS}$ planning problem is star-free.

Lemma 1. *Given a $\mathcal{STRIPS}$ planning problem $\mathcal{P}$, the language described by $\mathcal{P}$ is non-counting.*

Proof. We take $n_0 = 2$ as the threshold. We need to show that for every $n \geq n_0$, $uv^n w$ is a solution to $\mathcal{P}$ *iff* $uv^{n+1}w$ is a solution as well.

($\Longrightarrow$): Assume that, without loss of generality, $v = \langle a_1 \cdots a_i \rangle$ for some $i \geq 1$, v^n is executed in the state s, i.e., $s_I \rightarrow_u^* s$, and $s \rightarrow_v^* s'$. The process $s \rightarrow_v^* s'$ can be captured by the formula $s' = (s \backslash D) \cup A$ with $D \subseteq \bigcup_{j=1}^{i} del(a_j)$ and $A \subseteq \bigcup_{j=1}^{i} add(a_j)$. Observe that $s' = (s' \backslash D) \cup A$, because for every $d \in D$, if $d \notin A$, then d has already been removed from s, and hence $d \notin s'$, otherwise, d will first be removed from s' and be added back latter on by A. It is thus trivial to prove the argument by induction on n.

($\Longleftarrow$): Assume that $s \rightarrow_{v^n}^* s_n$ and that $s \rightarrow_{v^{n+1}}^* s_{n+1}$. By definition, $n + 1 \geq 3$, and thus $s_n = s_{n+1}$ according to our previous argument. It follows immediately that if $uv^{n+1}w$ is a solution to $\mathcal{P}$, then $uv^n w$ is also a solution. □

One can see that the presented lemma generalizes Prop.1 by Höller et al. (2016), and it follows from this lemma that every language described by a $\mathcal{STRIPS}$ planning problem is both regular and non-counting, and it is thus star-free. Further, by Prop. 1 (Höller et al. 2016), we can construct the language $\{\langle aa \rangle\}$, which by definition is star-free and cannot be expressed by any $\mathcal{STRIPS}$ planning problem (Höller et al. 2016, Thm. 2). Hence, the languages of the $\mathcal{STRIPS}$ formalism form a strict subset of star-free languages.

Corollary 1. $\mathcal{L}_{\mathcal{STRIPS}} \subsetneq \mathcal{SF}$.

Now that we can place the languages of the $\mathcal{STRIPS}$ formalism to the precise location in the Chomsky hierarchy, i.e., below star-free languages, we then move on to consider the expressive power of the $\mathcal{STRIPS}$-$\mathcal{L}$ formalism. The next theorem shows that the $\mathcal{STRIPS}$-$\mathcal{L}$ formalism is strictly more expressive than the $\mathcal{STRIPS}$ formalism by exploiting the feature that TEGs written as LTL formulae provide the ability of imposing state constraints over state trajectories produced by action sequences.

Theorem 1. $\mathcal{L}_{\mathcal{STRIPS}} \subsetneq \mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{L}}$.

Proof. We first observe that the goal description g in an arbitrary $\mathcal{STRIPS}$ planning problem can be written as a TEG in LTL: $\diamond(\square(\bigwedge_{p \in g} p))$. Therefore, any language that can be described by a $\mathcal{STRIPS}$ planning problem can also be described by a $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem.

To prove the strict inclusion relationship, we consider a $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem $P = (\mathcal{F}, \mathcal{A}, s_I, g)$ where $\mathcal{F} = \{p_a \mid a \in \mathcal{A}\}$, for each $a \in \mathcal{A}$, $prec(a) = \emptyset$, $add(a) = \{p_a\}$, and $del(a) = \{p_{a'} \mid a' \neq a, a' \in \mathcal{A}\}$, and $g = \bigcirc(\bigcirc(\bigcirc p_{\hat{a}}))$ for some $\hat{a} \in \mathcal{A}$. This planning problem describes the language $\Sigma^2 \hat{a} \Sigma^*$ which we denote as $\mathcal{L}$. We prove by contradiction that there exist no $\mathcal{STRIPS}$ planning problems which can express $\mathcal{L}$. Suppose that there exists a $\mathcal{STRIPS}$ planning problem which can express this language. Thus, the action sequence $\langle b\, b\, \hat{a}\, a_{i_1} \cdots a_{i_k} \rangle$ with $b \in \mathcal{A}$ and $a_{i_j} \in \mathcal{A}$ for each $1 \leq j \leq k$ must be a solution to this planning problem. However, by Prop. 1 (Höller et al. 2016), the action sequence $\langle b\, b \cdots b\, \hat{a}\, a_{i_1} \cdots a_{i_k} \rangle$ should then be a solution as well, which results in a contradiction. Therefore, there exists at least one $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem which expresses a language that cannot be expressed by any $\mathcal{STRIPS}$ planning problem. □

We move on to consider the relative position of the languages of the $\mathcal{STRIPS}$-$\mathcal{L}$ formalism in the Chomsky hierarchy with respect to star-free languages. We will show that the $\mathcal{STRIPS}$-$\mathcal{L}$ formalism is strictly less expressive than star-free languages, i.e., $\mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{L}} \subsetneq \mathcal{SF}$. For this, we first prove that every language described by a $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem is star-free.

Given an arbitrary $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem $\mathcal{P} = (\mathcal{F}, \mathcal{A}, \delta, s_I, g)$, we construct $\mathcal{P}' = (\mathcal{F}', \mathcal{A}, \delta', s_I, g)$ such that $\mathcal{F}' = \mathcal{F} \cup P_A$ with $P_A = \{p_a \mid a \in \mathcal{A}\}$ and $\mathcal{F} \cap P_A = \emptyset$, and for each $a \in \mathcal{A}$ with $\delta(a) = (prec(a), add(a), del(a))$, $\delta'(a) = (prec(a), add(a) \cup \{p_a\}, del(a) \cup (P_A \backslash \{p_a\})$. By construction, $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}')$. It is thus sufficient to prove that $\mathcal{L}(\mathcal{P}')$ is star-free.

To this end, we consider a language $\mathcal{L}_s(\mathcal{P}')$ defined over the alphabet $\Sigma_S \subseteq 2^{\mathcal{F}'}$ containing all states $s \in 2^{\mathcal{F}'}$ such that $|s \cap P_A| = 1$. Note that the constructed alphabet Σ_S is a set of *states*, which is a wildly used construction when considering the language of an LTL formula, e.g., as seen in the works by Baier and Katoen (2008) and De Giacomo and Vardi (2013). The language $\mathcal{L}_s(\mathcal{P}')$ is defined as follows.

Definition 9. $\mathcal{L}_s(\mathcal{P}')$ is the set of all words $\omega = \langle s_1 \cdots s_n \rangle$, $n \in \mathbb{N}$, such that ω is a state sequence in $\mathcal{P}'$ which is the consequence of applying an action sequence in s_I and its infinite extension $\omega' = \langle s_1 \cdots s_n s_n \cdots \rangle$ satisfies g, i.e., $\omega' \vDash g$.

We first show if $\mathcal{L}_s(\mathcal{P}')$ is star-free, then so is $\mathcal{L}(\mathcal{P}')$.

Lemma 2. *If $\mathcal{L}_s(\mathcal{P}')$ is star-free, then $\mathcal{L}(\mathcal{P}')$ is star-free.*

Proof. Note that by taking $n \in \mathbb{N}$, $\mathcal{L}_s(\mathcal{P}')$ excludes the word $\omega = \langle s_I \rangle$ from which we observe that there exists a bijective mapping between $\mathcal{L}_s(\mathcal{P}')$ and $\mathcal{L}(\mathcal{P}')\backslash\{\varepsilon\}$ where ε refers to the empty word (i.e., the empty action sequence). We use $\mathcal{L}^*(\mathcal{P}')$ to refer to $\mathcal{L}(\mathcal{P}')\backslash\{\varepsilon\}$. We can further identify the homomorphism h from this bijective mapping where for each $s \in \Sigma_S$, $h(s) = a$ for an action $a \in \mathcal{A}$ if $p_a \in s$, and $h(s_I) = \varepsilon$. More concretely, $\mathcal{L}^*(\mathcal{P}')$ can be represented as $\{h(\omega) \mid \omega \in \mathcal{L}\}$ where for each word $\omega = \langle s_1 \cdots s_n \rangle$ ($n \in \mathbb{N}$) in $\mathcal{L}$, $h(\omega)$ is $h(\langle s_1 \cdots s_n \rangle) = \langle h(s_1) \cdots h(s_n) \rangle$.

By our assumption, $\mathcal{L}_s(\mathcal{P}')$ is star-free and henceforth regular. $\mathcal{L}^*(\mathcal{P}')$ is thus also regular because regular languages are closed under homomorphisms (see the work by Hopcroft, Motwani, and Ullman (2007) for more details).

Moreover, $\mathcal{L}^*(\mathcal{P}')$ is non-counting provided that $\mathcal{L}_s(\mathcal{P}')$ is. For this, we take $n_0^* = \max\{2, n_0+1\}$ where n_0 satisfies that for each $n \geq n_0$ and any $\pi_u, \pi_v, \pi_w \in \Sigma_S^*$, $\pi_u \pi_v^n \pi_w$ is a word in $\mathcal{L}_s(\mathcal{P}')$ *iff* $\pi_u \pi_v^{n+1} \pi_w$ is in $\mathcal{L}_s(\mathcal{P}')$. Consider any $u, v, w \in \Sigma^*$ ($\Sigma = \mathcal{A}$) such that $uv^n w \in \mathcal{L}^*(\mathcal{P}')$ for some $n \geq n_0^*$. That is, $uv^n w$ is an action sequence that is a solution to $\mathcal{P}'$. We assume that $uv^n w$ results in the state sequence π. We can divide π into three subsequences, i.e., $\pi = \pi_u \pi_{v^n} \pi_w$. π_u is the state sequence resulting from applying u in s_I. We use $s_u \in 2^{\mathcal{F}'}$ to refer to the last state in π_u, i.e., $s_I \rightarrow_u^* s_u$. π_{v^n} is the consequence of applying v^n in s_u. We use s_v to refer to the last state in π_{v^n}, i.e., $s_u \rightarrow_{v^n}^* s_v$. Similarly, π_w and s_w refer to the state sequence resulting from applying w in s_v and the last state in π_w, respectively. Further, suppose that $s_u \rightarrow_v^* s$ for some $s \in 2^{\mathcal{F}'}$. Since $n \geq 2$, we know that $s_u \rightarrow_{v^k}^* s$ for each $k \geq 1$ by consulting the argument proved in Lem. 1. Thus, $s = s_v$. We use π_v to denote the state sequence that is the consequence of applying v in the state s_u. Further, we also have $s_v \rightarrow_v^* s_v$, because $s_u \rightarrow_{v^2}^* s_v$ and $s_u \rightarrow_v^* s_v$ hold simultaneously. Thus, applying v^k with $k \geq 1$ in s_v results in the state sequence γ_v^k where γ_v is the state sequence that is the consequence of applying v in s_v. It follows that π_{v^n} can be written as $\pi_{v^n} = \pi_v \gamma_v^{n-1}$. Consequently, $\pi = \pi_u \pi_v \gamma_v^{n-1} \pi_w$. Since $\pi \in \mathcal{L}_s(\mathcal{P}')$ and $n - 1 \geq n_0$, we have $\beta = \pi_u \pi_v \gamma_v^n \pi_w \in \mathcal{L}_s(\mathcal{P}')$ as well by the non-counting property of $\mathcal{L}_s(\mathcal{P}')$. Applying the homomorphism h on β results in $h(\beta) = uv^{n+1}w$. Hence, $uv^{n+1}w \in \mathcal{L}^*(\mathcal{P}')$.

On the other hand, the fact that $uv^n w$ is in $\mathcal{L}(\mathcal{P}')$ given $uv^{n+1}w \in \mathcal{L}(\mathcal{P}')$ for any $n \geq n_0^*$ can be proved in the same way. Thus, $\mathcal{L}^*(\mathcal{P}')$ is both regular and non-counting, and hence it is star-free if $\mathcal{L}_s(\mathcal{P}')$ is star-free. As a consequence, $\mathcal{L}(\mathcal{P}') = \mathcal{L}^*(\mathcal{P}') \cup \{\varepsilon\}$ is thus also a star-free language if $\mathcal{L}_s(\mathcal{P}')$ is star-free, because the union of a star-free language and the language $\{\varepsilon\}$ is star-free by definition. □

To prove that $\mathcal{L}_s(\mathcal{P}')$ is star-free, we observe that $\mathcal{L}_s(\mathcal{P}')$ can be seen as the intersection of two languages $\mathcal{L}_s^1(\mathcal{P}')$ and $\mathcal{L}_s^2(\mathcal{P}')$ in which $\mathcal{L}_s^1(\mathcal{P}')$ is the set of all words (i.e., state sequences) that are consequences of applying action sequences in s_I in $\mathcal{P}'$ and $\mathcal{L}_s^2(\mathcal{P}')$ is the set of all words satisfying g by extending to infinity, i.e.,

$$\mathcal{L}_s^2(\mathcal{P}') = \{\langle s_1 \cdots s_n \rangle \mid n \in \mathbb{N}, \langle s_1 \cdots s_n s_n \cdots \rangle \vDash g\}$$

We will show that both $\mathcal{L}_s^1(\mathcal{P}')$ and $\mathcal{L}_s^2(\mathcal{P}')$ are star-free.

Lemma 3. *$\mathcal{L}_s^1(\mathcal{P}')$ is star-free.*

Proof. The star-freeness of $\mathcal{L}_s^1(\mathcal{P}')$ can be recognized by noticing that $\mathcal{L}_s^2(\mathcal{P}')$ is an inverse homomorphism h^{-1} of the language of the $\mathcal{STRIPS}$ planning problem $\mathcal{P}^* = (\mathcal{F}', \mathcal{A}, \delta', s_I, \emptyset)$ where h is the homomorphism defined previously. Since $\mathcal{L}(\mathcal{P}^*)$ is star-free and henceforth regular, and regular languages are closed under inverse homomorphism (Hopcroft, Motwani, and Ullman 2007), $\mathcal{L}_s^1(\mathcal{P}')$ is thus regular. To show that $\mathcal{L}_s^1(\mathcal{P}')$ is non-counting, we take $n_0^* = n_0$ where n_0 satisfies that for any $u, v, w \in \Sigma^*$, $uv^n w \in \mathcal{L}(\mathcal{P}^*)$ *iff* $uv^{n+1}w \in \mathcal{L}(\mathcal{P}^*)$ for each $n \geq n_0$. Consider any $\pi_u, \pi_v, \pi_w \in \Sigma_P^*$ such that $\pi = \pi_u \pi_v^n \pi_w$ is a word in the language $\mathcal{L}_s^1(\mathcal{P}')$ for some $n \geq n_0^*$, the inverse homomorphism implies that $\omega = h(\pi_u)h^n(\pi_v)h(\pi_w)$ is a word in $\mathcal{L}_s^1(\mathcal{P}')$. Let $h(\pi_u) = u$, $h(\pi_v) = v$, and $h_{\pi_w} = w$, the non-counting property of $\mathcal{L}(\mathcal{P}^*)$ ensures that $uv^{n+1}w \in \mathcal{L}(\mathcal{P}^*)$. Since the consequence of applying $uv^{n+1}w$ in s_I is $h(\pi_u)h^{n+1}(\pi_v)h(\pi_w)$, we have $\pi_u \pi_v^n \pi_w \in \mathcal{L}_s^1(\mathcal{P}')$. The converse of the argument can be proved similarly. □

Next we prove that $\mathcal{L}_s^2(\mathcal{P}')$ is star-free. For this, we show that $\mathcal{L}_s^2(\mathcal{P}')$ can be described by a First Order Logic (FOL) formula as any language that can be described by a FOL formula is star-free (Thomas 1997; Diekert and Gastin 2008).

To this end, we first introduce the concept of *word models* for FOL adapted from the one by Thomas (1997) and by De Giacomo and Vardi (2013). Given a set of propositions P and an alphabet $\Sigma_P \subseteq 2^P$, a word $\omega = \langle a_1 \cdots a_n \rangle$ over Σ_P can be described as a *structure* $(\mathcal{W}, R)$ where $\mathcal{W} = \{i \mid 1 \leq i \leq |\omega|\}$ is the domain of the structure, and $\mathcal{R} = \{S, <\} \cup \{Q_p \mid p \in P\}$ is the set of relations. For any $x, y \in \mathcal{W}$, $(x, y) \in S$ if $y = x+1$. $<$ is the natural order relation defined over natural numbers. Q_p for some $p \in P$ is a unary relation such that for any $x \in \mathcal{W}$, $x \in Q_p$ if $p \in a_x$. The language of a FOL formula ϑ, $\mathcal{L}(\vartheta)$, is the set of all structures (i.e., words) satisfying ϑ.

Lemma 4. *$\mathcal{L}_s^2(\mathcal{P}')$ is star-free.*

Proof. We first give the construction $\mathcal{T}$ adapted from the ones by Diekert and Gastin (2008) and De Giacomo and Vardi (2013), which, by given a set of propositions P, transforms an LTL formula φ into a FOL formula $\mathcal{T}(\varphi)(x)$ that contains a free variable x and satisfies that for each sequence $\pi = \langle s_1 \cdots s_n \rangle$ and its infinite extension $\pi' = \langle s_1 \cdots s_n s_n \cdots \rangle$ ($n \in \mathbb{N}$ and $s_i \in 2^P$ for each $1 \leq i \leq n$), $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i]$ *iff* $\pi_i' \vDash \varphi$ for each i with $1 \leq i \leq n$. $\mathcal{T}$ can be constructed inductively as follows.

- If $\varphi = \top$, $\mathcal{T}(\varphi)(x) = \top$.
- If $\varphi = p$ for some $p \in P$, $\mathcal{T}(\varphi)(x) = Q_p(x)$.
- If $\varphi = \varphi_1 \wedge \varphi_2$, $\mathcal{T}(\varphi)(x) = \mathcal{T}(\varphi_1)(x) \wedge \mathcal{T}(\varphi_2)(x)$.
- If $\varphi = \neg\varphi_1$, $\mathcal{T}(\varphi)(x) = \neg\mathcal{T}(\varphi_1)(x)$.

- If $\varphi = \bigcirc\varphi_1$, $\mathcal{T}(\varphi)(x) = \Big(\exists y\big(S(x,y) \wedge \mathcal{T}(\varphi_1)(y)\big)\Big) \vee \Big(\big(\nexists y(S(x,y))\big) \wedge \mathcal{T}(\varphi_1)(x)\Big)$.
- Lastly, for the case $\varphi = \varphi_1 \,\mathtt{U}\, \varphi_2$, $\mathcal{T}(\varphi)(x) = \Big(\exists y\big((x \leq y) \wedge \mathcal{T}(\varphi_2)(y)\big)\Big) \wedge \Big(\forall z\big((x \leq z < y) \rightarrow \mathcal{T}(\varphi_1)(z)\big)\Big)$.

The construction presented differs from the ones by Diekert and Gastin (2008) and De Giacomo and Vardi (2013) only in the case where $\varphi = \bigcirc\varphi_1$. Intuitively, the construction for this case means that if the position $y = x + 1$ exists in π, i.e., x is not the end of a sequence π, then $\pi_y \vDash \mathcal{T}(\varphi_1)(y)$, otherwise, $\pi_x \vDash \mathcal{T}(\varphi_1)(x)$. We can show that, given any $\pi = \langle s_1 \cdots s_n \rangle$, $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i] \iff \pi'_i \vDash \varphi$ holds for each $1 \leq i \leq n$ by induction on $\mathcal{T}$.

We now prove that given an LTL formula φ and a state sequence $\pi = \langle s_1 \cdots s_n \rangle$, $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i] \iff \pi'_i \vDash \varphi$ holds for each $1 \leq i \leq n$, where $\pi'_i = \langle s_1 \cdots s_n s_n \cdots \rangle$.

We prove this by induction on the construction of $\mathcal{T}$.

- $\varphi = \top$: The property holds naturally.
- $\varphi = p$: If $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i]$, then $p \in s_i$. Thus, $\pi'_i \vDash \varphi$. Conversely, if $\pi'_i \vDash \varphi$, $p \in s_i$ holds as well. Consequently, $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i]$.
- $\varphi = \varphi_1 \wedge \varphi_2$: Suppose $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i]$. We have $\pi_i \vDash \mathcal{T}(\varphi_1)[x \mapsto i]$ and $\pi_i \vDash \mathcal{T}(\varphi_2)[x \mapsto i]$. By the induction hypothesis, $\pi'_i \vDash \varphi_1$ and $\pi'_i \vDash \varphi_2$. Hence, $\pi'_i \vDash \varphi$. Conversely, if $\pi'_i \vDash \varphi_1$ and $\pi'_i \vDash \varphi_2$, we have $\pi_i \vDash \mathcal{T}(\varphi_1)[x \mapsto i]$ and $\pi_i \vDash \mathcal{T}(\varphi_2)[x \mapsto i]$ by the induction hypothesis. Thus, $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i]$.
- $\varphi = \neg\varphi_1$: Suppose $\pi_i \vDash \neg\mathcal{T}(\varphi_1)[x \mapsto i]$. It follows that $\pi_i \nvDash \mathcal{T}(\varphi_1)[x \mapsto i]$. Thus, by the induction hypothesis, $\pi'_i \nvDash \varphi_1$, which is equivalent to $\pi'_i \vDash \varphi$. Conversely, if $\pi'_i \nvDash \varphi_1$, we also have $\pi_i \nvDash \mathcal{T}(\varphi_1)[x \mapsto i]$ by the induction hypothesis. Thus, $\pi_i \vDash \neg\mathcal{T}(\varphi_1)[x \mapsto i]$.
- $\varphi = \bigcirc\varphi_1$: For any $1 \leq i < n$, we observe that $\pi_i \vDash T(\varphi)[x \mapsto i]$ *iff* $\pi_{i+1} \vDash T(\varphi_1)[y \mapsto i+1]$. Since $i+1 \leq n$, we have $\pi_{i+1} \vDash \mathcal{T}(\varphi_1)[y \mapsto i+1]$ *iff* $\pi'_{i+1} \vDash \varphi_1$ by the hypothesis. Thus, $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i] \iff \pi'_i \vDash \varphi$ for any $1 \leq i < n$. Further, when $i = n$, $\pi_i \vDash \mathcal{T}(\varphi_1)[x \mapsto i]$ *iff* $\pi_i \vDash \mathcal{T}(\varphi_1)[x \mapsto i]$, and $\pi'_i \vDash \varphi$ *iff* $\pi'_{i+1} \vDash \varphi_1$ by the semantics of LTL. Since $i = n$ and the state s_n repeats itself indefinitely, we have $\pi'_{i+1} \vDash \varphi_1$ *iff* $\pi'_i \vDash \varphi_1$. Hence, $\pi_i \vDash \mathcal{T}(\varphi_1)[x \mapsto i]$ *iff* $\pi'_i \vDash \varphi_1$ for $i = n$. Thus, $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i]$ *iff* $\pi'_i \vDash \varphi$ for all $1 \leq i \leq n$.
- $\varphi = \varphi_1 \,\mathtt{U}\, \varphi_2$: Suppose $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i]$. There exists a j with $i \leq j \leq n$ such that $\pi_j \vDash \mathcal{T}(\varphi_2)[y \mapsto j]$. By the induction hypothesis, $\pi'_j \vDash \varphi_2$. Further, for each k with $i \leq k < j$, $\pi_k \vDash \mathcal{T}(\varphi_1)[z \mapsto k]$. By the induction hypothesis, we also have $\pi'_k \vDash \varphi_1$. Taken together, $\pi'_i \vDash \varphi$. Conversely, suppose $\pi'_i \vDash \varphi$. There exists a j with $j \geq i$ such that $\pi'_j \vDash \varphi_2$. We only need to consider the case where $j \leq n$ because s_n in π' repeats itself indefinitely, and hence $\pi'_j \vDash \varphi_2$ *iff* $\pi'_n \vDash \varphi_2$ for all $j > n$. By the induction hypothesis, $\pi'_j \vDash \varphi_2$ implies $\pi_j \vDash \mathcal{T}(\varphi_2)[y \mapsto j]$. Further, for each $i \leq k < j$, $\pi'_k \vDash \varphi_1$. It follows that $\pi_k \vDash \mathcal{T}(\varphi_1)[z \mapsto k]$. Thus, $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i]$.

Hence, $\pi_i \vDash \mathcal{T}(\varphi)[x \mapsto i] \iff \pi'_i \vDash \varphi$. We can thus construct the FOL formula $\mathcal{T}(g)[x \mapsto 1]$ describing the language $\mathcal{L}^2_s(\mathcal{P}')$, which implies that $\mathcal{L}^2_s(\mathcal{P}')$ is star-free. □

Therefore, the language $\mathcal{L}_s(\mathcal{P}')$, which is the intersection of $\mathcal{L}^1_s(\mathcal{P}')$ and $\mathcal{L}^2_s(\mathcal{P}')$, is star-free. It follows that $\mathcal{L}(\mathcal{P}')$ and henceforth $\mathcal{L}(\mathcal{P})$ are star-free.

Lastly, we show that the subset relationship is strict. We will prove that there exists no $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem that can express the star-free languages $\{\langle a\, a\rangle\}$.

Lemma 5. $\{\langle a\, a\rangle\} \notin \mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{L}}$.

Proof. The proof is a simple extension of the one given by Höller et al. (2016) for showing that $\{\langle a\, a\rangle\} \notin \mathcal{L}_{\mathcal{STRIPS}}$. They used this to show that $\mathcal{L}_{\mathcal{STRIPS}}$ can't express all regular languages (cf. Cor. 1). $\{\langle a\, a\rangle\}$ however also happens to be star-free, so we can generalize this to STRIPS with LTL.

By Prop. 1 (Höller et al. 2016), for an arbitrary $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem $\mathcal{P}$, the infinite extension of the state sequence obtained by applying $\overline{a} = \langle a \cdots a\rangle$ with $|\overline{a}| \geq 2$ in the initial state of $\mathcal{P}$ and of the one obtained by applying $\langle a\, a\rangle$ are identical. Hence, if $\langle a\, a\rangle$ is a solution, then $\overline{a}$ is also a solution, which shows the lemma. □

Taken together, we have the result $\mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{L}} \subsetneq \mathcal{SF}$.

Theorem 2. $\mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{L}} \subsetneq \mathcal{SF}$.

One consequence of artificially extending a state sequence to infinite by repeating its last state is that $\mathcal{STRIPS}$-$\mathcal{L}$ planning problems lack the ability of expressing some star-free languages in which words are of a fixed length, e.g., the language $\{\langle a\, a\rangle\}$ used in the presented proof. It is thus natural to ask whether the $\mathcal{STRIPS}$-$\mathcal{FL}$ formalism in which state sequences remain finite has the same expressive power as star-free languages (De Giacomo and Vardi 2013).

The answer is positive. To get an intuition about this, we construct a planning problem $\mathcal{P} = (\mathcal{F}, \mathcal{A}, \delta, s_I, g)$ that can express the language $\{\langle a\, a\rangle\}$ which cannot be expressed by any $\mathcal{STRIPS}$ and $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem. We let $\mathcal{F} = \{p_{a'} \mid a' \in \mathcal{A}\}$ and $\delta(a) = (\emptyset, \{p_a\}, \emptyset)$ for the action $a \in \mathcal{A}$. For all $a' \in \mathcal{A}$ with $a' \neq a$, we define $\delta(a') = (\{p_{a'}\}, \emptyset, \emptyset)$. Further, we let $g = \bigcirc(p_a \wedge \bigcirc(p_a \wedge \odot))$ and $s_I = \emptyset$. By construction, the sequence $\langle a\, a\rangle$ is the only solution, because the term '$\odot$' in the formula enforces that any state sequence satisfying g must terminate after applying two actions, and because a is the only applicable action. It can be seen from this example that the ability of expressing '*the end of the sequence*' in f-LTL makes a $\mathcal{STRIPS}$-$\mathcal{FL}$ planning problem able to restrict its solutions to a fixed length, which thus gives the planning problems in this formalism the ability to express star-free languages with fixed length. Informally speaking, the $\mathcal{STRIPS}$-$\mathcal{FL}$ formalism is able to *count up to a bound*, whereas it is not possible for the $\mathcal{STRIPS}$ and $\mathcal{STRIPS}$-$\mathcal{L}$ formalism.

We formally prove the argument that the $\mathcal{STRIPS}$-$\mathcal{FL}$ formalism is as expressive as star-free languages in two steps. We first show that every language described by a $\mathcal{STRIPS}$-$\mathcal{FL}$ planning problem is star-free.

Theorem 3. $\mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{FL}} \subseteq \mathcal{SF}$.

Proof. Let $\mathcal{P} = (\mathcal{F}, \mathcal{A}, \delta, s_I, g)$ be a $\mathcal{STRIPS}$-$\mathcal{FL}$ planning problem, De Giacomo and Vardi (2013) have shown

that the language $\mathcal{L}(g)$ of the f-LTL formula g over the alphabet $2^{\mathcal{F}}$ is star-free. Thus, there exists a deterministic finite automaton $\mathcal{D} = (\mathcal{Q}, 2^{\mathcal{F}}, \delta_q, q_0, F)$ capturing $\mathcal{L}(g)$ in which $\mathcal{Q}$ is the set of the states of the DFA, $\delta_q : \mathcal{Q} \times 2^{\mathcal{F}} \mapsto \mathcal{Q}$ is the state transition function, q_0 is the initial state, and $F \subseteq \mathcal{Q}$ is the set of final states (e.g., see the construction by De Giacomo and Favorito (2021)). Additionally, the behaviors of actions in $\mathcal{P}$ can be simulated by a state transition system $\mathcal{T} = (2^{\mathcal{F}}, \mathcal{A}, \delta_a)$ where $\delta_a : 2^{\mathcal{F}} \times \mathcal{A} \mapsto 2^{\mathcal{F}}$ is a function such that for any $s, s' \in 2^{\mathcal{F}}$ and $a \in \mathcal{A}$, $\delta(s, a) = s'$ *iff* $s \rightarrow_a s'$ (De Giacomo and Vardi 1999; Höller et al. 2014).

We first show that there is a DFA $\hat{\mathcal{D}} = (\hat{\mathcal{Q}}, \mathcal{A}, \hat{q}_0, \hat{\delta}_q, \hat{F})$ capturing the language of $\mathcal{P}$ that is a product of $\mathcal{D}$ and $\mathcal{T}$. We construct $\hat{\mathcal{D}}$ as follows. The set of states $\hat{\mathcal{Q}} = \mathcal{Q} \times 2^{\mathcal{F}}$. The initial state $\hat{q}_0 = (q_0, s_I)$. For any $\hat{q}, \hat{q}' \in \hat{\mathcal{Q}}$ with $\hat{q} = (q, s)$ and $\hat{q}' = (q', s')$, and $a \in \mathcal{A}$, $\hat{\delta}_q(\hat{q}, a) = \hat{q}'$ *iff* $\delta_q(q, s) = q'$ and $\delta_a(s, a) = s'$. Lastly, the set of final states $\hat{F} = F \times 2^{\mathcal{F}}$. The function $\hat{\delta}$ guarantees that 1) the state s can trigger the transition from q to q' in $\mathcal{Q}$, and 2) $s \rightarrow_a s'$. Thus, by construction, a word in Σ^*, i.e., an action sequence is accepted by $\hat{\mathcal{D}}$ *iff* the state sequence that is a consequence of applying this action sequence in s_I is accepted by $\mathcal{D}$. This shows that the language of $\mathcal{P}$ is regular.

We prove the star-freeness of $\mathcal{L}(\mathcal{P})$ by showing that $\mathcal{L}(\mathcal{P})$ is non-counting. Since $\mathcal{L}(g)$ is star-free, there exists an $n_0 \in \mathbb{N}$ such that for arbitrary words u, v, and w over the alphabet $2^{\mathcal{F}}$, $uv^n w \in \mathcal{L}(g)$ *iff* $uv^{n+1}w \in \mathcal{L}(g)$ for all $n \geq n_0$. Since applying an action sequence repeatedly in some state results in a repeated state sequence, we can let $n_0^* = \max\{2, n_0\}$ which ensures that for all $n \geq n_0^*$, $xy^n z \in \mathcal{L}(\mathcal{P})$ for any $x, y, z \in \Sigma^*$ *iff* $xy^{n+1}z \in \mathcal{L}(\mathcal{P})$. Any language described by a $\mathcal{STRIPS}$-$\mathcal{FL}$ planning problem is thus star-free. □

Next we prove the converse of this theorem.

Theorem 4. $\mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{FL}} \supseteq \mathcal{SF}$.

Proof. Given any star-free language $\mathcal{L}$ over the alphabet Σ, Wilke (1999) has proven that $\mathcal{L}$ can be described by an f-LTL formula φ defined over the set of propositions $P = \{p_a \mid a \in \Sigma\}$. Concretely, a word $\omega = \langle a_1 \cdots a_n \rangle$ ($n \in \mathbb{N}_0$ and $a_i \in \Sigma$ for each $1 \leq i \leq n$) is in $\mathcal{L}$ *iff* the state sequence $\pi = \langle \{p_{a_1}\} \cdots \{p_{a_n}\} \rangle$ satisfies φ, i.e., $\omega \in \mathcal{L} \iff \pi \vDash \varphi$. We exploit this result to show that for any such f-LTL formula φ together with the star-free language $\mathcal{L}$ defined by it, there exists a $\mathcal{STRIPS}$-$\mathcal{FL}$ planning problem describing the same language. We consider the planning problem $\mathcal{P} = (\mathcal{F}, \mathcal{A}, \delta, \emptyset, g)$ in which $\mathcal{F} = P$, $g = \varphi$, and for any $a \in \mathcal{A}$, $\delta(a) = (\emptyset, \{p_a\}, \{p_{a'} \mid a' \in \mathcal{A}, a' \neq a\})$ ($\mathcal{A} = \Sigma$ by our assumption). By construction, any action sequence $\overline{a} = \langle a_1 \cdots a_n \rangle$ results in the state sequence $\pi = \langle \{p_{a_1}\} \cdots \{p_{a_n}\} \rangle$, and it is a solution *iff* $\pi \vDash g$. We can thus conclude that, by consulting the result by Wilke (1999), a word ω is in $\mathcal{L}$ *iff* ω can be seen as a solution to $\mathcal{P}$. □

Taken together, the languages of the $\mathcal{STRIPS}$-$\mathcal{FL}$ formalism are equivalent to star-free languages.

Corollary 2. $\mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{FL}} = \mathcal{SF}$.

Expressiveness of Hierarchical Formalisms

We move on now to consider the expressiveness of hierarchical planning formalisms defined early. One key observation made by Höller et al. (2014) about the language of a planning problem $\mathcal{P} = (\mathcal{D}, c_I, s_I, g)$ with $\mathcal{D} = (\mathcal{F}, \mathcal{A}, \mathcal{C}, \mathcal{M}, \delta)$ in some hierarchical planning formalism, where g is a goal description, an LTL formula, or an f-LTL formula according to the formalism, is that it can be viewed as the intersection of the language of a hierarchical planning problem $\mathcal{P}_1 = (\mathcal{D}', c_I, s_I, g')$ with $prec(a)$, $add(a)$, and $del(a)$ being empty sets for all $a \in \mathcal{A}$ (i.e., $\mathcal{D}$ and $\mathcal{D}'$ differ solely in the preconditions and effects of actions), and $g' = \emptyset$ or $\top$ with respect to the formalism and a non-hierarchical one $\mathcal{P}_2 = (\mathcal{F}, \mathcal{A}, \delta, s_I, g)$. We use the notations $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$ and $\mathcal{L}_{\text{no-H}}(\mathcal{P})$ to respectively refer to the language of $\mathcal{P}_1$ and of $\mathcal{P}_2$. The following two simple propositions related to this observation will be useful in our discussion which, informally speaking, state that the set of all languages governed by the hierarchical parts of planning problems in any one of the presented hierarchical formalism (with or without task insertions, but not both) is identical to each other.

Proposition 2. *Let $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ be planning problems in the formalisms $\mathcal{HTN}$, $\mathcal{HTN}$-$\mathcal{L}$, and $\mathcal{HTN}$-$\mathcal{FL}$ respectively which have the same method set, then $\mathcal{L}_{\text{no-PE}}(\mathcal{P}_1) = \mathcal{L}_{\text{no-PE}}(\mathcal{P}_2) = \mathcal{L}_{\text{no-PE}}(\mathcal{P}_3)$.*

Proposition 3. *Let $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ respectively be planning problems in the formalisms $\mathcal{TIHTN}$, $\mathcal{TIHTN}$-$\mathcal{L}$, and $\mathcal{TIHTN}$-$\mathcal{FL}$ which have the same method set, then $\mathcal{L}_{\text{no-PE}}(\mathcal{P}_1) = \mathcal{L}_{\text{no-PE}}(\mathcal{P}_2) = \mathcal{L}_{\text{no-PE}}(\mathcal{P}_3)$.*

We start by considering the $\mathcal{TIHTN}$ formalism, whose languages have been shown to be a strict subset of regular languages (Höller et al. 2016, Thm. 2). We give a more precise upper bound here for the expressive power of the $\mathcal{TIHTN}$ formalism and show that the languages of this formalism are a strict subset of star-free languages.

Theorem 5. $\mathcal{L}_{\mathcal{TIHTN}} \subsetneq \mathcal{SF}$.

Proof. Given an arbitrary $\mathcal{TIHTN}$ planning problem $\mathcal{P}$, since $\mathcal{L}(\mathcal{P}) = \mathcal{L}_{\text{no-PE}}(\mathcal{P}) \cap \mathcal{L}_{\text{no-H}}(\mathcal{P})$ and $\mathcal{L}_{\text{no-H}}(\mathcal{P})$ is star-free by Cor.1, it is sufficient to show that $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$ is star-free. Let $\mathcal{P}_1$ be the relaxed version of $\mathcal{P}$ whose language is $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$, the result by (Geier and Bercher 2011) provided the fact that for any solution tn (i.e., a primitive task network) to $\mathcal{P}_1$, there exists another task network tn' into which can be decomposed from the initial task by an acyclic decomposition tree and can be expanded to tn via task insertions. Since there are finite many such acyclic decomposition trees, the number of such tn' is finite. We use the notation $\mathcal{T}_{ac}$ to denote the set of all primitive task networks that are decomposed from the initial task by acyclic decomposition trees. Therefore, the set of the words $\mathcal{W} = \{\omega \mid \omega = \alpha(\overline{tn}), \overline{tn} \in L(tn), tn \in \mathcal{T}_{ac}\}$ is finite where $L(tn)$ is the set of all *executable* linearizations of tn in the initial state of $\mathcal{P}_1$. Thus, $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$ can be constructed as follows:

$$\mathcal{L}_{\text{no-PE}}(\mathcal{P}) = \bigcup_{\omega \in \mathcal{W}} \{\Sigma^* \, a_1 \, \Sigma^* \, a_2 \cdots \Sigma^* \, a_n \, \Sigma^*\}$$

where $\omega = \langle a_1 \cdots a_n \rangle \in \mathcal{W}$ for some $n \in \mathbb{N}$. Thus, the language $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$ is star-free.

Further, the language $\{\langle a\, a \rangle\}$ cannot be expressed by any $\mathcal{TIHTN}$ planning problem, because if the action sequence $\langle a\, a \rangle$ is a solution to a $\mathcal{TIHTN}$ planning problem, then by Prop.1 (Höller et al. 2016), an arbitrary number of the action a can be appended to this sequence, which results in another solution. Therefore, the languages of this formalism are a strict subset of star-free languages. □

One consequence of this result is that the languages of the $\mathcal{TIHTN}$-$\mathcal{L}$ formalism also form a strict subset of star-free languages, because for any $\mathcal{TIHTN}$-$\mathcal{L}$ planning problem $\mathcal{P}$, Prop.3 in conjunction with the argument made in Thm. 5 asserts the star-freeness of $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$, and the star-freeness of $\mathcal{L}_{\text{no-H}}(\mathcal{P})$ follows from Cor. 2. Hence, $\mathcal{L}(\mathcal{P})$ is star-free. Further, we can prove that the language $\{\langle a\, a \rangle\}$ cannot be expressed by any $\mathcal{TIHTN}$-$\mathcal{L}$ planning problem by noting that inserting an arbitrary number of a's to the action sequence $\langle a\, a \rangle$ results in another solution to $\mathcal{P}$.

Corollary 3. $\mathcal{L}_{\mathcal{TIHTN}} \subsetneq \mathcal{L}_{\mathcal{TIHTN}\text{-}\mathcal{L}} \subsetneq \mathcal{SF}$.

The result $\mathcal{L}_{\mathcal{TIHTN}} \subsetneq \mathcal{L}_{\mathcal{TIHTN}\text{-}\mathcal{L}}$ holds because of the fact that the language $\Sigma^2 a \Sigma^*$ can be expressed by some $\mathcal{TIHTN}$-$\mathcal{L}$ planning problem but not by any $\mathcal{TIHTN}$ planning problem. The justification for this is similar to the one made in Thm. 1, that is, if the word $\langle b\, b\, a\, a_{i_1} \cdots a_{i_n} \rangle$ with $a, b, a_{i_k} \in \Sigma$ for each $1 \leq k \leq n$ is in the language of some $\mathcal{TIHTN}$ planning problem, then by Prop. 1 (Höller et al. 2016), the word $\langle b\, b \cdots b\, a a_{i_1} \cdots a_{i_n} \rangle$ with more than two b's before a is also in the language of this planning problem, but it is not in the designated language. Hence, there exists no $\mathcal{TIHTN}$ planning problem which can express the language $\Sigma^2 a \Sigma^*$. Further, observe that for any $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem $\mathcal{P} = (\mathcal{F}, \mathcal{A}, \delta, s_I, g)$, the $\mathcal{TIHTN}$-$\mathcal{L}$ planning problem $\mathcal{P}' = (\mathcal{D}, c_I, s_I, g)$ with $\mathcal{D} = (\mathcal{F}, \mathcal{A}, \mathcal{C}, \{(c_I, (\emptyset, \emptyset, \emptyset))\}, \delta)$ expresses the same language as that of $\mathcal{P}$. Since the language $\Sigma^2 a \Sigma^*$ can be expressed by some $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem, it can thus be expressed by a $\mathcal{TIHTN}$-$\mathcal{L}$ planning problem.

We close the discussion over $\mathcal{TIHTN}$ by considering the expressive power of the $\mathcal{TIHTN}$-$\mathcal{FL}$ formalism.

Theorem 6. $\mathcal{L}_{\mathcal{TIHTN}\text{-}\mathcal{FL}} = \mathcal{SF}$.

Proof. We first show that every star-free language can be expressed by a $\mathcal{TIHTN}$-$\mathcal{FL}$ planning problem. This follows trivially from the fact that for any $\mathcal{STRIPS}$-$\mathcal{FL}$ planning problem $\mathcal{P} = (\mathcal{F}, \mathcal{A}, \delta, s_I, g)$, the language of $\mathcal{P}$ is identical to that of the $\mathcal{TIHTN}$-$\mathcal{FL}$ planning problem $\mathcal{P}' = (\mathcal{D}, c_I, s_I, g)$ with $\mathcal{D} = (\mathcal{F}, \mathcal{A}, \mathcal{C}, \{(c_I, (\emptyset, \emptyset, \emptyset))\}, \delta)$, i.e., $\mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{FL}} \subseteq \mathcal{L}_{\mathcal{TIHTN}\text{-}\mathcal{FL}}$. Since $\mathcal{L}_{\mathcal{STRIPS}\text{-}\mathcal{FL}} = \mathcal{SF}$, we thus have $\mathcal{SF} \subseteq \mathcal{L}_{\mathcal{TIHTN}\text{-}\mathcal{FL}}$. Conversely, for any $\mathcal{TIHTN}$-$\mathcal{FL}$ planning problem $\mathcal{P}$, the language $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$ and $\mathcal{L}_{\text{no-H}}(\mathcal{P})$ are both star-free by Prop.3 and by Cor.2. Hence, their intersection is also star-free. □

Our previous discussion is restricted to the (hierarchical or non-hierarchical) planning formalisms which are strictly less expressive than star-free languages (and henceforth regular languages). Next we turn our attention to the $\mathcal{HTN}$ planning formalism which has more expressive power than regular languages. We start by considering a special class of hierarchical planning problems (i.e., $\mathcal{HTN}$ / $\mathcal{HTN}$-$\mathcal{L}$ / $\mathcal{HTN}$-$\mathcal{FL}$ planning problems) where all task networks in all methods are totally ordered. We use the notations $\mathcal{TOHTN}$, $\mathcal{TOHTN}$-$\mathcal{L}$, and $\mathcal{TOHTN}$-$\mathcal{FL}$ to refer to the class of totally ordered planning problems in the respective formalism. The languages described by $\mathcal{TOHTN}$ have been shown to be equivalent to context-free languages ($\mathcal{CFL}$) by Höller et al. (2014, Thm. 6). The next result shows that the expressive power of $\mathcal{TOHTN}$-$\mathcal{L}$ and of $\mathcal{TOHTN}$-$\mathcal{FL}$ does not increase by incorporating LTL and f-LTL, i.e., they are still equivalent to $\mathcal{CFL}$. We use $\mathcal{L}_{\mathcal{TOHTN}\text{-}\mathcal{L}}$ and $\mathcal{L}_{\mathcal{TOHTN}\text{-}\mathcal{FL}}$ to refer to the languages respectively described by those classes of planning problems.

Theorem 7. $\mathcal{L}_{\mathcal{TOHTN}\text{-}\mathcal{L}} = \mathcal{L}_{\mathcal{TOHTN}\text{-}\mathcal{FL}} = \mathcal{CFL}$.

Proof. We start by considering the class $\mathcal{TOHTN}$-$\mathcal{L}$. For any planning problem $\mathcal{P}$ in this set, the result by Höller et al. (2014, Thm. 6) shows that $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$ is context-free, because by Prop. 2, there must be a planning problem $\mathcal{P}'$ in $\mathcal{TOHTN}$ such that $\mathcal{L}_{\text{no-PE}}(\mathcal{P}) = \mathcal{L}_{\text{no-PE}}(\mathcal{P}')$. Since $\mathcal{L}_{\text{no-H}}(\mathcal{P})$ is star-free and henceforth regular, its intersection with $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$ is thus context-free. Conversely, Höller et al. (2014, Thm. 6) also stated that for any context-free language $\mathcal{L}$ there exists a planning problem $\mathcal{P}$ in $\mathcal{TOHTN}$ such that $\mathcal{L} = \mathcal{L}_{\text{no-PE}}(\mathcal{P})$. It follows that any context-free language can be described by a problem in $\mathcal{TOHTN}$-$\mathcal{L}$. One can easily verify that the same argument holds for the class $\mathcal{TOHTN}$-$\mathcal{FL}$ as well. □

Next we consider $\mathcal{HTN}$, $\mathcal{HTN}$-$\mathcal{L}$, and $\mathcal{HTN}$-$\mathcal{FL}$.

Theorem 8. $\mathcal{L}_{\mathcal{HTN}} \subseteq \mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{L}} \subseteq \mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{FL}}$.

Proof. We start by arguing that $\mathcal{L}_{\mathcal{HTN}} \subseteq \mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{L}}$. Given an arbitrary $\mathcal{HTN}$ planning problem $\mathcal{P}$, by Prop. 2, there must exist an $\mathcal{HTN}$-$\mathcal{L}$ planning problem $\mathcal{P}'$ such that $\mathcal{L}_{\text{no-PE}}(\mathcal{P}) = \mathcal{L}_{\text{no-PE}}(\mathcal{P}')$. Further, we can always construct a $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem such that $\mathcal{L}_{\text{no-H}}(P) = \mathcal{L}_{\text{no-H}}(\mathcal{P}')$, because $\mathcal{L}_{\text{no-H}}(\mathcal{P})$ and $\mathcal{L}_{\text{no-H}}(\mathcal{P}')$ are governed by a $\mathcal{STRIPS}$ planning problem and a $\mathcal{STRIPS}$-$\mathcal{L}$ planning problem respectively, and the languages of the $\mathcal{STRIPS}$ formalism are a strict subset of that of $\mathcal{STRIPS}$-$\mathcal{L}$. Thus, every language described by an $\mathcal{HTN}$ planning problem can also be described by an $\mathcal{HTN}$-$\mathcal{L}$ planning problem.

The argument for $\mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{L}} \subseteq \mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{FL}}$ is similar to the one just provided. One can verify that for any $\mathcal{HTN}$-$\mathcal{L}$ planning problem $\mathcal{P}$, we can construct a $\mathcal{HTN}$-$\mathcal{FL}$ planning problem $\mathcal{P}'$ such that $\mathcal{L}_{\text{no-PE}}(\mathcal{P}) = \mathcal{L}_{\text{no-PE}}(\mathcal{P}')$ and $\mathcal{L}_{\text{no-H}}(\mathcal{P}) = \mathcal{L}_{\text{no-H}}(\mathcal{P}')$. □

Lastly, we place the languages of these three formalisms into the Chomsky hierarchy. We first observe from Thm. 7 that $\mathcal{CFL}$ is clearly a strict subset of $\mathcal{L}_{\mathcal{HTN}}$, of $\mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{L}}$, and of $\mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{FL}}$ respectively, because the languages (i.e., $\mathcal{CFL}$) described by totally ordered planning problems in a hierarchical formalism must be a subset of the languages of this formalism. For the upper bound, we consult the result by Höller et al. (2014, Cor. 2) that for any $\mathcal{HTN}$ planning problem $\mathcal{P}$, $\mathcal{L}_{\text{no-PE}}(\mathcal{P})$ is a context-sensitive language

($\mathcal{CSL}$), and by Prop. 2, the argument holds for the $\mathcal{HTN}$-$\mathcal{L}$ and $\mathcal{HTN}$-$\mathcal{FL}$ formalism as well. Further, for any planning problem $\mathcal{P}$ in a hierarchical formalism, $\mathcal{L}_{\text{no-H}}(\mathcal{P})$ is star-free and henceforth regular, and the intersection of a context-sensitive language and a regular one is still context-sensitive. Therefore, the next result follows immediately.

Corollary 4. $\mathcal{CFL} \subseteq \mathcal{L}_{\mathcal{HTN}} \subseteq \mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{L}} \subseteq \mathcal{L}_{\mathcal{HTN}\text{-}\mathcal{FL}} \subseteq \mathcal{CSL}$.

Conclusion

In this paper, we discussed the expressive power of the $\mathcal{STRIPS}$ formalism and the $\mathcal{HTN}$ formalism in conjunction with LTL as well as its variant finite LTL. For the former one, our results show that when combining the $\mathcal{STRIPS}$ formalism with LTL (i.e., the $\mathcal{STRIPS}$-$\mathcal{L}$ formalism), the set of languages described by planning problems in this formalism is a strict subset of star-free languages, though the languages of LTL are equivalent to star-free ones over words of infinite length. By contrast, the $\mathcal{STRIPS}$-$\mathcal{FL}$ formalism, which fuses finite LTL with the $\mathcal{STRIPS}$ formalism, turns out to be as expressive as star-free languages. For the latter one, we have shown that all variants, i.e., the $\mathcal{HTN}$, $\mathcal{HTN}$-$\mathcal{L}$, and $\mathcal{HTN}$-$\mathcal{FL}$ formalism, are more expressive than context-free languages but less expressive than context-sensitive languages when task insertions are not allowed, otherwise, the expressiveness of these formalisms with task insertions is less or equivalent to star-free languages.

References

Alford, R.; Bercher, P.; and Aha, D. W. 2015. Tight Bounds for HTN Planning with Task Insertion. In *IJCAI 2015*, 1502–1508. AAAI.

Bacchus, F.; and Kabanza, F. 1996. Planning for Temporally Extended Goals. In *AAAI 1996*, 1215–1222. AAAI.

Baier, C.; and Katoen, J. 2008. *Principles of Model Checking*. MIT.

Baier, J. A.; and McIlraith, S. A. 2006a. Planning with First-Order Temporally Extended Goals using Heuristic Search. In *AAAI 2006*, 788–795. AAAI.

Baier, J. A.; and McIlraith, S. A. 2006b. Planning with Temporally Extended Goals Using Heuristic Search. In *ICAPS 2006*, 342–345. AAAI.

Bercher, P.; Alford, R.; and Höller, D. 2019. A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations. In *IJCAI 2019*, 6267–6275. IJCAI.

Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2006. Planning with Qualitative Temporal Preferences. In *KR 2006*, 134–144. AAAI.

Camacho, A.; Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2019. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *IJCAI 2019*, 6065–6073. IJCAI.

Chomsky, N. 1956. Three Models for the Description of Language. *IEEE Transactions on Information Theory*, 2(3): 113–124.

De Giacomo, G.; and Favorito, M. 2021. Compositional Approach to Translate LTLf/LDLf into Deterministic Finite Automata. In *ICAPS 2021*, 122–130. AAAI.

De Giacomo, G.; and Vardi, M. Y. 1999. Automata-Theoretic Approach to Planning for Temporally Extended Goals. In *ECP 1999*, volume 1809, 226–238. Springer.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI 2013*, 854–860. AAAI.

Diekert, V.; and Gastin, P. 2008. First-order definable languages. In *Logic and Automata: History and Perspectives*, volume 2, 261–306. Amsterdam University Press.

Edelkamp, S. 2006. On the Compilation of Plan Constraints and Preferences. In *ICAPS 2006*, 374–377. AAAI.

Erol, K.; Hendler, J.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1): 69–93.

Fikes, R.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3/4): 189–208.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Geier, T.; and Bercher, P. 2011. On the Decidability of HTN Planning with Task Insertion. In *IJCAI 2011*, 1955–1961. AAAI.

Ghallab, M.; Howe, A. E.; Knoblock, C. A.; McDermott, D.; Ram, A.; Veloso, M. M.; Weld, D. S.; and Wilkins, D. E. 1998. PDDL–the planning domain definition language. *Technical Report CVC TR-98-003/DCS TR-1165*.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language Classification of Hierarchical Planning Problems. In *ECAI 2014*, 447–452. IOS.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2016. Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages. In *ICAPS 2016*, 158–165. AAAI.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2007. *Introduction to automata theory, languages, and computation*. Addison-Wesley.

Huth, M.; and Ryan, M. 2000. *Logic in computer science : modelling and reasoning about systems*. Cambridge University Press.

Mateescu, A.; and Salomaa, A. 1997. *Formal Languages: an Introduction and a Synopsis*, 1–39. Springer.

Pinchinat, S.; Rubin, S.; and Schwarzentruber, F. 2022. Formula Synthesis in Propositional Dynamic Logic with Shuffle. In *AAAI 2022*. AAAI.

Pnueli, A. 1977. The Temporal Logic of Programs. In *SFCS 1977*, 46–57. IEEE Computer Society.

Thomas, W. 1997. *Languages, Automata, and Logic*, 389–455. Springer.

Wilke, T. 1999. Classifying Discrete Temporal Properties. In *STACS 1999*, volume 1563, 32–46. Springer.