Temporalising Separation Logic for Planning with Search Control Knowledge*

Xu Lu, Cong Tian† and Zhenhua Duan†
ICTT and ISN Laboratory, Xidian University
Xi'an, 710071, P.R. China
xulu@stu.xidian.edu.cn, {ctian,zhhduan}@mail.xidian.edu.cn

Abstract

Temporal logics are widely adopted in Artificial Intelligence (AI) planning for specifying Search Control Knowledge (SCK). However, traditional temporal logics are limited in expressive power since they are unable to express spatial constraints which are as important as temporal ones in many planning domains. To this end, we propose a two-dimensional (spatial and temporal) logic namely PPTL^{SL} by temporalising separation logic with Propositional Projection Temporal Logic (PPTL). The new logic is well-suited for specifying SCK containing both spatial and temporal constraints which are useful in AI planning. We show that PPTL^{SL} is decidable and present a decision procedure. With this basis, a planner namely *S-TSolver* for computing plans based on the spatio-temporal SCK expressed in PPTL^{SL} formulas is developed. Evaluation on some selected benchmark domains shows the effectiveness of *S-TSolver*.

1 Introduction

Artificial Intelligence (AI) planning [Ghallab *et al.*, 2004] has been studied for decades and become even more attractive in recent years. However, existing planners still suffer from a difficult problem: explosion of the search space. One of the most powerful approaches to cope with it is to provide planners with additional Search Control Knowledge (SCK) in specific domains such that the planner can get rid of irrelevant parts of the search space. Techniques for applying SCK in planning have received much attention. It has been proved that SCK is helpful in improving efficiency of the domain-independent planners [Huang *et al.*, 1999]. Even with the basic search strategy, e.g. depth-first search, and proper SCK, planners are able to achieve surprising effectiveness in a range of benchmark problems.

Several planners such as TLPlan [Bacchus and Kabanza, 2000], TALplanner [Kvarnström and Magnusson, 2003], and SHOP2 [Nau *et al.*, 2003], have successfully exploited SCK to guide planning. Concretely, the forward-chaining planner, TLPlan, provides a first-order Linear Temporal Logic (LTL) with bounded quantification based platform for reasoning about SCK. With this approach, we can specify, for instance, what can or cannot happen at the next time step by using the "next" operator in LTL. TALplanner, another forward-chaining planner, supports a unified planning framework where SCK and other information relevant to planning domain and problem instance are all represented as logic formulas in a member of Temporal Action Logics (TAL) family. Compared with TLPlan, TALplanner uses the same planning paradigm, but it is capable of producing parallel plans. In SHOP2, SCK is expressed with Hierarchical Task Network (HTN) [Georgievski and Aiello, 2015] to guide the search. A transition-based SCK inspired by finite state automata is introduced in [Chrpa and Barták, 2016] where knowledge about dependencies between planning operators is represented with transitions of an automaton so as to restrict the number of operator instances during the planning process. Additionally, several research groups investigate advanced mechanisms on the topic of learning SCK in a given domain [Fern *et al.*, 2004; Yoon *et al.*, 2008].

SCK based planning approaches are proved to be effective in practice. Our observation is that despite temporal constraints in SCK, consideration of spatial constraints will be of equal importance. In many circumstances, just temporal SCK is not enough to guarantee a planner to perform effectively as expected. However, temporal logics employed in the existing logic-based planning frameworks are unable to describe spatial structures at each time step. Thus, this paper is motivated to propose a two-dimensional (spatial and temporal) logic namely PPTL^{SL} for describing spatio-temporal SCK. To do so, we temporalise separation logic [Reynolds, 2002] with PPTL (Propositional Projection Temporal Logic) [Duan, 1996] to express both spatial and temporal dimensions of SCK. PPTL is a powerful linear time based temporal logic which facilitates to describe full regular language that goes beyond both LTL and CTL [Tian and Duan, 2008]. Separation logic is a spatial logic for reasoning about dynamic allocated data structures, e.g. linked lists, trees and DAGs (Directed Acyclic Graph). Within PPTL^{SL}, a decidable fragment of separation logic is used to describe the spatial structure at a point of time and temporal operators of PPTL are adopted to capture the evolution of spatial structures over time.

*This research is supported by the NSFC Grant Nos. 61420106004 and 91418201.

†Corresponding authors. All authors are joint first authors.

The main contributions of this paper are twofold: on one hand, we propose a two-dimensional logic system for both spatial and temporal constraints in planning; on the other hand, we prove the decidability of the proposed logic and implement a planner running on benchmarks to demonstrate its effectiveness. With the best of our knowledge, our work is the first use of separation logic in planning.

Related work: Spatio-temporal logic MTLA presented in [Merz *et al.*, 2003] is specialized for mobile systems. It augments Temporal Logic of Actions with spatial modalities to express the topology of configurations in tree structures. But description of other non-tree structures cannot be supported. SpaTel is a spatio-temporal logic [Haghighi *et al.*, 2015] which can describe a property such as "power consumption in city A and city B never exceed 150 MW for longer than 20 minutes". The spatial dimension here is represented by static individual locations instead of spatial structures. The approach in [Belouaer and Maris, 2010] defines SMT encoding rules for spatio-temporal planning by allowing synchronization of non-instantaneous actions in space. In this approach, the temporal and spatial dimensions are independent. The Spatio-Temporal Planning System (STPS) proposed in [Boukharrou *et al.*, 2015] provides a formal description of possible plan actions of an ambient intelligent agent including timing constraints, action duration and spatial requirements of an agent plan. However, only limited spatial knowledge such as "an agent is at some location" can be described in STPS. Compared with the existing spatio-temporal logics, our logic reflects the evolution of a variety of spatial structures especially spatial non-interference features in AI planning. For example, the property "two spatial structures (e.g. linked lists) are always disjoint" can be expressed in PPTL$^{\mathrm{SL}}$, but cannot be expressed by other relevant logics.

2 A Motivating Example

CityCar domain encoded in PDDL (Planning Domain Definition Language) standard [Fox and Long, 2003] is a newly introduced benchmark in International Planning Competition (IPC) 2014 [Vallati *et al.*, 2015]. It aims to simulate the impact of road building/demolition in traffic networks. A city is represented as a graph, in which each node is a junction and edges are roads. Some cars start from different positions (identified as garages) and have to reach their final destinations as soon as possible. There are a finite number of one-way roads available, which can be built for connecting two neighbor junctions and allowing a car to move between them. If it is necessary, roads can also be removed and placed somewhere else. Each action has a different cost.

Figure 1 shows a problem instance of the CityCar domain with 2 garages (g_0, g_1), 9 junctions $(j_0, \ldots, j_8)$, 5 roads $(r_0, \ldots, r_4)$ and 2 cars (c_0, c_1). Car c_0 is at garage g_0, and c_1 is at g_1. The goal junctions of c_0 and c_1 are j_6 and j_8, respectively. At the moment, roads r_0 and r_3 are used. In this domain, we can express the SCK "if any road is available, do not destroy a road which has already been built" in LTL formula $\Box(\forall r : (in\_place(r) \wedge \exists r_1 : \neg in\_place(r_1)) \rightarrow \bigcirc in\_place(r))$. Here $in\_place(r)$ is a predicate which means road r has been put in place. However, with only temporal

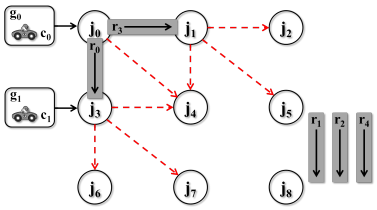

Figure 1: A problem instance of the CityCar domain

specifications, we still cannot find a plan of good quality. Actually, the network topology can be shaped in a variety of ways and a road can be built between any two neighbor junctions at each step (120 alternative ways in total in Figure 1). A mechanism is required to express both spatial and temporal SCK to guide or control the search. For example, a spatio-temporal SCK "each newly-built road should be reachable from some car (§)" will avoid substantial bad road building plans inappropriate for a car to move as early as possible. Under this restriction, only 33 road building actions are feasible in Figure 1. As another example, the spatio-temporal SCK "two disjoint paths (no common locations except the starting and ending points of the paths) between any two junctions do not exist (‡)" is also useful for the sake of pruning plans where redundant roads appear. With this SCK, 12 road building actions can be further reduced. The red dashed-line arrows indicate the places where a road can be built under the guidance of (§) and (‡).

3 Preliminary

This section presents a decidable fragment of separation logic. Let Var be a countable set of spatial variables, Π a countable set of atomic propositions, Loc a finite set of spatial locations, $\mathsf{Val} = Loc \cup \{ nil \}$ the set including the unfilled location nil and locations in Loc, and $\mathbb{B}$ the boolean domain $\{ true, false \}$. A term e can be nil, a variable $x \in Var$ or a location $l \in Loc$. A decidable fragment of separation logic (SL for short) is defined by the following grammar:

$$\phi ::= \pi | e_1 = e_2 | \neg\phi | \phi_1 \vee \phi_2 | \exists x : \phi | e_0 \mapsto \{ e_1, \ldots, e_n \} | \phi_1 \# \phi_2$$

where $\pi \in \Pi$. SL is a variation of the one presented in [Brochenin *et al.*, 2012] by extending single field formula $e \mapsto \{ e' \}$ to multiple fields $e_0 \mapsto \{ e_1, \ldots, e_n \}$ such that a variety of spatial structures can be described other than singly-linked lists.

A state is a triple (I_π, I_v, I_s), where $I_\pi : \Pi \rightharpoonup \mathbb{B}$, $I_v : Var \rightharpoonup \mathsf{Val}$ and $I_s : Loc \rightharpoonup \bigcup_{i=1}^{n} \mathsf{Val}^i$. Here, $\rightharpoonup$ is the notation for partial mapping. I_π, I_v and I_s serve as the evaluations of propositions, spatial variables and spatial cells, respectively. We call I_s the *spatial container* and use $dom(f)$ to denote the domain of mapping f. Given two mappings f_1 and f_2, $f_1 \perp f_2$ means that f_1 and f_2 have disjoint domains, and $f_1 \bullet f_2$ denotes the union of f_1 and f_2 with disjoint domains. The evaluation of a term e relative to a state $s = (I_\pi, I_v, I_s)$ is written as $s[e] \in \mathsf{Val}$: $s[nil] = nil, s[l] = l, s[x] = I_v(x)$. The semantics of SL formulas is defined as follows:

$\boldsymbol{s \models_{SL} \pi}$ iff $s(\pi) = true$. $\boldsymbol{s \models_{SL} e_1 = e_2}$ iff $s[e_1] = s[e_2]$.
$\boldsymbol{s \models_{SL} e_0 \mapsto \{ e_1, \ldots, e_n \}}$ iff $dom(I_s) = \{ s[e_0] \}$ and $I_s(s[e_0]) = (s[e_1], \ldots, s[e_n])$. $\boldsymbol{s \models_{SL} \neg\phi}$ iff $s \not\models_{SL} \phi$.
$\boldsymbol{s \models_{SL} \phi_1 \vee \phi_2}$ iff $s \models_{SL} \phi_1$ or $s \models_{SL} \phi_2$.

$\boldsymbol{s} \models_{SL} \boldsymbol{\phi_1 \# \phi_2}$ iff $\exists I_{s_1}, I_{s_2} : I_{s_1} \perp I_{s_2}$, $I_s = I_{s_1} \bullet I_{s_2}, I_\pi, I_v, I_{s_1} \models_{SL} \phi_1$, and $I_\pi, I_v, I_{s_2} \models_{SL} \phi_2$.

$\boldsymbol{s} \models_{SL} \exists \boldsymbol{x} : \boldsymbol{\phi}$ iff $\exists l \in \mathsf{Val} : I_\pi, I_v[x \Leftarrow l], I_s \models_{SL} \phi$.

Intuitively, $e_0 \mapsto \{e_1, \ldots, e_n\}$ means there are links between e_0 and every e_i ($1 \le i \le n$). Formula $\phi \# \phi'$ describes a space which can be separated into two disjoint parts, one makes ϕ true and the other makes ϕ' true. The following shows some useful derived formulas:

$e \mapsto e_i \stackrel{\text{def}}{=} e \mapsto \{e_1, \ldots, e_i, \ldots, e_n\}$

$ls^1(e_1, e_2) \stackrel{\text{def}}{=} e_1 \mapsto e_2 \quad ls^{n+1}(e_1, e_2) \stackrel{\text{def}}{=} \exists x : e_1 \mapsto x \# ls^n(x, e_2)$

$e_1 \to^+ e_2 \stackrel{\text{def}}{=} \bigvee_{i=1}^{n} ls^i(e_1, e_2) \# true \quad e_1 \to^* e_2 \stackrel{\text{def}}{=} e_1 = e_2 \vee e_1 \to^+ e_2$

Formula $e \mapsto e_i$ denotes there is a link between e and e_i. $ls^n(e_1, e_2)$ precisely describes a path from e_1 to e_2 of length n without any other locations (e_2 is also excluded). $e_1 \to^+ e_2$ and $e_1 \to^* e_2$ indicate that e_2 is reachable from e_1.

4 Spatio-Temporal Logic PPTL$^{\text{SL}}$

This section presents the spatio-temporal logic PPTL$^{\text{SL}}$ and shows how spatio-temporal SCK can be expressed with PPTL$^{\text{SL}}$ formulas.

4.1 Temporalising SL with PPTL

Formulas P of PPTL$^{\text{SL}}$ is defined by the following grammar,

$$P ::= \phi \mid \neg P \mid P_1 \vee P_2 \mid \bigcirc P \mid (P_1, \ldots, P_m)\, prj\, P \mid P^*$$

where ϕ denotes an SL formula. $\bigcirc$ (next), prj (projection) and $*$ (star) are basic temporal operators in PPTL [Duan, 1996; Duan *et al.*, 2008]. A formula is called a state formula if it does not contain any temporal operators, otherwise it is a temporal formula.

An interval $\sigma = \langle s_0, s_1, \ldots \rangle$ is a non-empty sequence of states, possibly finite or infinite. The length of σ, denoted by $|\sigma|$, is ω if σ is infinite, otherwise it is the number of states minus one. We consider the set N_0 of non-negative integers, define $N_\omega = N_0 \cup \{\omega\}$, and extend the comparison operators, $=, <, \le$, to N_ω by considering $\omega = \omega$, and for all $i \in N_0$, $i < \omega$. Moreover, we define $\preceq$ as $\le \setminus \{(\omega, \omega)\}$. With such a notation, $\sigma_{(i \ldots j)} (0 \le i \preceq j \le |\sigma|)$ denotes the sub-interval $\langle s_i, \ldots, s_j \rangle$. The concatenation of σ with another interval σ' is denoted by $\sigma \cdot \sigma'$. Let $\sigma = \langle s_k, \ldots, s_{|\sigma|} \rangle$ be an interval and $r_1, \ldots, r_n$ be integers ($n \ge 1$) such that $0 \le r_1 \le r_2 \le \cdots \le r_n \preceq |\sigma|$. The projection of σ onto $r_1, \ldots, r_n$ is the interval, $\sigma \downarrow (r_1, \ldots, r_n) = \langle s_{t_1}, \ldots, s_{t_m} \rangle$, where $t_1, \ldots, t_m$ are obtained from $r_1, \ldots, r_n$ by deleting all duplicates.

An interpretation of a PPTL$^{\text{SL}}$ formula is a triple $\mathcal{I} = (\sigma, k, j)$ where $\sigma = \langle s_0, s_1, \ldots \rangle$ is an interval, k a non-negative integer and j an integer or ω such that $0 \le k \preceq j \le |\sigma|$. We write $(\sigma, k, j) \models P$ to mean that a formula P is interpreted over a sub-interval $\sigma_{(k \ldots j)}$ of σ with the current state being s_k. The notation $s_k = (I^k_\pi, I^k_v, I^k_s)$ indexed by k represents the k-th state of an interval σ. The satisfaction relation $\models$ for PPTL$^{\text{SL}}$ formulas is defined by:

$\boldsymbol{\mathcal{I} \models \phi}$ iff $s_k \models_{SL} \phi$. $\quad \boldsymbol{\mathcal{I} \models P_1 \vee P_2}$ iff $\mathcal{I} \models P_1$ or $\mathcal{I} \models P_2$.

$\boldsymbol{\mathcal{I} \models \neg P}$ iff $\mathcal{I} \not\models P$. $\boldsymbol{\mathcal{I} \models \bigcirc P}$ iff $k < j$ and $(\sigma, k+1, j) \models P$.

$\boldsymbol{\mathcal{I} \models (P_1, \ldots, P_m)\, prj\, P}$ iff there exist integers $r_0, \ldots, r_m$, and $k = r_0 \le r_1 \le \cdots \le r_m \preceq j$ such that $(\sigma, r_{i-1}, r_i) \models P_i$ for all $1 \le i \le m$ and $(\sigma', 0, |\sigma'|) \models P$ for one of the following σ' :

(a) $r_m < j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma_{(r_m+1 \ldots j)}$

(b) $r_m = j$ and $\sigma' = \sigma \downarrow (r_0, \ldots, r_{i'})$ for some $0 \le i' \le m$.

$\boldsymbol{\mathcal{I} \models P^*}$ iff there are finitely many integers $r_0, \ldots, r_n$ and $k = r_0 \le r_1 \le \cdots \le r_{n-1} \preceq r_n = j (n \ge 0)$ such that $(\sigma, r_{i-1}, r_i) \models P$ for all $1 \le i \le n$; or there are infinitely many integers $r_0, \ldots,$ and $k = r_0 \le r_1 \le r_2 \le \cdots$ such that $\lim_{i \to \infty} r_i = \omega$ and $(\sigma, r_{i-1}, r_i) \models P$ for all $i \ge 1$.

A formula P is satisfied over an interval σ, written as $\sigma \models P$, if $(\sigma, 0, |\sigma|) \models P$ holds. We also have some derived formulas:

$$\varepsilon \stackrel{\text{def}}{=} \neg \bigcirc true \qquad P_1; P_2 \stackrel{\text{def}}{=} (P_1, P_2)\, prj\, \varepsilon$$

$$\Diamond P \stackrel{\text{def}}{=} true; P \qquad \Box P \stackrel{\text{def}}{=} \neg \Diamond \neg P$$

Here, ε denotes an interval with zero length, $\Box$ and $\Diamond$ have the standard meanings as in LTL. Formula $P_1; P_2$ requires that P_1 holds from now on until some point in the future, and from that point on, P_2 holds. Additionally, we borrow the $GOAL$ modality from TLPlan, and only allow it to apply to state formulas. For example, suppose $arrived(c_0, j_6)$ is true in the goal state, then the formula $GOAL(arrived(c_0, j_6))$ will be evaluated to true, otherwise to false.

Table 1: Predicates for the CityCar Domain

Predicate	Description
$arrived(c, j)$	Car c arrives at junction j.
$at\_car\_jun(c, j)$	Car c is at junction j.
$at\_car\_road(c, r)$	Car c is at road r.
$road\_connect(r, j_1, j_2)$	Road r connects junctions j_1 and j_2.

4.2 Specifying Spatio-Temporal SCK with PPTL$^{\text{SL}}$

For the CityCar domain, the predicates involved are listed in Table 1. Despite the common temporal SCK, we can describe several interesting and practical spatio-temporal SCK by PPTL$^{\text{SL}}$. First we define a predicate $at\_car\_jun\_road(c, j)$ indicating that car c is either at junction j or a road with the ending junction being j.

$$at\_car\_jun\_road(c, j) \stackrel{\text{def}}{=} at\_car\_jun(c, j) \vee (\exists j', r : at\_car\_road(c, r) \wedge road\_connect(r, j', j))$$

- The spatio-temporal SCK (§) mentioned in the motivating example can be expressed by:

$$\Box(\forall r, j_1, j_2 : \neg road\_connect(r, j_1, j_2) \to \bigcirc(road\_connect(r, j_1, j_2) \to \exists j_3, c : (at\_car\_jun\_road(c, j_3) \wedge j_3 \to^* j_1)))$$

- The spatio-temporal SCK (‡) can be described by:

$$\Box(\forall j_1, j_2, j_3, j_4 : (\exists r_1, r_2 : road\_connect(r_1, j_1, j_3) \wedge road\_connect(r_2, j_1, j_4)) \to \neg(j_3 \to^* j_2 \# j_4 \to^* j_2))$$

If there are two connections from j_1 to j_3 and j_4 ($road\_connect(r_1, j_1, j_3) \wedge road\_connect(r_2, j_1, j_4)$), we must ensure shared locations from j_3 to j_2 and j_4 to j_2 are nonexistent ($\neg(j_3 \to^* j_2 \# j_4 \to^* j_2)$).

• There is another important spatio-temporal SCK:

$$\Box(\forall c, j_1, j_2 : GOAL(arrived(c, j2)) \wedge at\_car\_jun(c, j_1) \wedge j_1 \rightarrow^+ j_2 \rightarrow \exists r, j_3 : road\_connect(r, j_1, j_3) \wedge j_3 \rightarrow^* j_2 \wedge \bigcirc(\Box at\_car\_jun(c, j_1); at\_car\_road(c, r)))$$

which can be read as "if a car can reach its goal junction via a path, then the car must move through the path". In this SCK, j_1 is the junction where car c at. Suppose there is a path from j_1 to the car's goal j_2 ($j_1 \rightarrow^+ j_2$) and j_3 is the next junction of j_1 in the path ($road\_connect(r, j_1, j_3) \wedge j_3 \rightarrow^* j_2$). The car cannot go anywhere but move on the road from j_1 to j_3.

5 From PPTL$^{\text{SL}}$ to PPTL

In this section, an equisatisfiable translation from PPTL$^{\text{SL}}$ to its restricted form called RPPTL$^{\text{SL}}$ is presented. Then, based on it, a decision procedure of PPTL$^{\text{SL}}$ is obtained.

5.1 Equisatisfiable Translation

Similar to the method in [Calcagno *et al.*, 2005], we first encode SL into a quantifier-free first-order logic.

Definition 1. A bounded variable (proposition) assignment written as $I_v[X](I_\pi[Y])$ denotes the set of variable (proposition) assignments such that $I_v \in I_v[X]$ iff $dom(I_v) = X$ ($I_\pi \in I_\pi[Y]$ iff $dom(I_\pi) = Y$), where $X \subseteq Var (Y \subseteq \Pi)$. A bounded spatial container written as $I_s[n]$ denotes the set of spatial container such that $I_s \in I_s[n]$ iff $|dom(I_s)| \leq n$, where $n \in \mathbb{N}$.

Given a spatial container $I_s \in I_s[n]$, we will use a vector c of n tuples of values, $((c_{1,1}, \ldots, c_{1,m_1}), \ldots, (c_{n,1}, \ldots, c_{n,m_n}))$, to represent I_s. If $c_{i,1} = nil$, the i-th tuple does not represent an active spatial cell. Otherwise if $c_{i,1} = l_i$, the i-th tuple represents that the location l_i contains $c_{i,2}, \ldots,$ and c_{i,m_i}. For instance, $I_s[2]$ is a set consisting of spatial containers with the domain size no more than 2. In addition, a vector that allows the same location occurring more than once should be avoided. For example, $((l_1, l_2), (l_1, l_2))$ or $((l_1, l_2), (l_1, l_3))$ does not represent a valid spatial container. The partial function $vh_n : (\bigcup_{i \in \{ m_1, \ldots, m_n \}} Val^i)^n \rightharpoonup I_s[n]$ is employed to overcome this problem.

$$vh_n(c) = \begin{cases} \text{Undef} & \text{if } \exists i, j : 1 \leq i, j \leq n, i \neq j, \\ & c_{i,1} = c_{j,1}, c_{i,1} \neq nil \text{ and } c_{j,1} \neq nil; \\ \{ (c_{i,1}, \ldots, c_{i,m_i}) \mid 1 \leq i \leq n \} & \text{otherwise.} \end{cases}$$

Let C denote a vector of n tuples of variables. If a vector c with the same size is assigned to C, C will also potentially represent a spatial container. Given a vector of values $c = ((c_{1,1}, \ldots, c_{1,m_1}), \ldots, (c_{n,1}, \ldots, c_{n,m_n}))$ and a vector of variables $C = ((C_{1,1}, \ldots, C_{1,m_1}), \ldots, (C_{n,1}, \ldots, C_{n,m_n}))$, we write $[C \Leftarrow c]$ to denote the pointwise assignment of c to C which can be considered as a set of tuples $\{ (C_{i,j}, c_{i,j}) \mid 1 \leq i \leq n \text{ and } 1 \leq j \leq |C_i| \}$. The notation $|C|(|c|)$ denotes the number of the tuples in $C(c)$, and $|C_i|(|c_i|)$ the number of variables (values) in the i-th tuple.

In the sequel, a set of pairs $D = \{ (x_1, y_1), (x_2, y_2), \ldots \}$ with $\nexists (x, y), (x, z) \in D$ and $y \neq z$ is implicitly interpreted as a function. Conversely, a function f can be interpreted as a set of pairs $\{ (x, f(x)) \mid x \in dom(f) \}$. The standard notation $\bigvee_{x \in \{ l_1, \ldots, l_n \}} \phi$ is used to represent $\phi[l_1/x] \vee \cdots \vee \phi[l_n/x]$, which can be lifted to vectors, i.e., $\bigvee_{C \in \mathsf{Val}^n} \phi$. Similarly, $\bigwedge_{x \in \{ l_1, \ldots, l_n \}} \phi$. $fv(\phi)(fp(\phi))$ denotes the set of free variables (propositions) that appear in ϕ. $C = C' \circledast C''$ defined below is adopted for capturing the semantics of "$\#$".

$$C = C' \circledast C'' \stackrel{\text{def}}{=} \bigwedge_{i=1}^{|C|} \begin{pmatrix} (\bigwedge_{j=1}^{|C_i|} C'_{i,j} = C_{i,j} \wedge C''_{i,1} = nil) \\ \vee (C'_{i,1} = nil \wedge \bigwedge_{j=1}^{|C_i|} C''_{i,j} = C_{i,j}) \end{pmatrix}$$

The function $f(\phi, C)$ takes ϕ and C as two parameters and produces a non-spatial state formula ϕ_s which is defined by: $\phi_s ::= \pi \mid e_1 = e_2 \mid \neg\phi_s \mid \phi_{s_1} \vee \phi_{s_2}$.

$$f(e_0 \mapsto \{ e_1, \ldots, e_n \}, C) \stackrel{\text{def}}{=} \bigvee_{i=1}^{|C|} \begin{cases} C_{i,1} \neq nil \wedge \bigwedge_{j=1, i \neq j}^{|C|} C_{j,1} = nil \wedge \bigwedge_{j=1}^{n+1} C_{i,j} = e_{j-1} & \text{if } |C_i| = n+1; \\ false & \text{otherwise.} \end{cases}$$

$$f(\pi, C) \stackrel{\text{def}}{=} \pi \qquad f(e_1 = e_2, C) \stackrel{\text{def}}{=} e_1 = e_2$$

$$f(\neg\phi, C) \stackrel{\text{def}}{=} \neg f(\phi, C) \qquad f(\exists x : \phi, C) \stackrel{\text{def}}{=} \bigvee_{x \in \mathsf{Val}} f(\phi, C)$$

$$f(\phi_1 \# \phi_2, C) \stackrel{\text{def}}{=} \bigvee_{C', C'' \in \mathsf{Val}^{\Sigma_{i=1}^n |C_i|}} (C = C' \circledast C'' \wedge f(\phi_1, C') \wedge f(\phi_2, C''))$$

where both C' and C'' are vectors of fresh variables.

Lemma 1. For any state formula ϕ, variable vector C and value vector c where $|C| = |c| = n, (I_\pi, I_v, I_s) \in (I_\pi[fp(\phi)], I_v[fv(\phi)], I_s[n]), vh_n(c) = I_s$ and $fv(\phi) \cap fv(C) = \emptyset$,

$$(I_\pi, I_v, I_s) \models_{SL} \phi \quad \text{iff} \quad (I_\pi, I_v \cup [C \Leftarrow c], \emptyset) \models_{SL} f(\phi, C)$$

Proof sketch. The proof proceeds by induction on ϕ. The difficulty lies in the case $\phi_1 \# \phi_2$. In $f(\phi_1 \# \phi_2, C)$ where vector C is employed to simulate the spatial container, the key is to prove $C = C' \circledast C''$ that makes $[C' \Leftarrow c']$ and $[C'' \Leftarrow c'']$ correspond to two separate spatial containers whose union equals to the spatial container represented by $[C \Leftarrow c]$. This obeys the semantics of $\#$. Further, $(I_\pi, I_v, I_s) \models_{SL} \phi_1$ iff $(I_\pi, I_v \cup [C' \Leftarrow c'], \emptyset) \models_{SL} f(\phi_1, C')$ and $(I_\pi, I_v, I_s) \models_{SL} \phi_2$ iff $(I_\pi, I_v \cup [C'' \Leftarrow c''], \emptyset) \models_{SL} f(\phi_2, C'')$ hold by induction hypothesis. This proof is similar to the one for Theorem 1 in [Calcagno *et al.*, 2005]. □

We now translate PPTL$^{\text{SL}}$ formulas to its restricted form RPPTL$^{\text{SL}}$ whose syntax is given below. Compared with PPTL$^{\text{SL}}$, atomics of RPPTL$^{\text{SL}}$ are in ϕ_s instead of ϕ.

$$P_s ::= \phi_s \mid \neg P_s \mid P_{s_1} \vee P_{s_2} \mid \bigcirc P_s \mid (P_{s_1}, \ldots, P_{s_m})\, prj\, P_s \mid P_s^*$$

The translation function F defined below will map a PPTL$^{\text{SL}}$ formula and a variable vector to a RPPTL$^{\text{SL}}$ formula. Also, this function preserves the satisfaction of P.

$$F(\phi, C) \stackrel{\text{def}}{=} f(\phi, C) \qquad F(\neg P, C) \stackrel{\text{def}}{=} \neg F(P, C)$$

$$F(\bigcirc P, C) \stackrel{\text{def}}{=} \bigcirc F(P, C) \qquad F(P^*, C) \stackrel{\text{def}}{=} F(P, C)^*$$

$$F(P_1 \vee P_2, C) \stackrel{\text{def}}{=} F(P_1, C) \vee F(P_2, C)$$

$$F((P_1, ..., P_m)\, prj\, P_0, C) \stackrel{\text{def}}{=} (F(P_1, C), ..., F(P_m, C))\, prj\, F(P_0, C))$$

Theorem 1. For any PPTL$^{\text{SL}}$ formula P, interval σ and vectors $C, c^1, \ldots, c^{|\sigma|}$ where $fv(P) \cap fv(C) = \emptyset$, it has

$$(\sigma, 0, |\sigma|) \models P \quad \text{iff} \quad (\sigma', 0, |\sigma'|) \models F(P, C)$$

Here, $\sigma' = \sigma[(I^i_\pi, I^i_v \cup [C \Leftarrow c^i], \emptyset)/(I^i_\pi, I^i_v, I^i_s)]$ is an interval obtained from σ by replacing (I^i_π, I^i_v, I^i_s) with $(I^i_\pi, I^i_v \cup [C \Leftarrow c^i], \emptyset), (I^i_\pi, I^i_v, I^i_s) \in (I_\pi[fp(P)], I_v[fv(P)], I_s[n]), |C| = |c^i| = n$, and $vh_n(c^i) = I^i_s (1 \leq i \leq |\sigma|)$.

Proof sketch. By Lemma 1, the translation f preserves the satisfiability of state formulas. The rest cases are straightforward to be proved based on a structural induction on P. □

Theorem 1 enables one to only concentrate on RPPTL$^{\text{SL}}$ instead of PPTL$^{\text{SL}}$ for easily attaining a decision procedure.

5.2 A Decision Procedure

Since RPPTL$^{\text{SL}}$ and PPTL have the same syntax structures except for atomic formulas, we are inspired to reuse the decision procedure of PPTL for RPPTL$^{\text{SL}}$.

To do so, similar to PPTL, we define normal form of RPPTL$^{\text{SL}}$ formulas: $P_s \equiv \bigvee_{j=1}^{n'} (P_{e_j} \wedge \varepsilon) \vee \bigvee_{i=1}^{n} (P_{c_i} \wedge \bigcirc P'_i)$, where P_{e_j} and P_{c_i} are conjunctions of atomic formulas or their negations, and P'_i is a general RPPTL$^{\text{SL}}$ formula. Informally, the normal form of a formula P_s decomposes itself into current states (P_{e_j}, P_{c_i}) and future formulas (P'_i) that shows which parts of P_s are still to be satisfied after reaching the current states. One can derive that any RPPTL$^{\text{SL}}$ formula is able to be written to its normal form.

Theorem 2. Any RPPTL$^{\text{SL}}$ formula P_s can be rewritten into its normal form.

Proof sketch. The normal form translation of PPTL formulas is mainly based on logic laws relevant to temporal operators. Since the only difference between RPPTL$^{\text{SL}}$ and PPTL is the state formulas which are free of temporal operators, RPPTL$^{\text{SL}}$ inherits all the temporal laws in PPTL [Duan, 1996; Duan *et al.*, 2008]. Hence we can translate P_s to its normal form in the same way for translating a PPTL formula to normal form. □

Consequently, the decision procedure of PPTL which relies heavily on normal form can be reused for checking the satisfiability of both RPPTL$^{\text{SL}}$ and PPTL$^{\text{SL}}$ formulas. The progression algorithm of [Bacchus and Kabanza, 2000] decomposes an LTL formula into current state formulas and future temporal formulas. The idea of normal form of RPPTL$^{\text{SL}}$ is somewhat similar. This allows to directly use an incremental way to check SCK as the progression algorithm in [Bacchus and Kabanza, 2000] by evaluating state formulas attained during normal form translation at each time stamp.

6 Implementation and Experiments

We have implemented a planner *S-TSolver* which handles SCK in collaborating with the classical forward chaining algorithm and SMT solver Z3 [De Moura and Bjørner, 2008]. To plan with *S-TSolver*, a domain, an instance and SCK expressed in PPTL$^{\text{SL}}$ formulas are taken as input. At each step, the forward chaining algorithm responds for exploring the current world of the instance by executing an action, while the equisatisfiable translation (Theorem 1) and the progression technique (Theorem 2) are employed to transform each SCK into its normal form. The current world of the instance and the current state formulas of SCK which are both within the scope of SMT [Barrett *et al.*, 2009] are encoded into SMT-LIB format [Barrett *et al.*, 2010] and fed to Z3 for satisfiability checking. If Z3 returns "sat", the action is added to the plan sequence which is empty initially; otherwise the forward chaining algorithm randomly chooses another action which can be executed to repeat the above process until the goal is achieved or all the worlds are explored.

We evaluate the performance of *S-TSolver* in planning on two domains: *CityCar* and *Transport*. We choose these two domains because they are typical domains with both spatial and temporal constraints such that spatio-temporal SCK can be used for planning. Moreover, the two domains are representative: the spatial structure changes over time for the CityCar domain and is fixed for the Transport domain. We compare the results of *S-TSolver* with the top three planners [Vallati *et al.*, 2014] in IPC2014 on the two domains, respectively. All the experiments are conducted on a PC running Ubuntu 16.04 on an Intel® Core™ i3-550 CPU 3.20GHz and 8Gb of RAM.

Original from IPC2008, the Transport domain is a variant of the well-known domain logistics. The task of this domain is to transport several packages from the initial locations to the desired destinations. A package is transported from one location to another one by loading it into a truck, driving the truck to the destination, and unloading the truck. Analogous to the CityCar domain, there is a network for trucks to move. Therefore, we can specify some spatio-temporal SCK for this domain, e.g. "each truck must move to the nearest location which is the goal of some package loaded by the truck or where some package is required to be transported". Note that "nearest" means the minimum steps for a truck to move. We select some instances from this domain.

The experiment results are demonstrated in Table 2. The first column shows the problem instances. The column "cost" presents the lowest plan cost among the plans found by a planner. The column "time" is the search time and the column "len" is the length of the plan. Usually the higher the cost, the longer the length. "*S-TSolver*(T)" is the result of *S-TSolver* with only temporal SCK, and "*S-TSolver*(S-T)" is the result with both temporal and spatio-temporal SCK. "time-out" indicates a planner cannot provide a plan within 10 minutes. Every lowest cost and best time are highlighted in bold.

For the CityCar domain, the top three tools performed well in IPC2014 are *arvandherd*, *jasper* and *uniform*. From Table 2, it is observed that both *S-TSover*(T) and *S-TSover*(S-T) are capable of solving all the problem instances, but *arvandherd*, *jasper* and *uniform* can only solve part of them. With regard to search time, the time spent by *S-TSolver*(T) is less than *S-TSolver*(S-T) since extra time is consumed to deal with spatio-temporal SCK. The time spent by *uniform* is less than 30s for the solved instances as a preset time limitation (≈30s) is given in this tool for each search iteration. If no plans are found within the limitation, another search will be started. *arvandherd* and *jasper* give no solutions to some difficult in-

Table 2: Experiment results: CityCar Domain and Transport Domain

CityCar	*S-TSolver*(T)			*S-TSolver*(S-T)			*arvandherd*			*jasper*			*uniform*		
instances	cost	time(s)	len	cost	time(s)	len	cost	time(s)	len	cost	time(s)	len	cost	time(s)	len
p01	102	**0.04**	30	**70**	0.68	17	**70**	2.07	17	**70**	2.03	17	**70**	0.18	17
p02	152	**0.17**	35	128	4.6	29	136	12.89	28	549	2.55	55	**118**	0.74	29
p03	106	**0.34**	27	116	2	28	116	16.2	28	**100**	13.84	32	**100**	1.26	32
p04	264	**0.31**	67	**136**	8.48	41	202	11.19	39	298	8.89	51	142	20.12	37
p05	86	**0.05**	39	126	5.22	41	100	9.31	33	492	4.21	67	**86**	2.02	39
p06	594	**1.08**	92	**138**	8.11	28	184	18.78	26	287	48.92	34	176	26.22	26
p07	542	**0.87**	62	**158**	7.86	28	174	29.39	25	214	31.36	27	—	time-out	—
p08	294	**0.37**	30	**114**	4.79	22	124	27.6	22	194	20.18	26	—	time-out	—
p09	322	**0.57**	46	**128**	5.71	28	158	57.74	29	342	175.12	44	—	time-out	—
p10	206	**0.51**	42	188	16.04	30	**182**	30.91	35	—	time-out	—	—	time-out	—
p11	438	**0.75**	64	**192**	11.8	36	292	59.97	40	211	84.43	35	—	time-out	—
p12	348	**0.66**	48	**146**	19.5	38	344	14.53	47	320	19.24	41	146	27.68	38
p13	754	**1.89**	138	**232**	19.88	60	432	152.5	62	467	323	57	—	time-out	—
p14	250	**0.3**	48	**164**	16.82	48	—	time-out	—	—	time-out	—	—	time-out	—
p15	1226	**2.16**	150	**264**	28.23	54	355	34.9	50	453	275.84	63	—	time-out	—
p16	524	**0.82**	110	**150**	10.07	44	209	83.7	45	—	time-out	—	—	time-out	—
p17	2784	**5.62**	412	**204**	57.48	50	444	174.97	66	—	time-out	—	—	time-out	—
p18	334	**0.49**	90	**170**	43.05	56	—	time-out	—	—	time-out	—	—	time-out	—
p19	1530	**3.48**	242	**304**	35.43	76	503	250.35	67	—	time-out	—	—	time-out	—
p20	610	**1.15**	100	**230**	33.08	48	—	time-out	—	—	time-out	—	—	time-out	—
Transport	*S-TSolver*(T)			*S-TSolver*(S-T)			*yahsp3*			*mercury*			*dae_yahsp*		
instances	cost	time(s)	len	cost	time(s)	len	cost	time(s)	len	cost	time(s)	len	cost	time(s)	len
p01	1841	1.86	121	680	6.35	55	1696	**0.13**	114	**617**	66.16	55	954	157.54	69
p02	1524	**1.35**	98	**633**	8.28	54	1163	572.6	85	639	52.36	56	892	35.55	71
p03	4198	7.13	248	1084	24.16	78	2018	**0.67**	150	**848**	170.9	69	1422	179.84	102
p04	5819	31.17	368	890	21.59	76	2308	**0.51**	200	**800**	148.34	71	1774	90.15	127
p05	3952	**10.18**	291	**837**	41.77	79	2443	551.42	233	907	147.26	92	1877	109.74	156
p06	11569	**34.04**	802	1088	70.84	107	2569	265.45	240	**1040**	102.04	104	2452	204.5	206
p07	3961	26.75	316	**1131**	145.36	118	3031	**1.36**	298	1220	107.86	124	2895	69.41	236
p08	1975	**0.42**	64	**990**	13.07	34	1684	75.96	52	1003	80.18	34	1141	109.11	40
p09	6141	**2.49**	227	**691**	8.38	55	2032	524.17	73	1211	31.5	47	1833	123.34	62
p10	21113	**24.29**	845	**1817**	83.82	70	3550	108.58	131	1836	98.42	69	2512	132.92	91
p11	31871	20.48	1317	**1869**	111.29	69	4997	**0.24**	198	2001	87.12	76	4693	85.09	174
p12	61059	73.51	2481	**2296**	133.66	82	6034	**0.39**	222	3393	56.58	108	4470	162.98	157
p13	1429	0.39	52	1054	1.86	35	2723	**0.07**	69	**1031**	70.48	39	1632	14.26	54
p14	4098	**2.01**	56	1287	5.57	44	3551	19.66	101	**1073**	117.8	40	2277	34.43	64
p15	6316	**6.61**	165	2201	17.66	60	3904	542.81	97	**1899**	67.74	60	2549	19.86	82
p16	4937	**1.38**	121	1286	7.87	49	3026	412.28	92	**1173**	64.8	48	2155	61.27	67
p17	4962	25.88	156	**1532**	21.01	54	2429	**0.7**	97	1591	102.12	58	2130	264.08	83
p18	2644	11.81	75	930	**11.14**	33	1349	235.7	49	**868**	11.78	33	1093	213.57	38
p19	11657	29.78	370	3131	62.02	94	6479	**0.57**	227	**2540**	104.78	91	5268	252.02	146
p20	20682	30.14	797	**2211**	143.39	90	5982	**0.83**	190	3073	96.64	112	5369	283.23	166

stances. They spend even hundreds of seconds to search for a plan. For the plan cost, the plans generated by *S-TSolver*(S-T) have better quality than other tools. This is because the strategies expressed by spatio-temporal SCK can easily guide a car to its goal without doing many useless actions.

For the Transport domain, *yahsp3*, *mercury* and *dae_yahsp* are the best three tools in IPC2014. When running *yahsp3*, it is able to give a quick plan in a short time. However, it will further take a long time to improve the plan. Sometimes *S-TSolver*(S-T) takes more time than *mercury* because of the overhead in analyzing the spatio-temporal SCK. Consider the plan cost, *S-TSolver*(T) produces plans with poor quality since a truck does not know where to go and just attempts to move randomly. The plan costs of *S-TSolver*(S-T) are much better than those of *yahsp3* and *dae_yahsp*, but in the same level with *mercury*. This is because the spatio-temporal SCK we use is a little coarse. As mentioned earlier, "nearest" path means the minimum steps from one location to another, not the length of roads in the path. For example, $ls^{n-1}(e_1, e_2)$ is "nearer" than $ls^n(e_1, e_2)$. Although this is true in most situations, special cases may still occur that makes a "nearer" path has longer length. A potential way to solve it is to add data field in the model of our logic. As an example, $l_0 \mapsto \{(d_1, l_1), (d_2, l_2)\}$ not only denotes the link between l_0 and l_1 (l_0 and l_2), but also the data property d_1 (d_2), e.g. length, of the link.

As a summary, our approach is helpful in finding plans of good quality and speeding up planning for the domains with both temporal and spatial constraints.

7 Conclusion

In this paper, an expressive two-dimensional logic PPTL$^{\text{SL}}$ is formalized for specifying both spatial and temporal dimensions of SCK. Satisfiability of PPTL$^{\text{SL}}$ is proved decidable based on an equisitisfiable translation from PPTL$^{\text{SL}}$ to RPPTL$^{\text{SL}}$. Based on it, a planner is implemented and evaluated on two benchmark domains. Experiments demonstrate that our method prunes substantial irrelevant search space and finds plans of good quality.

In the future, we plan to do more experiments on the planning domains (e.g. Tidybot-IPC2011, TPP-IPC2006, and DriverLog-IPC2002 etc.) that are suitable for spatio-temporal SCK. Also, we would like to adopt the advanced heuristic algorithms in order to further improve efficiency of the tool. Another research direction is to relate our work to other study that has a richer notion of space, e.g. first-order theories of spatio-temporal change such as [Hazarika, 2005].

References

[Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1):123–191, 2000.

[Barrett *et al.*, 2009] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. 2009.

[Barrett *et al.*, 2010] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The smt-lib standard version 2.6. 2010.

[Belouaer and Maris, 2010] Lamia Belouaer and Frédéric Maris. Smt spatio-temporal planning. *nd Int*, 209:6, 2010.

[Boukharrou *et al.*, 2015] Radja Boukharrou, Jean-Michel Ilié, and Djamel Eddine Saïdouni. Spatio-temporal planning for mobile ambient agents. *Procedia Computer Science*, 56:96–103, 2015.

[Brochenin *et al.*, 2012] Rémi Brochenin, Stéphane Demri, and Etienne Lozes. On the almighty wand. *Information and Computation*, 211:106–137, February 2012.

[Calcagno *et al.*, 2005] Cristiano Calcagno, Philippa Gardner, and Matthew Hague. From separation logic to first-order logic. In Vladimiro Sassone, editor, *FOSSACS*, volume 3441 of *Lecture Notes in Computer Science*, pages 395–409, Edinburgh, UK, April 4–8 2005. Springer Berlin Heidelberg.

[Chrpa and Barták, 2016] Lukáš Chrpa and Roman Barták. Guiding planning engines by transition-based domain control knowledge. In *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2016.

[De Moura and Bjørner, 2008] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[Duan *et al.*, 2008] Zhenhua Duan, Cong Tian, and Li Zhang. A decision procedure for propositional projection temporal logic with infinite models. *Acta Informatica*, 45(1):43–78, 2008.

[Duan, 1996] Zhenhua Duan. *An Extended Interval Temporal Logic and a Framing Technique for Temporal Logic Programming*. PhD thesis, University of Newcastle upon Tyne, 1996.

[Fern *et al.*, 2004] Alan Fern, Sung Wook Yoon, and Robert Givan. Learning domain-specific control knowledge from random walks. In *ICAPS*, pages 191–199, 2004.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

[Georgievski and Aiello, 2015] Ilche Georgievski and Marco Aiello. Htn planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156, 2015.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

[Haghighi *et al.*, 2015] Iman Haghighi, Austin Jones, Zhaodan Kong, Ezio Bartocci, Radu Grosu, and Calin Belta. Spatel: a novel spatial-temporal logic and its applications to networked systems. In *HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pages 189–198, 2015.

[Hazarika, 2005] Shyamanta M Hazarika. *Qualitative spatial change: space-time histories and continuity*. PhD thesis, The University of Leeds, 2005.

[Huang *et al.*, 1999] Yi-Cheng Huang, Bart Selman, Henry Kautz, et al. Control knowledge in planning: benefits and tradeoffs. In *AAAI/IAAI*, pages 511–517, 1999.

[Kvarnström and Magnusson, 2003] Jonas Kvarnström and Martin Magnusson. Talplanner in ipc-2002: Extensions and control rules. *Journal of Artificial Intelligence Research*, 20:343–377, 2003.

[Merz *et al.*, 2003] Stephan Merz, Martin Wirsing, and Júlia Zappe. A spatio-temporal logic for the specification and refinement of mobile systems. In *FASE 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, pages 87–101, 2003.

[Nau *et al.*, 2003] Dana S Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *J. Artif. Intell. Res.(JAIR)*, 20:379–404, 2003.

[Reynolds, 2002] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th Annual IEEE Symposium on Logic in Computer Science, 2002.*, pages 55–74, Washington, DC, USA, 2002. IEEE Computer Society.

[Tian and Duan, 2008] Cong Tian and Zhenhua Duan. Propositional projection temporal logic, büchi automata and ω-regular expressions. In Manindra Agrawal, Ding-Zhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, pages 47–58, Xi'an, China, April 25-29 2008. Springer Berlin Heidelberg.

[Vallati *et al.*, 2014] M Vallati, L Chrpa, and TL McCluskey. The eighth international planning competition. *Description of Participant Planners of the Deterministic Track*, 2014.

[Vallati *et al.*, 2015] Mauro Vallati, Lukáš Chrpa, Marek Grzes, Thomas L McCluskey, Mark Roberts, and Scott Sanner. The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98, 2015.

[Yoon *et al.*, 2008] Sungwook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9(Apr):683–718, 2008.