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\_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 \[\Box (\forall r : \text{in}_{\text{place}}(r) \land \exists r_1 : \neg \text{in}_{\text{place}}(r_1)) \rightarrow \Box \text{in}_{\text{place}}(r)\]. Here \(\text{in}_{\text{place}}(r)\) is a predicate which means road \(r\) has been put in place. However, with only temporal 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 (\(\langle r \rangle\))” 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 (\(\langle r \rangle\))” 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 (\(\langle r \rangle\)) and (\(\langle r \rangle\)).

3 Preliminary

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

\[
\phi ::= \pi[e_1 = e_2] \lor \neg \phi_1 \lor \phi_2 \lor \exists x : \phi[e_0 \mapsto \{e_1, \ldots, e_n\}] \phi_1 \# \phi_2
\]

where \(\pi \in \Pi\). SL is a variation of the one presented in [Brochenin et al., 2012] by extending single field formula \(e \mapsto \langle e'\rangle\) 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 \((I_\pi, I_\mathcal{L}oc, I_\mathcal{V})\), where \(I_\pi : \Pi \rightarrow \mathcal{B}\), \(I_\mathcal{V} : \mathcal{V}ar \rightarrow \mathcal{V}al\) and \(I_\mathcal{L}oc : \mathcal{L}oc \rightarrow \mathcal{V}al^\pi\). Here, \(\rightarrow\) is the notation for partial mapping. \(I_\pi, I_\mathcal{L}oc\) and \(I_\mathcal{V}\) serve as the evaluations of propositions, spatial variables and spatial cells, respectively. We call \(I_\pi\) the spatial container and use \(dom(f)\) to denote the domain of mapping \(f\). Given two mappings \(f_1\) and \(f_2\), \(f_1 \bot f_2\) 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 \(s = (I_\pi, I_\mathcal{L}oc, I_\mathcal{V})\) is written as \(s[e] \in \mathcal{V}al\): \(s[\text{nil}] = \text{nil}\), \(s[l] = l\), \(s[x] = I_\mathcal{V}(x)\). The semantics of SL formulas is defined as follows:

\[
s \models_{SL} \pi \text{ if } s(\pi) = True. \quad s \models_{SL} e_1 = e_2 \text{ if } s[e_1] = s[e_2].
\]

\[
s \models_{SL} e_0 \mapsto \{e_1, \ldots, e_n\} \text{ if } dom(I_\mathcal{V}) = \{s[e_0]\} \text{ and } \forall e : \neg s[e] \implies s \not\models_{SL} \phi. \quad s \models_{SL} \phi_1 \lor \phi_2 \text{ if } s \not\models_{SL} \phi_1 \lor s \not\models_{SL} \phi_2.
\]
Formula $e \rightarrow e_1$ denotes there is a link between $e$ and $e_1$. $ls^n(e_1,e_2)$ precisely describes a path from $e_1$ to $e_2$ of length $n$ without any other locations ($e_2$ is also excluded). $e_1 \rightarrow^+ e_2$ and $e_1 \rightarrow^e e_2$ indicate that $e_2$ is reachable from $e_1$.

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

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

4.1 Temporal SL with PPTL

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

$$P ::= \phi \mid \neg P \mid P_1 \lor P_2 \mid \bigcirc P \mid (P_1, \ldots, P_m)_{prj} P \mid P^*$$

where $\phi$ 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 $\sigma = (s_0, s_1, \ldots)$ is a non-empty sequence of states, possibly finite or infinite. The length of $\sigma$, denoted by $|\sigma|$, is $\omega$ if $\sigma$ 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, $<, <, \le, >, >$ by considering $\omega = \omega$, and for all $i \in N_\omega$, $i < \omega$. Moreover, we define $\leq$ as $\leq \omega$ where $\omega$. With such a notation, $\sigma(i,j)$ holds for all $i \leq j \leq |\sigma|$. The sub-interval $s_{i,j}$ denotes the concatenation of $\sigma$ with another interval $\sigma'$ denoted by $\sigma \cdot \sigma'$. Let $\sigma = (s_k, \ldots, s_{|\sigma|})$ be an interval and $r_1, \ldots, r_n$ be integers (all $\geq 0$) such that $0 \leq r_1 \leq r_2 \leq \cdots \leq r_n \leq |\sigma|$. The projection of $\sigma$ onto $r_1, \ldots, r_n$ is the interval, $\sigma(r_1, \ldots, r_n) = (s_{r_1}, \ldots, s_{r_n})$, where $t_1, \ldots, t_m$ are obtained from $r_1, \ldots, r_n$ by deleting all duplicates.

An interpretation of a PPTL$^\mathbb{SL}$ formula is a triple $\mathcal{I} = (\sigma, k, j)$ where $\sigma = (s_0, s_1, \ldots)$ is an interval, $k$ a non-negative integer and $j$ an integer such that $0 \leq j \leq |\sigma|$. We write $(\sigma, k, j) \models P$ to mean that a formula $P$ is interpreted over a sub-interval $\sigma(k,j)$ of $\sigma$ with the current state being $s_k$. The notation $s_k = (I^L_k, I^U_k, I^L_k)$ indexed by $k$ represents the $k$-th state of an interval $\sigma$. The satisfaction relation $\models$ for PPTL$^\mathbb{SL}$ formulas is defined by:

$$\mathcal{I} \models \phi$$ if $s_k \models_{SL} \phi$.

$$\mathcal{I} \models P_1 \lor P_2$$ if $\mathcal{I} \models P_1$ or $\mathcal{I} \models P_2$.

$$\mathcal{I} \models \neg P$$ if $\mathcal{I} \not\models P$.

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

(a) $r_m < j$ and $\sigma' = \sigma \cup (r_0, \ldots, r_m) \cup \sigma(r_{m+1}, j)$

(b) $r_m = j$ and $\sigma' = \sigma \cup (r_0, \ldots, r_m)$ for some $0 \leq i' < m$. $\mathcal{I} \models P$ if there are finitely many intervals $r_0, \ldots, r_m$ and $k = r_0 \leq r_1 \leq \cdots \leq r_{n-1} \leq r_n = j(n > 0)$ such that $(\sigma, r_{i-1}, r_i) \models P$ for all $1 \leq i \leq n$; or there are infinitely many intervals $r_0, \ldots, j_k$ and $k = r_0 \leq r_1 \leq \cdots \leq r_{m-1} \leq r_m = \omega$ such that $\lim_{i \rightarrow \infty} r_i = \omega$ and $(\sigma, r_{i-1}, r_i) \models P$ for all $i \geq 1$.

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

$$e \models \bigcirc true$$

$$P_1, P_2 \models (P_1, P_2)_{prj} e$$

$$\bigcirc P \models true$$

$$\square P \models \neg \bigcirc \neg P$$

Here, $e$ denotes a formula with zero length, $\square$ and $\bigcirc$ 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 $\text{arrived}(c_0,j_0)$ is true in the goal state, then the formula $\text{GOAL}(\text{arrived}(c_0,j_0))$ will be evaluated to true, otherwise to false.

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

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$^\mathbb{SL}$. First we define a predicate $\text{at-car\_jun\_road}(c, j)$ indicating that car $c$ is either at junction $j$ or a road with the ending junction being $j$.

\[ \text{at-car\_jun\_road}(c, j) \equiv \text{at-car\_jun}(c, j) \lor (\exists j', r : \text{at-car\_road}(c, r) \land \text{road\_connect}(r, j, j')) \]

- The spatio-temporal SCK (5) mentioned in the motivating example can be expressed by:

$$\square(\forall r, j_1, j_2 : \neg \text{road\_connect}(r, j_1, j_2) \rightarrow (\bigcirc \text{road\_connect}(r, j_1, j_2) \rightarrow \exists j_3, c : \text{at\_car\_road}(c, j_3) \land j_3 \rightarrow\!\!\!\!\!\!\!\!\!\!\text{\_jun}(j_1, j_3))$$

- The spatio-temporal SCK (7) can be described by:

\[ (\forall v_1, j_2, j_3, j_4 : (\exists r_1, r_2 : \text{road\_connect}(r_1, j_1, j_3) \land \text{road\_connect}(r_2, j_1, j_4)) \rightarrow (j_3 \rightarrow\!\!\!\!\!\!\!\!\!\!\text{\_jun}(j_3, j_4) \land \text{road\_connect}(r_2, j_1, j_4) \rightarrow \neg (j_3 \rightarrow\!\!\!\!\!\!\!\!\!\!\text{\_jun}(j_3, j_4))) \]

If there are two connections from $j_1$ to $j_3$ and $j_4$ \((\text{road\_connect}(r_1, j_1, j_3) \land \text{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 non-existent \((\neg (j_3 \rightarrow\!\!\!\!\!\!\!\!\!\!\text{\_jun}(j_3, j_4) \land \text{road\_connect}(r_2, j_1, j_4) \rightarrow \neg (j_3 \rightarrow\!\!\!\!\!\!\!\!\!\!\text{\_jun}(j_3, j_4))))."}
5 From PPTLst to PPTL

In this section, an equisatisfiable translation from PPTLst to its restricted form called RPPTLst is presented. Then, based on it, a decision procedure of PPTLst 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 \( I_x[X] [I_y[Y]] \) denotes the set of variables (proposition) assignments such that \( I_x \in I_y[X] \) iff \( \text{dom}(I_x) = X \) and \( I_y \in I_x[Y] \) iff \( \text{dom}(I_y) = Y \), where \( X \subseteq \text{Var}(Y) \subseteq \Pi \). A bounded spatial container written as \( I_s[n] \) denotes the set of spatial container such that \( I_s \in I_s[n] \) iff \( \text{dom}(I_s) \leq n \), 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, \((c_1,\ldots,c_{1,m_1}),\ldots,(c_n,\ldots,c_{n,m_n})\), to represent \( I_s \). If \( c_{i,j} = \text{nil} \), the \( i \)-th tuple does not represent an active spatial cell. Otherwise if \( c_{i,j} = l_i \), 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 \( \nu_{n}(c) \colon \{\cup_{i=1}^{n} M_{i}\}^{n} \mapsto I_s[n] \) is employed to overcome this problem.

Let \( C \) denote a vector of \( n \) tuples of variables. If a vector \( c \) with the same size as \( C \), \( C \) will also potentially represent a spatial container. Given a vector of values \( c = ((c_1,\ldots,c_{1,m_1}),\ldots,(c_n,\ldots,c_{n,m_n})) \) and a vector of variables \( C = ((C_1,\ldots,C_{1,m_1}),\ldots,(C_n,\ldots,C_{n,m_n})) \), we write \( C \models c \) to denote the pointwise assignment of \( C \) to \( c \). The number of the tuples in \( C \) and \( |C| \) can be considered as a set of tuples \( \{ (C_i, |C|_i) \mid 1 \leq i \leq n \text{ and } 1 \leq j \leq |C_i| \} \). The notation \( |C|_i \) denotes the number of the tuples in \( C_i \), and \( |C|_i \) 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 \( f(x,y) \cdot (x,z) \in D \) and \( y \neq z \) is implicitly interpreted as a function. Conversely, a function \( f \) can be interpreted as a set of pairs \( \{(x,f(x)) \mid x \in \text{dom}(f)\} \). The standard notation
Theorem 1. For any PPTL$_{SL}$ formula $P$, interval $\sigma$ and vectors $C, c^1, \ldots, c^{|\sigma|}$ where $f_v(P) \cap f_v(C) = \emptyset$, it has $(\sigma, 0, |\sigma|) \models P$ iff $(\sigma', 0, |\sigma'|) \models F(P, C)$.

Here, $\sigma' = \sigma[I_2^j, I_1^j] \cup [C \leftarrow c^1], 0]/[I_2^j, I_1^j, I_3^j]$ is an interval obtained from $\sigma$ by replacing $I_2^j, I_1^j, I_3^j$ with $I_3^j, I_1^j, I_2^j$ with $[C \leftarrow c^1], 0)$. $(I_2^j, I_1^j, I_3^j) \in \{I_2^j, f_v(P), I_3^j[n]\}$, $|C| = |c^1| = n$, and $v_h v(c^j) = I_3^j(1 \leq i \leq |\sigma|)$.

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 $\sigma$. □

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

5.2 A Decision Procedure

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

To do so, similar to PPTL, we define normal form of RPPTL$_{SL}$ formulas: $P_s = \bigvee_{j=1}^{n_1} (P_{c^j} \land \varepsilon) \lor \bigvee_{i=1}^{n_2} (P_{c^i} \land \bigcirc P_i')$, where $P_{c^j}$ and $P_{c^i}$ are conjunctions of atomic formulas or their negations, and $P_i'$ is a general RPPTL$_{SL}$ formula. Informally, the normal form of a formula $P_s$ decomposes itself into current states $(P_{c^j}, P_{c^i})$ 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$_{SL}$ formula is able to be written to its normal form.

Theorem 2. Any RPPTL$_{SL}$ 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$_{SL}$ and PPTL is the state formulas which are free of temporal operators, RPPTL$_{SL}$ 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$_{SL}$ and PPTL$_{SL}$ 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$_{SL}$ 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 Björner, 2008]. To plan with S-TSolver, a domain, an instance and SCK expressed in PPTL$_{SL}$ 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 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® CoreTM 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-TSolver(T) and S-TSolver(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

<table>
<thead>
<tr>
<th>CityCar</th>
<th>S-TSolver(T)</th>
<th>S-TSolver(S-T)</th>
<th>arrvandherd</th>
<th>jasper</th>
<th>SCK</th>
<th>uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td>instances</td>
<td>cost</td>
<td>time(s)</td>
<td>len</td>
<td>cost</td>
<td>time(s)</td>
<td>len</td>
</tr>
<tr>
<td>p01</td>
<td>102</td>
<td>0.04</td>
<td>30</td>
<td>70</td>
<td>0.08</td>
<td>27</td>
</tr>
<tr>
<td>p02</td>
<td>152</td>
<td>0.17</td>
<td>35</td>
<td>128</td>
<td>4.6</td>
<td>29</td>
</tr>
<tr>
<td>p03</td>
<td>106</td>
<td>0.34</td>
<td>27</td>
<td>116</td>
<td>2.2</td>
<td>28</td>
</tr>
<tr>
<td>p04</td>
<td>264</td>
<td>0.31</td>
<td>67</td>
<td>136</td>
<td>8.48</td>
<td>41</td>
</tr>
<tr>
<td>p05</td>
<td>86</td>
<td>0.05</td>
<td>39</td>
<td>126</td>
<td>5.22</td>
<td>41</td>
</tr>
<tr>
<td>p06</td>
<td>594</td>
<td>1.08</td>
<td>92</td>
<td>138</td>
<td>8.11</td>
<td>28</td>
</tr>
<tr>
<td>p07</td>
<td>542</td>
<td>0.87</td>
<td>62</td>
<td>158</td>
<td>7.86</td>
<td>18</td>
</tr>
<tr>
<td>p08</td>
<td>294</td>
<td>0.37</td>
<td>30</td>
<td>114</td>
<td>4.79</td>
<td>22</td>
</tr>
<tr>
<td>p09</td>
<td>322</td>
<td>0.57</td>
<td>46</td>
<td>128</td>
<td>5.71</td>
<td>28</td>
</tr>
<tr>
<td>p10</td>
<td>206</td>
<td>0.51</td>
<td>42</td>
<td>188</td>
<td>16.04</td>
<td>30</td>
</tr>
<tr>
<td>p11</td>
<td>438</td>
<td>0.75</td>
<td>64</td>
<td>192</td>
<td>11.8</td>
<td>36</td>
</tr>
<tr>
<td>p12</td>
<td>348</td>
<td>0.66</td>
<td>48</td>
<td>146</td>
<td>19.5</td>
<td>38</td>
</tr>
<tr>
<td>p13</td>
<td>754</td>
<td>1.89</td>
<td>138</td>
<td>232</td>
<td>19.88</td>
<td>40</td>
</tr>
<tr>
<td>p14</td>
<td>250</td>
<td>0.3</td>
<td>48</td>
<td>164</td>
<td>16.82</td>
<td>48</td>
</tr>
<tr>
<td>p15</td>
<td>1226</td>
<td>2.16</td>
<td>150</td>
<td>264</td>
<td>28.23</td>
<td>54</td>
</tr>
<tr>
<td>p16</td>
<td>524</td>
<td>0.82</td>
<td>110</td>
<td>150</td>
<td>10.07</td>
<td>44</td>
</tr>
<tr>
<td>p17</td>
<td>2784</td>
<td>5.62</td>
<td>412</td>
<td>204</td>
<td>57.48</td>
<td>50</td>
</tr>
<tr>
<td>p18</td>
<td>334</td>
<td>0.49</td>
<td>90</td>
<td>170</td>
<td>43.05</td>
<td>56</td>
</tr>
<tr>
<td>p19</td>
<td>1530</td>
<td>3.48</td>
<td>242</td>
<td>304</td>
<td>35.43</td>
<td>76</td>
</tr>
<tr>
<td>p20</td>
<td>610</td>
<td>1.15</td>
<td>100</td>
<td>230</td>
<td>33.08</td>
<td>48</td>
</tr>
</tbody>
</table>

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, $l_{s^{1.1}}(c_1, e_2)$ is “nearest” than $l_{s^{i}}(c_1, e_2)$. Although this true in most situations, special cases may still occur that makes a “near- er” path has longer length. A potential way to solve it is to add data field in the model of our logic. As an example, $l_0 \rightarrow \{ (d_1, l_1), (d_2, l_2) \}$ 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_{SL} is formalized for specifying both spatial and temporal dimensions of SCK. Satisfiability of PPTL_{SL} is proved decidable based on an equisatisfiable translation from PPTL_{SL} to RPPTL_{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, TIPP-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 s-tudy that has a richer notion of space, e.g. first-order theories of spatio-temporal change such as [Hazarika, 2005].
References


