STRIPS Planning in Infinite Domains

Caelan Reed Garrett
MIT CSAIL
Cambridge, MA 02139
caelan@csail.mit.edu

Tomás Lozano-Pérez
MIT CSAIL
Cambridge, MA 02139
tlp@csail.mit.edu

Leslie Pack Kaelbling
MIT CSAIL
Cambridge, MA 02139
lpk@csail.mit.edu

I. Introduction

Many important planning domains naturally occur in continuous spaces involving complex constraints among variables. Consider planning for a robot tasked with organizing several blocks. The robot must find a sequence of *pick*, *place*, and *move* actions involving continuous robot configurations, robot trajectories, block poses, and block grasps. These variables must satisfy kinematic, collision, and motion constraints which affect the feasibility of the actions. Each constraint typically requires a special purpose procedure to efficiently evaluate it or produce satisfying values for it such as an inverse kinematic solver, collision checker, or motion planner.

We propose an approach, called STRIPSTREAM, which can model such a domain by providing a generic interface for blackbox procedures to be incorporated in an action language. The implementation of the procedures is abstracted away using *streams*: finite or infinite sequences of objects such as poses, configurations, and trajectories. We introduce the following two additional stream capabilities to effectively model domains with complex predicates that are only true for small sets of their argument values:

- **conditional streams**: a stream of objects may be defined as a function of other objects; for example, a stream of possible positions of one object given the position of another object that it must be on top of or a stream of possible settings of parameters of a factory machine given desired properties of its output.
- **certified streams**: streams of objects may be declared not only to be of a specific type, but also to satisfy an arbitrary conjunction of predicates; for example, one might define a certified conditional stream that generates positions for an object that satisfy requirements that the object be on a surface, that a robot be able to reach the object at that position, and that the robot be able to see the object while reaching.

The approach is entirely domain-independent, and reduces to STRIPS in the case of finite domains. The only additional requirement is the specification of a set of streams that can generate objects satisfying the static predicates in the domain. It is accompanied by two algorithms, a simple and a focused version, which operate by constructing and solving a sequence of STRIPS planning problems. This strategy takes advantage of the highly optimized search strategies and heuristics that exist for STRIPS planning, while expanding the domain of applicability of those techniques. In this extended abstract, we present the STRIPSTREAM representation and implement a robotics domain. See the full version of this paper for the presentation of the algorithms and additional domains [1].

II. Representation

In this section we describe the representational components of a planning domain and problem, which include static and fluent predicates, operators, and streams. *Objects* serve as arguments to predicates and as parameters to operators; they are generated by streams.

A *static predicate* is a predicate which, for any tuple of objects, has a constant truth value throughout a problem instance. Static predicates generally serve to represent constraints on the parameters of an operator. We restrict static predicates to only ever be mentioned positively because, in the general infinite case, it is not possible to verify that a predicate does not hold.

An *operator* schema is specified by a tuple of formal parameters $(X_1, \ldots, X_n)$ and conjunctions of static positive preconditions **stat**, fluent literal preconditions **pre**, and fluent literal effects **eff** and has the same semantics as in STRIPS. An operator instance is a ground instantiation of an operator schema with objects substituted in for the formal parameters. When necessary, we augment the set of operator schemas with a set of axioms that naively use the same schema form as operators. We assume the set of axioms can be compiled into a set of derived predicates as used in PDDL.

A *generator* $g = \langle \bar{y}^1, \bar{y}^2, ... \rangle$ is a finite or infinite sequence of object tuples $\bar{y} = (y_1, ..., y_n)$. The procedure NEXT$(g)$ returns the next element in generator g and returns the special object **None** to indicate that the stream has been exhausted and contains no more objects. A *conditional generator* $f(\bar{x})$ is a function from a tuple of input objects $\bar{x} = (x_1, ..., x_n)$ to a generator $g_{\bar{x}}$ which generates tuples from a domain not necessarily the same as the domain of $\bar{x}$.

An *stream* schema, $\sigma(\bar{Y} \mid \bar{X})$, is specified by a tuple of input parameters $\bar{X} = (X_1, ..., X_m)$, a tuple of output parameters $\bar{Y} = (Y_1, ..., Y_n)$, a conditional generator **gen** $= f(\bar{X})$ defined on $\bar{X}$, a conjunction of input static atoms **inp** defined on $\bar{X}$, and a conjunction of output static atoms **out** defined on $\bar{X}$ and $\bar{Y}$. The conditional generator f is a function, implemented in the host programming language, that returns a generator object such that, for all $\bar{x}$ satisfying the conditions **inp**, $\forall \bar{y} \in f(\bar{x}), (\bar{x}, \bar{y})$ satisfy the conditions **out**. A stream instance is a ground instantiation of a stream schema with objects substituted in for input parameters $(X_1, \dots, X_n)$; it is *conditioned* on those object values and, if the **inp** conditions are satisfied, then it will generate a stream of tuples of objects each of which satisfies the certification conditions **out**.

The notion of a conditional stream is quite general; there are two specific cases that are worth understanding in detail. An *unconditional stream* $\sigma(\bar{Y} \mid ())$ is a stream with no inputs whose associated function f returns a single generator, which might be used to generate objects of a given type, for example, independent of whatever other objects are specified in a domain. A *test stream* $\sigma(() \mid \bar{X})$ is a degenerate, but still useful, type of stream with no outputs. In this case, $f(X_1, ..., X_m)$ contains either the single element (), indicating that the **inp** conditions hold of $\bar{X}$, or contains no elements at all, indicating that the **inp** conditions do not hold of $\bar{X}$. It can be interpreted as an implicit Boolean test.

A *planning domain* $\mathcal{D} = (\mathcal{P}_s, \mathcal{P}_f, \mathcal{C}_0, \mathcal{A}, \mathcal{X}, \Sigma)$ is specified by finite sets of static predicates $\mathcal{P}_s$, fluent predicates $\mathcal{P}_f$, initial constant objects $\mathcal{C}_0$, operator schemas $\mathcal{A}$, axiom schemas $\mathcal{X}$, and stream schemas Σ. Note that the initial objects (as well as objects generated by the streams) may in general not be simple symbols, but can be numeric values or even structures such as matrices or objects in an underlying programming language. They must provide a unique ID, such as a hash value, for use in the STRIPS planning phase.

A STRIPSTREAM *problem* $\Pi = (\mathcal{D}, O_0, s_0, s_*)$ is specified by a planning domain $\mathcal{D}$, a finite set of initial objects O_0, an initial state composed of a finite set of static or fluent atoms s_0, and a goal set defined to be the set of states satisfying fluent literals s_*. We make a version of the closed world assumption on the initial state s_0, assuming that all true fluents are contained in it. This initial state will not be complete: in general, it will be impossible to assert all true static atoms when the universe is infinite.

III. Example robotic domain

Although the specification language is domain independent, our primary motivating examples for the application of STRIPSTREAM are pick-and-place problems. The objects in this domain include a finite set of blocks, 6-dimensional block poses and grasp transforms, 11-dimensional robot configurations, and trajectories specified by sequences of configurations. The static predicates in this domain include simple static types ($IsBlock$, $IsPose$, $IsGrasp$, $IsConf$, $IsTraj$) and typical fluents ($HandEmpty$, $Holding$, $AtPose$, $AtConfig$). In addition, atoms of the form $IsKin(P, G, Q, T)$ describe a static relationship between a pose P, grasp G, robot configuration Q, and trajectory T. Finally, fluents of the form $Safe(b', B, G, P, T)$ are true in the circumstance that: if object B were held at using grasp G while executing trajectory T, it would not collide with object b' at its current pose. These predicate definitions enable the following MOVE and PICK operator schema definitions. The PLACE operator (omited) is similar to the PICK operator.

MOVE(Q_1, Q_2):
  stat = $\{IsConf(Q_1), IsConf(Q_2)\}$
  pre = $\{AtConf(Q_1)\}$
  eff = $\{AtConf(Q_2), \neg AtConf(Q_1)\}$

PICK(B, P, G, Q, T):
  stat = $\{IsBlock(B), ..., IsTraj(T), IsKin(P, G, Q, T)\}$
  pre = $\{AtPose(B, P), HandEmpty(), AtConfig(Q)\} \cup \{Safe(b', B, G, T) \mid b' \in \mathcal{B}\}$
  eff = $\{Holding(B, G), \neg AtPose(B, P), \neg HandEmpty()\}$

We use the following axioms to evaluate the $Safe$ predicate. We need two slightly different definitions to handle the cases where the block B_1 is placed at a pose, and where it is in the robot's hand (denoted by **or**). The $Safe$ axioms mention each block independently which allows us to compactly perform collision checking. Without using axioms, PLACE would require a parameter for the pose of each block in $\mathcal{B}$, resulting in an prohibitively large grounded problem.

SAFEAXIOM(B_1, P_1, B_2, G, T):
  stat = $\{IsBlock(B_1), ..., IsTraj(T), IsCFree(B_1, P_1, B_2, G, T)\}$
  pre = $\{AtPose(B_1, P_1)\}$ **or** $\{Holding(B_1)\}$
  eff =$\{Safe(B_1, B_2, G, T)\}$

Next, we provide stream definitions. The simplest stream is an unconditional generator of poses. It uses SAMPLE-POSE to randomly sample poses. These poses satisfy $IsPose$.

POSE$(P \mid ())$:
  gen = **lambda**() : SAMPLE-POSE()
  inp = $\emptyset$
  out = $\{IsPose(P)\}$

The conditional stream CFREE is a test, calling the underlying function COLLIDE(B_1, P_1, B_2, G, T). The stream is empty if block B_1 at pose P_1 collides with block B_2 held using grasp G while the robot executes trajectory T. It contains the single element ( ) if it does not collide. It is used to certify statically satisfies the $IsCFree$ predicate.

CFREE$(() \mid B_1, P_1, B_2, G, T)$:
  gen = **lambda**(B_1, P_1, B_2, G, T) : $\langle()$ **if not** COLLIDE$(B_1, P_1, B_2, G, T)\rangle$
  inp = $\{IsBlock(B_1), ..., IsTraj(T)\}$
  out = $\{IsCFree(B_1, P_1, B_2, G, T)\}$

Finally, KIN specifies a conditional stream, which takes a pose P and a grasp G as inputs and generates a stream of configurations and trajectories. It must first produce a grasp configuration Q that reaches manipulator transform PG^{-1} using INVERSE-KIN. Additionally, it calls a motion planner MOTIONS to generate legal trajectory values T from a constant rest configuration q_{rest} to the grasping configuration Q that do not collide with the fixed environment.

KIN$(Q, T \mid P, G)$:
  gen = **lambda**(P) : $\langle(Q, T) \mid Q \sim$ INVERSE-KIN(PG^{-1}), $T \sim$ MOTIONS$(q_{rest}, Q)\rangle$
  inp = $\{IsPose(P), IsGrasp(G)\}$
  out = $\{IsKin(P, G, Q, T), IsConf(Q), IsTraj(T)\}$

References

[1] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. STRIPS planning in infinite domains. *arXiv preprint arXiv:1701.00287*, 2017.