STRIPS Planning in Infinite Domains

Caelan Reed Garrett[1], Tomás Lozano-Pérez[1], and Leslie Pack Kaelbling[1]

***Abstract*— Many robotic planning applications involve continuous actions with highly non-linear constraints, which cannot be modeled using modern planners that construct a propositional representation. We introduce STRIPSTREAM: an extension of the STRIPS language which can model these domains by supporting the specification of blackbox generators to handle complex constraints. The outputs of these generators interact with actions through possibly infinite streams of objects and static predicates. We provide two algorithms which both reduce STRIPSTREAM problems to a sequence of finite-domain planning problems. The representation and algorithms are entirely domain independent. We demonstrate our framework on simple illustrative domains, and then on a high-dimensional, continuous robotic task and motion planning domain.**

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 in a room. 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 highly non-linear 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.

[1]MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139, USA {caelan,tlp,lpk}@csail.mit.edu

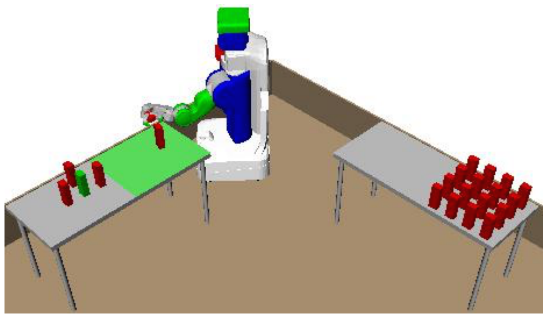
Fig. 1: Problem 2-16.

Through streams, STRIPSTREAM can compactly model a large class of continuous, countably infinite, and large finite domains. By conditioning on partial argument values and using sampling, it can even effectively model domains where the set of valid action argument values is lower dimensional than the possible argument space. For example, in our robotics domain, the set of inverse kinematics solutions for a particular pose and grasp is much lower dimensional than the full set of robot configurations. However, using a conditional stream, we can specify an inverse kinematics solver which directly samples from this set given a pose and grasp.

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. Additionally, the focused version can efficiently solve problems where using streams is computationally expensive by carefully choosing to only call potentially useful streams.

II. Related work

Semantic attachments [1], predicates computed by an external program, also provide a way of integrating blackbox procedures and PDDL planners. Because semantic attachments take a state as input, they can only be used in forward state-space search. Furthermore, they are ignored in heuristics. This results in poor planner performance, particularly when the attachments are expensive to evaluate such as in robotics domains. Finally, because semantic attachments are restricted to be functions, they are unable to model domains with infinitely many possible successor states.

Many approaches to robotics planning problems, including motion planning and task-and-motion planning, have developed strategies for handling continuous spaces that go beyond *a priori* discretization. Several approaches, for example [2], [3], [4], [5], [6], [7], have been suggested for integrating sampling-based robot motion planning methods with symbolic planning methods. Of these approaches, those able to plan in realistic robot domains have typically been quite special purpose; the more general purpose approaches have typically been less capable.

III. 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* $g = \langle \bar{o}^1, \bar{o}^2, ... \rangle$ is a finite or infinite sequence of object tuples $\bar{o} = (o_1, ..., o_n)$. 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 $\bar{x} = x_1, ..., x_n$ 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 $(X_1, \ldots, X_n)$; 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.

Let $\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ be the universe of all objects and the set of true initial atoms that can be generated from a finite set Σ of stream schemas, a finite set $C_0 \cup O_0$ of initial objects, and initial state s_0. We give all proofs in the the extended version of this paper [8].

Theorem 1. *$\mathcal{O}_\Pi$ and $\mathcal{S}_\Pi$ are recursively enumerable (RE) [8].*

A *solution* to a STRIPSTREAM problem Π is a finite sequence of operator instances π with object parameters contained within $\mathcal{O}_\Pi$ that is applicable from $\mathcal{S}_\Pi$ and results in a state that satisfies s_*. STRIPSTREAM is undecidable but semi-decidable, so we restrict our attention to feasible instances.

Theorem 2. *The existence of a solution for a* STRIPSTREAM *problem Π is undecidable [8].*

Theorem 3. *The existence of a solution for a* STRIPSTREAM *problem Π is semi-decidable [8].*

IV. Planning algorithms

We present two algorithms for solving STRIPSTREAM problems: the *incremental* planner takes advantage of certified conditional streams in the problem specification to generate the necessary objects for solving the problem;

the *focused* planner adds the ability to focus the object-generation process based on the requirements of the plan being constructed. Both algorithms are sound and complete: if a solution exists they will find it in finite time.

Both planners operate iteratively, alternating between adding elements and atoms to a current set of objects and initial atoms and constructing and solving STRIPS planning problem instances. A STRIPS problem $(\mathcal{P}, \mathcal{A}, O, s_{init}, s_*)$ is specified by a set of predicates, a set of operator schemas, a set of constant symbols, an initial set of atoms, and a set of goal literals. Let S-PLAN$(\mathcal{P}, \mathcal{A}, O, s_{init}, s_*)$ be any sound and complete planner for the STRIPS subset of PDDL. We implement S-PLAN using FastDownward [9].

A. Incremental planner

The incremental planner maintains a queue of stream instances Q and incrementally constructs set $\mathcal{O}$ of objects and set $\mathcal{S}$ of fluents and static atoms that are true in the initial state. The *done* set D contains all streams that have been constructed and exhausted. In each iteration of the main loop, a STRIPS planning instance is constructed from the current sets $\mathcal{O}$ and $\mathcal{S}$, with the same predicates, operator and axiom schemas, and goal. If a plan is obtained, it is returned. If not, then $K \geq 1$ attempts to add new objects are made where K is a meta-parameter. In each one, a stream $\sigma(\bar{Y} \mid \bar{x})$ is popped from Q and a new tuple of objects $\bar{y}$ is extracted from it. If the stream is exhausted, it is stored in D. Otherwise, the objects in $\bar{y}$ are added to $\mathcal{O}$, the output fluents from σ applied to $(\bar{x}, \bar{y})$ are added to $\mathcal{S}$, and a new set of streams Σ_n is constructed. For all stream schemas σ and possible tuples of the appropriate size $\bar{x}'$, if the input conditions $\sigma'.\textbf{inp}(\bar{x}')$ are in $\mathcal{S}$, then the instantiated stream $\sigma'(\bar{Y}' \mid \bar{x}')$ is added to Q if it has not been added previously. We also return the stream $\sigma(\bar{Y} \mid \bar{x})$ to Q so we may revisit it in the future. The psuedo-code is shown below.

```
INCREMENTAL$(((\mathcal{P}_s, \mathcal{P}_f, \mathcal{C}_0, \mathcal{A}, \mathcal{X}, \Sigma), O_0, s_0, s_*), \text{S-PLAN}, K)$ :
    $\mathcal{O} = \mathcal{C}_0 \cup O_0;\ \mathcal{S} = s_0;\ D = \emptyset$
    $Q = \text{QUEUE}(\{\sigma(\bar{Y} \mid \bar{x}) \mid \sigma \in \Sigma, \bar{x} \in \mathcal{O}^{m_\sigma}, \sigma.\textbf{inp}(\bar{x}) \subseteq \mathcal{S}\})$
    while True:
        $\pi = \text{S-PLAN}(\mathcal{P}_s \cup \mathcal{P}_f, \mathcal{A} \cup \mathcal{X}, \mathcal{O}, \mathcal{S}, s_*)$
        if $\pi \neq$ None: return $\pi$
        for $k \in [1..K]$:
            if EMPTY$(Q)$:
                if $k = 1$: return None else break
            $\sigma(\bar{Y} \mid \bar{x}) = \text{POP}(Q)$
            $\bar{y} = \text{NEXT}(\sigma.f(\bar{x}))$
            if $\bar{y} \neq$ None:
                $\mathcal{O} = \mathcal{O} \cup \bar{y};\ \mathcal{S} = \mathcal{S} \cup \sigma.\textbf{out}((\bar{x}, \bar{y}))$
                $\Sigma_n = \{\sigma'(\bar{Y}' \mid \bar{x}') \mid \sigma' \in \Sigma, \bar{x}' \in \mathcal{O}^{m_{\sigma'}},$
                                                  $\sigma'.\textbf{inp}(\bar{x}') \subseteq \mathcal{S}\}$
                PUSH$(Q, \Sigma_n \setminus (Q \cup D))$; PUSH$(Q, \{\sigma(\bar{Y} \mid \bar{x})\})$
            else
                $D = D \cup \{\sigma(\bar{Y} \mid \bar{x})\}$
```

In practice, many S-PLAN calls report infeasibility immediately because they have infinite admissible heuristic values.

Theorem 4. *The incremental algorithm is complete [8].*

B. Focused Planner

The focused planner is particularly aimed at domains for which it is expensive to draw an object from a stream; this occurs when the stream elements are certified to satisfy geometric properties such as being collision-free or having appropriate inverse kinematics relationships, for example. To focus the generation of objects on the most relevant parts of the space, we allow the planner to use "dummy" abstract objects as long as it plans to generate concrete values for them. These concrete values will be generated in the next iteration and will, hopefully, contribute to finding a solution with all ground objects.

As before, we transform the STRIPSTREAM problem into a sequence of PDDL problems, but this time we augment the planning domain with abstract objects, two new fluents, and a new set of operator schemas. Let $\{\gamma_1, ..., \gamma_\theta\}$ be a set of *abstract objects* which are not assumed to satisfy any static predicates in the initial state. We introduce the fluent predicate $Concrete$, which is initially false for any object γ_i but true for all actual ground objects; so for all $o \in \mathcal{O}$, we add $Concrete(o)$ to s_{init}. The planner can "cause" an abstract object γ_i to satisfy $Concrete(\gamma_i)$ by generating it using a special *stream operator*, as described below. We define procedure TFORM-OPS that transforms each operator schema $a(x_1, ..., x_n) \in \mathcal{A}$ by adding preconditions $Concrete(x_i)$ for $i = 1, ..., n$ to ensure that the parameters for a are grounded before its application during the search.

To manage the balance in which streams are called, for each stream schema σ, we introduce a new predicate $Blocked_\sigma$; when applied to arguments $(X_1, \ldots, X_n)$, it will temporarily prevent the use of stream $\sigma(Y_1, ...Y_m \mid X_1, ..., X_n)$. Additionally, we add any new objects and static atoms first to sets $\mathcal{O}_t$ and $\mathcal{S}_t$ temporarily before adding them to $\mathcal{O}$ and $\mathcal{S}$ to ensure any necessary existing streams are called. Alternatively, we can immediately add directly to $\mathcal{O}$ and $\mathcal{S}$ a finite number of times before first adding to $\mathcal{O}_t$ and $\mathcal{S}_t$ and still preserve completeness. Let the procedure TFORM-STREAMS convert each stream schema into an operator schema σ of the following form.

```
StreamOperator$_\sigma(X_1, ..., X_m, Y_1, ..., Y_n)$:
    pre = $\sigma.\textbf{inp} \cup\ \{Concrete(X_i) \mid i = 1, ..., m\} \cup$
                        $\{\neg Blocked_\sigma(X_1, ..., X_m)\}$
    eff = $\sigma.\textbf{out} \cup \{Concrete(Y_i) \mid i = 1, ..., n\}$
```

It allows S-PLAN to explicitly plan to generate a tuple of concrete objects from stream $\sigma(Y_1, ...Y_m \mid x_1, ..., x_n)$ as long as the x_i have been made concrete and the stream instance is not blocked.

The procedure FOCUSED, shown below, implements the focused approach to planning. It takes the same inputs as the incremental version, but with the maximum number of abstract objects $\theta \geq 1$ specified as a meta-parameter, rather than K. It also maintains a set $\mathcal{O}$ of concrete objects and a set $\mathcal{S}$ of fluent and static atoms true in the initial state. In each iteration of the main loop, a STRIPS planning instance is constructed: the initial state is augmented with the set of static atoms indicating which streams are blocked and fluents asserting that the objects in $\mathcal{O}$ are concrete; the set of operator schemas is transformed as described above and augmented with the stream operator schemas, and the set of objects is augmented with the abstract objects. If a plan is obtained and it contains only operator instances, then it will have only concrete objects, and it can be returned directly. If the plan contains abstract objects, it also contains stream operators, and ADD-OBJECTS is called to generate an appropriate set of new objects. If no plan is obtained, and if no streams are currently blocked as well as no new objects or initial atoms have been produced since the last reset, then the problem is proved to be infeasible. Otherwise, the problem is reset by unblocking all streams and adding $\mathcal{O}_t$ and $\mathcal{S}_t$ to $\mathcal{O}$ and $\mathcal{S}$, in order to allow a new plan with abstract objects to be generated.

```
FOCUSED$(((\mathcal{P}_s, \mathcal{P}_f, \mathcal{C}_0, \mathcal{A}, \mathcal{X}, \Sigma), O_0, s_0, s_*), \text{S-PLAN}, \theta)$ :
    $\mathcal{O} = \mathcal{C}_0 \cup O_0$; $\mathcal{S} = s_0$; $\mathcal{O}_t = \mathcal{S}_t = \beta_t = \beta_p = \emptyset$
    $\bar{\mathcal{A}} = \text{TFORM-OPS}(\mathcal{A})$; $\bar{\Sigma} = \text{TFORM-STREAMS}(\Sigma)$
    while True:
        $\pi = \text{S-PLAN}(\mathcal{P}_s \cup \mathcal{P}_f, \bar{\mathcal{A}} \cup \mathcal{X} \cup \bar{\Sigma}, \mathcal{O} \cup \{\gamma_1, ..., \gamma_\theta\},$
                $\mathcal{S} \cup \beta_t \cup \beta_p \cup \{Concrete(o \in \mathcal{O})\}, s_*)$
        if $\pi \neq$ None:
            if $\forall a \in \pi$. SCHEMA$(a) \in \bar{\mathcal{A}}$: return $\pi$
            $\mathcal{O}_t, \mathcal{S}_t, \beta_t, \beta_p = \text{ADD-OBJECTS}(\pi, \mathcal{O}_t, \mathcal{S}_t, \beta_t, \beta_p, \bar{\Sigma})$
        else
            if $\mathcal{O}_t = \mathcal{S}_t = \beta_t = \emptyset$: return None // Infeasible
            $\mathcal{O} = \mathcal{O} \cup \mathcal{O}_t$; $\mathcal{S} = \mathcal{S} \cup \mathcal{S}_t$
            $\mathcal{O}_t = \mathcal{S}_t = \beta_t = \emptyset$ // Enable all objects & streams
```

Given a plan π that contains abstract objects, we process it from beginning to end, to generate a collection of new objects with appropriate conditional relationships. Procedure ADD-OBJECTS initializes an empty binding environment and then loops through the instances a of stream operators in π. For each stream operator instance, we substitute concrete objects in for abstract objects, in the input parameters, dictated by the bindings bd, and then draw a new tuple of objects from that conditional stream. If there is no such tuple of objects, the stream is exhausted and it is permanently removed from future consideration by adding the fluent $\textit{Blocked}_\sigma(\bar{o}_x)$ to the set β_p. Otherwise, the new objects are added to $\mathcal{O}_t$ and appropriate new static atoms to $\mathcal{S}_t$. This stream is temporarily blocked by adding fluent $\textit{Blocked}_\sigma(\bar{o}_x)$ to the set β_t, and the bindings for abstract objects are recorded.

```
ADD-OBJECTS$(\pi, \mathcal{O}_t, \mathcal{S}_t, \beta_t, \beta_p, \bar{\Sigma})$ :
    $bd = \{\ \}$ // Empty dictionary
    for $\sigma(\bar{y} \mid \bar{x}) \in \{a \mid a \in \pi$ and $a$ is instance of element of $\bar{\Sigma}\}$:
        $\bar{o}_x = apply\text{-}bindings(bd, (x_1, ..., x_m))$
        $\bar{o}_y = \text{NEXT}(\sigma.f(\bar{o}_x))$
        if $\bar{o}_y =$ None:
            $\beta_p = \beta_p \cup \{\textit{Blocked}_\sigma(\bar{o}_x)\}$ // Permanent
        else
            $\mathcal{O}_t = \mathcal{O}_t \cup \bar{o}_y$; $\mathcal{S}_t = \mathcal{S}_t \cup \sigma.\textbf{out}((\bar{o}_x, \bar{o}_y))$
            $\beta_t = \beta_t \cup \{\textit{Blocked}_\sigma(\bar{o}_x)\}$ // Temporary
            for $i \in [1..m]$ : $bd[y_i] = o_{y,i}$
    return $\mathcal{O}_t, \mathcal{S}_t, \beta_t, \beta_p$
```

The focused algorithm is similar to the lazy shortest path algorithm for motion planning in that it determines which streams to call, or analogously which edges to evaluate, by repeatedly solving optimistic problems [10]. Stream operators can be given meta-costs that reflect the time overhead to draw elements from the stream and the likelihood the stream produces the desired values. For example, stream operators that use already concrete outputs can be given large meta-costs because they will only certify a desired predicate in the typically unlikely event that their generator returns objects matching the desired outputs. A cost-sensitive planner will avoid returning plans that require drawing elements from expensive or unnecessary streams. We can combine the behaviors of incremental and focused algorithms to eagerly call inexpensive streams and lazily call expensive streams. This can be seen as automatically applying some stream operators before calling S-PLAN.

Theorem 5. *The focused algorithm is complete [8].*

V. Example discrete domain

Although the specification language is domain independent, our primary motivating examples for the application of STRIPSTREAM are pick-and-place problems in infinite domains. We start by specifying an infinite discrete pick-and-place domain as shown in figure 2. We purposefully describe the domain in a way that will generalize well to continuous and high-dimensional versions of fundamentally the same problem. The objects in this domain include a finite set of blocks , an infinite set of poses indexed by the positive integers, and an infinite set of robot configurations also indexed by the positive integers. The static predicates in this domain include simple static types ($\textit{IsConf}$, $\textit{IsPose}$, $\textit{IsBlock}$) and typical fluents ($\textit{HandEmpty}$, $\textit{Holding}$, $\textit{AtPose}$, $\textit{AtConfig}$). In addition, atoms of the form $\textit{IsKin}(P, Q)$ describe a static relationship between an object pose P and a robot configuration Q: in this simple domain, the atom is true if and only if $P = Q$. Finally, fluents of the form $\textit{Safe}(b', B, P)$ are true in the circumstance that: if object B were placed at pose P, it would not collide with object b' at its current pose. Because the set of blocks $\mathcal{B}$ is known statically in advance, we explicitly include all the $\textit{Safe}$ conditions. These predicate definitions enable the following operator schemas definitions:

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

PICK$(B, P, Q)$:
    stat = $\{\textit{IsBlock}(B), \textit{IsPose}(P), \textit{IsConf}(Q), \textit{IsKin}(P, Q)\}$
    pre = $\{\textit{AtPose}(B, P), \textit{HandEmpty}(), \textit{AtConfig}(Q)\}$
    eff = $\{\textit{Holding}(B), \neg\textit{AtPose}(B, P), \neg\textit{HandEmpty}()\}$

PLACE$(B, P, Q)$:
    stat = $\{\textit{IsBlock}(B), \textit{IsPose}(P), \textit{IsConf}(Q), \textit{IsKin}(P, Q)\}$
    pre = $\{\textit{Holding}(B), \textit{AtConfig}(Q)\} \cup \{\textit{Safe}(b' \in \mathcal{B}, B, P)\}$
    eff = $\{\textit{AtPose}(B, P), \textit{HandEmpty}(), \neg\textit{Holding}(B)\}$
```

We use the following axioms to evaluate the $\textit{Safe}$ 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 $\textit{Safe}$ 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($B_1, P_1, B_2, P_2$):
    stat = $\{IsBlock(B_1), IsPose(P_1), IsBlock(B_2),$
        $IsPose(P_2), IsCollisionFree(B_1, P_1, B_2, P_2)\}$
    pre = $\{AtPose(B_1, P_1)\}$ or $\{Holding(B_1)\}$
    eff =$\{Safe(B_1, B_2, P_2)\}$
```

Next, we provide stream definitions. The simplest stream is an unconditional generator of poses, which are represented as objects POSE(i) and satisfy the static predicate *IsPose*.

```
POSE($P \mid ()$):
    gen = lambda() : $\langle$(POSE($i$)) for $i = 0, 1, 2...\rangle$
    inp = $\emptyset$
    out = $\{IsPose(P)\}$
```

The conditional stream CFREE is a test, calling the underlying function COLLIDE(B_1, P_1, B_2, P_2); the stream is empty if block B_1 at pose P_1 collides with block B_2 at pose P_2, and contains the single element ( ) if it does not collide. It is used to certify that the tuple (B_1, P_1, B_2, P_2) statically satisfies the *IsCollisionFree* predicate.

```
CFREE($() \mid B_1, P_1, B_2, P_2$):
    gen = lambda($B_1, P_1, B_2, P_2$) :
                $\langle$() if not COLLIDE($B_1, P_1, B_2, P_2$)$\rangle$
    inp = $\{IsBlock(B_1), IsPose(P_1), IsBlock(B_2), IsPose(P_2)\}$
    out = $\{IsCollisionFree(B_1, P_1, B_2, P_2)\}$
```

Finally, KIN specifies a conditional stream, which takes a pose P as input and generates a stream of configurations (in this very simple case, containing a single element) certified to satisfy the *IsKin* relation. It relies on an underlying function INVERSE-KIN(p) to produce an appropriate robot configuration given a block pose. We made a modeling choice by specifying KIN in this form rather than KIN($P, Q \mid ()$) or KIN($() \mid P, Q$). The conditional formulation KIN($Q \mid P$) is advantageous over the others because it produces a paired inverse kinematics configuration quickly and without substantially expanding the size of the problem. See the extended version of this paper for additional discussion [8].

```
KIN($Q \mid P$):
    gen = lambda($P$) : $\langle$(INVERSE-KIN($P$))$\rangle$
    inp = $\{IsPose(P)\}$
    out = $\{IsConf(Q), IsIK(P, Q)\}$
```

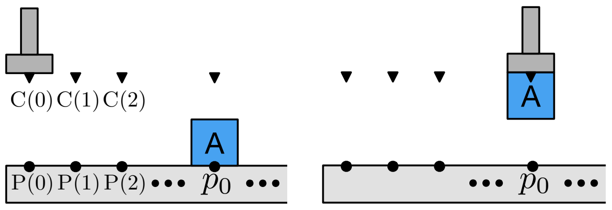

Fig. 2: The initial state and goal state in an infinite, discrete pick-and-place problem requiring picking block A.

VI. CONTINUOUS DOMAINS

The STRIPSTREAM approach can be applied directly in continuous domains such as the problem in figure 3. In this case, the streams will have to generate samples from sets of continuous dimensions, and the way that samples are generated may have a significant impact on the efficiency and completeness of the approach with respect to the domain problem. The STRIPSTREAM planing algorithms are complete with respect to the streams of enumerated values they are given, but if these value streams are not complete with respect to the underlying problem domain, then the resulting combined system may not be complete with respect to the original problem. Samplers that produce a *dense* sequence [11] are good candidates for stream generation.

With some minor modifications, we can extend our discrete pick-and-place domain to a bounded interval $[0, L]$ of the real line. Poses and configurations are now continuous objects $p, q \in [0, L]$ from an uncountably infinite domain. The stream POSE now has a generator that samples $[0, L]$ uniformly at random. While in the discrete case the choice of streams just affected the size of the problem, in the continuous case, the choice of streams can affect the feasibility of the problem. The KIN($P, Q \mid ()$) stream has zero probability of producing any desired pose or configuration. The KIN($() \mid P, Q$) test stream requires a pose and configuration as inputs. When sampled independently at random, there is zero probability that they form a valid kinematic pair.

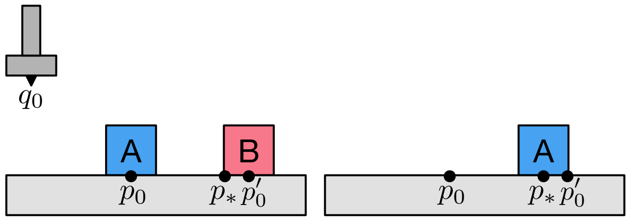

Fig. 3: Initial and goal state for the continuous pick-and-place problem requiring moving block A to p_*, with a single obstructing block B.

VII. REALISTIC ROBOT DOMAIN

Finally, we extend our continuous pick-and-place to the high-dimensional setting of a robot operating in household-like environments. Poses of physical blocks are 6-dimensional and robot configurations are 11-dimensional. We introduce two new object types: grasps and trajectories. Each block has a set of 6D relative grasp transforms at which it can

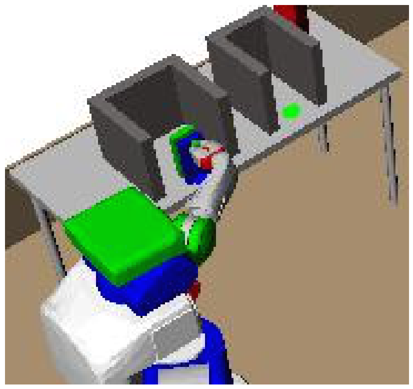

Fig. 4: Problem 1.

Π	increm. $K = 1$			increm. $K = 100$			focused		
	%	t	c	%	t	c	%	t	c
1	88	2	268	68	5	751	84	11	129
2-0	100	23	1757	100	9	2270	100	2	180
2-8	0	-	-	100	55	17217	100	7	352
2-16	0	-	-	100	112	36580	100	19	506

TABLE I: The success percentage (%), runtime (t), search iterations (i), and number of stream calls (c) for the high-dimensional task and motion planning experiments.

be grasped by the robot. Trajectories are finite sequences of configuration waypoints which must be included in collision checking. The extended PICK operator, CFREE test and KIN stream templates are:

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

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

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 = $\{\mathit{IsPose}(P), \mathit{IsGrasp}(G)\}$
    out = $\{\mathit{IsKin}(P, G, Q, T), \mathit{IsConf}(Q), \mathit{IsTraj}(T)\}$
```

PICK adds grasp G and trajectory T as parameters and includes $\mathit{Safe}(b', B, G, T)$ preconditions to verify that T while holding B at grasp G is safe with respect to each other block b'. $\mathit{Safe}(b', B, G, T)$ is updated using SAFEAXIOM which has a $\mathit{IsCollisionFree}(B_1, P_1, B_2, G, T)$ static precondition. Here, a collision check for block B_1 at pose P_1 is performed for each configuration in T. Instead of simple blocks, physical objects in this domain are general unions of convex polygons. Although checking collisions here is more complication than in 1D, it can be treated in the same way, as an external function.

The KIN streams must first produce a grasp configuration Q that reaches manipulator transform PQ^{-1} using INVERSE-KIN. Additionally, they include 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. In this domain, the procedures for collision checking and finding kinematic solutions are significantly more involved and computationally expensive than in the previous domains, but their underlying function is the same.

VIII. EXPERIMENTS

We applied the incremental and focused algorithms on four challenging pick-and-place problems to demonstrate that a general-purpose representation and algorithms can be used to achieve good performance in difficult problems. For both algorithms, test streams are always evaluated as soon as they are instantiated. We experimented on two domains shown in figures 4 and 1, which are similar to problems introduced by [5]. The first domain, in which problem 1 is defined, has goal conditions that the green object be in the right bin and the blue object remain at its initial pose. This requires the robot to not only move and replace the blue block but also to place the green object in order to find a new grasp to insert it into the bin. The second domain, in which problems 2-0, 2-8, and 2-16 are defined, requires moving an object out of the way and placing the green object in the green region. For problem 2-n where $n \in \{0, 8, 16\}$ there are n other blocks on a separate table that serve as distractors. The streams were implemented using the OpenRAVE robotics framework [12]. A Python implementation of STRIPSTREAM can be found here: `https://github.com/caelan/stripstream`.

The results compare the incremental algorithm where $K = 1$ and $K = 100$ with the focused algorithm. Table I shows the results of 25 trials, each with a timeout of 120 seconds. The incremental algorithms result in significantly more stream calls than the focused algorithm. These calls can significantly increase the total runtime because each inverse kinematic and collision primitive itself is expensive. Additionally, the incremental algorithms are significantly affected by the increased number of distractors, making them unsuitable for complex real-world environments. The focused algorithm, however, is able to selectively choose which streams to call resulting in significantly better performance in these environments.

REFERENCES

[1] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, 2009, pp. 114–121.

[2] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[3] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[4] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[5] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Backward-forward search for manipulation planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015. [Online]. Available: http://lis.csail.mit.edu/pubs/garrett-iros15.pdf

[6] N. T. Dantam, Z. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and Systems (RSS)*, 2016.

[7] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *arXiv preprint arXiv:1608.01335*, 2016.

[8] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "STRIPS planning in infinite domains," *arXiv preprint arXiv:1701.00287*, 2017.

[9] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research (JAIR)*, vol. 26, pp. 191–246, 2006.

[10] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1. IEEE, 2000, pp. 521–528.

[11] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: msl.cs.uiuc.edu/planning

[12] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-08-34, 2008.