Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

Efficient Learning of Action Schemas and Web-Service Descriptions

Thomas J. Walsh and **Michael L. Littman**
Department of Computer Science, Rutgers University
110 Frelinghuysen Road, Piscataway, NJ 08854
{thomaswa, mlittman}@cs.rutgers.edu

Abstract

This work addresses the problem of efficiently learning action schemas using a bounded number of samples (interactions with the environment). We consider schemas in two languages— traditional STRIPS, and a new language STRIPS+WS that extends STRIPS to allow for the creation of new objects when an action is executed. This modification allows STRIPS+WS to model web services and can be used to describe web-service composition (planning) problems. We show that general STRIPS operators cannot be efficiently learned through raw experience, though restricting the size of action preconditions yields a positive result. We then show that efficient learning is possible without this restriction if an agent has access to a "teacher" that can provide solution traces on demand. We adapt this learning algorithm to efficiently learn web-service descriptions in STRIPS+WS.

Introduction

The term *action schemas* refers to a wide variety of techniques and languages for modeling sequential decision making problems. By describing action outcomes at a conceptual level, action schemas (as depicted in Table 1) provide generalization beyond propositional models. However, while machine-learning techniques for acquiring propositional action models have been widely studied in terms of theoretical efficiency (via the notion of *sample complexity*), the results on learning action-schema models have been largely empirical. The primary goal of this paper is to propose a provably efficient algorithm for learning descriptions in the simple action-schema language, STRIPS.

The imperative to develop such algorithms has recently gained traction in the real world with the proliferation of *web services*, and the need to link together services to complete complex tasks. Many researchers have attempted to automate this process with planning techniques (Hoffman, Bertoli, & Pistore 2007), but they usually assume a standardized language describing the services, both in terms of syntax and semantics. While popular protocols (e.g. REST, SOAP) realize the former assumption, semantic standardization is practically unachievable if we rely on the providers to label content, especially if the services come from different sources. Learning web-service behavior from examples is a pragmatic approach for constructing models for use with existing planners in the midst of this semantic anarchy.

Our work exploits the connection between web-service description learning and the classical problem of learning action schemas. Both tasks can be described with similar languages, though web-service descriptions need to capture "object creation". We consider learning in both of these settings with the aid of a teacher that returns a "trace" of grounded states and actions leading to the goal. Such teachers have long been employed in empirically evaluated action-schema learners (e.g. TRAIL (Benson 1996)). In this paper we prove they provide an exponential speedup. We also show this result can be ported to web-service learning by extending STRIPS with *functions* (STRIPS+WS). Our main contributions are providing positive and negative sample complexity results for learning action schemas and web-service descriptions.

Learning STRIPS Operators

We first consider learning action schemas in the STRIPS language (Fikes & Nilsson 1971). We show that learning STRIPS operators through raw-experience can require an exponential number of samples, but restricting the size of the precondition lists allows for sample-efficient learning. We then present an algorithm that interacts with a *teacher* to efficiently learn STRIPS schemas.

STRIPS Operators

The STRIPS language describes domains where world *states* are a conjunction of true *grounded fluents*[1] (e.g. *On*(**b**, **c**)) made up of a predicate from a set of size P, with parameters drawn from a set of known objects, $\mathcal{O}$. STRIPS actions are parameterized, of the form $a(\mathrm{X}_1,...,\mathrm{X}_m)$, where a is drawn from a set of size A. Each action's behavior is described by a schema, made up of three *operator lists*, the *Pre-conditions* (PRE), *Add-list* (ADD), and *Delete-list* (DEL). Each list is described by *fluents*, as in Table 1. ADD (DEL) describes what fluents become true (false) as a result of an action, while PRE governs whether the action will *succeed* or *fail*. We assume these latter outcomes are explicitly revealed as actions are taken. Objects in STRIPS cannot be created or

[1] We refer to predicates over variables (*On*(X, Y)) as "fluents" and their grounded form (*On*(**b**, **c**)) as "grounded fluents".

destroyed and fluents in the preconditions must be positive. Also, actions have a limited *scope*; all variables referenced in an action schema must appear in the action's parameter list. We assume that environments are deterministic, free of conditional effects, all states are fully observable (no hidden/delayed outcomes), and the arity of all actions and predicates are bounded by *constants* m and n, respectively. For details on these terms see Russell & Norvig (1995).

STRIPS Learning Problem

At the beginning of learning, an agent knows what actions are available and what predicates will be used to describe states. Learning proceeds in episodes, and at the beginning of each episode, an agent is presented with an initial state, s_0, and a goal description, $\mathcal{G}$. s_0 is a set of grounded fluents and $\mathcal{G}$ is a conjunction of predicates whose parameters come either from $\mathcal{O}$ or are existentially quantified (e.g. $\exists$ X *On*(**b**,X) $\wedge$ *Block*(X)). The agent is then free to take grounded actions and the environment responds with either *fail* or an updated state description. After reaching the goal *or* declaring that the goal is unreachable, the agent starts a new episode. The underlying schemas do not change between episodes, but s_0, $\mathcal{O}$, and $\mathcal{G}$ might.

Given a schema, well-studied planning techniques (Russell & Norvig 1995) can determine if the goal is reachable and produce a corresponding *plan* (a sequence of grounded actions, $a_0 ... a_k$). We assume throughout this work that our planner is complete and provides *correct* plans, that is $s_{i+1} = (s_i \cup a_i.\text{ADD} - a_i.\text{DEL})$; $s_i \models a_i.\text{PRE}$; and $s_{k+1} \models \mathcal{G}$. The goal of the *STRIPS learning problem* is to produce a set of STRIPS operators that when used by a complete planner, will induce a correct plan, or correctly state "no plan". We wish to achieve this guarantee with a limited *sample complexity*, a bound on the number of interactions the learner must have with the environment to construct the schema described above. In this work, we measure sample complexity in terms of *plan-prediction mistakes* (PPMs).

Definition 1. *A* plan-prediction mistake *occurs when an agent chooses a plan for a given episode that will not reach the goal, or asserts "no plan" when the goal is reachable.*

We say a learning algorithm is *efficient* in terms of sample complexity if it makes a polynomial (in P and A, with constants m and n) number of plan-prediction mistakes. The technique of limiting mistakes is a well known paradigm for bounding sample complexity in learning (Littlestone 1988) and sequential decision making (Kakade 2003). When a mistake is made, the model needs to be refined using experience. Later, we consider how different forms of experience affect the number of PPMs.

First, we discuss some complications of STRIPS learning. Regardless of the channel of experience, the basic operation that needs to be performed during learning is an *operator update* on action a's operator lists based on an observed state transition $\langle s, a, s' \rangle$. Unfortunately, a single experience may not be enough to pin down a fluent f's role in any of these lists as we see in the *operator update rules* below that outline how to update a.PRE (Rule 1) and the effect lists (Rules 2-5) when an action succeeds. Notice that several (not listed) cases for a.PRE are uninformative by themselves, though together they may be helpful.

After first trace

move(B, From, To):
PRE: *On*(B, From) , *Clear*(B) , *Clear*(From) , *Clear*(To) , *Block*(B) , *Block*(To) , *Table*(From)
ADD: *On*(B, To)
DEL: *On*(B, B) , *On*(From, B) , *On*(To, B) , *On*(B, From) , *On*(From, From) , *On*(From, To) , *On*(To, To) , *Clear*(To) , *Block*(From) , *Table*(B) , *Table*(To)

After second trace

move(B, From, To):
PRE: *On*(B, From) , *Clear*(B) , *Clear*(To) , *Block*(B) , *Block*(To)
ADD: *On*(B, To) , *Clear*(From)
DEL: *On*(B, B) , *On*(From, B) , *On*(To, B) , *On*(B, From) , *On*(From, From) , *On*(From, To) , *On*(To, To) , *Clear*(To) , *Table*(B) , *Table*(To)

Table 1: Learned STRIPS action schemas from the blocks world traces in the **Example** section. Variable names are inserted for readability.

1. **If** $f \notin s \wedge$ *Succeed* **Then** $f \notin a$.PRE.
2. **If** $f \notin s \wedge f \in s'$ **Then** $f \in a$.ADD, $f \notin a$.DEL.
3. **If** $f \in s \wedge f \notin s'$ **Then** $f \in a$.DEL, $f \notin a$.ADD.
4. **If** $f \notin s \wedge f \notin s'$ **Then** $f \notin a$.ADD.
5. **If** $f \in s \wedge f \in s'$ **Then** $f \notin a$.DEL.

Further complications arise with parameterized actions when the same object appears multiple times in an action's parameter list. For example, if the learner experiences the action a(**b**, **b**) causing *P*(**b**), it is not clear whether *P*(X) or *P*(Y) (or both) should be inserted into a(X, Y).ADD. We propose avoiding such ambiguity by learning different *action-versions*, that is, a separate operator is learned for each pattern of matching parameters. The number of possible action versions scales, albeit combinatorially, with m, which is constant and therefore doesn't affect the theoretical efficiency of learning (see Walsh & Littman (2008)).

Prior Work on STRIPS Learning

A number of prior works have focused on learning STRIPS-like action schemas. EXPO (Gil 1994) was bootstrapped by an incomplete STRIPS-like domain description with the rest being filled in through experience. The OBSERVER system (Wang 1995) also used a STRIPS-style language and was trained with a mixture of both raw experience and grounded expert traces. The TRAIL system (Benson 1996) used Inductive Logic Programming (ILP) to distill schemas from raw experience and a teacher. All of these early systems recognized and demonstrated the benefit of experience beyond simple interaction with the environment (bootstrapping, traces, teachers). However, none of these works provided a theoretical justification for this second channel of experience (as we do), nor did they bound the sample complexity of their learning algorithms. Our work provides a theoretical groundwork for understanding the empirical success of these earlier techniques.

In recent years, a number of systems have expanded empirical schema learning results beyond basic STRIPS operators. These include systems that modeled synthetic items (Holmes & Isbell Jr. 2005) and *probabilistic* STRIPS operators (Pasula, Zettlemoyer, & Kaelbling 2007) from traces of behavior. These contributions yielded exciting experimental results, but in this work we are examining the fundamental problems in schema learning, and thus we focus on simpler cases, particularly deterministic STRIPS and web-service operators.

Other algorithms, such as ARMS (Yang, Wu, & Jiang 2005), and SLAF (Shahaf 2007), considered learning when state information may be unseen or not immediately available, respectively. The SLAF research produced an algorithm and derived *computational* bounds on its runtime based on the type of language used (including STRIPS). In contrast, we are considering learning in fully observable environments and are concerned with the *sample* complexity of learning, which is also of critical importance in practical systems.

STRIPS Learning from Raw Experience

We first consider the STRIPS learning problem with *raw experience*, where at every step in an episode, the agent sends a grounded action to the environment and receives back either *success* and a new state (grounded fluents), or *failure* (meaning some precondition is not met). Note, the agent is not told what precondition was not satisfied, and may face uncertainty in the action's effects. To better understand the intricacies of this setting, we consider two special cases where parts of the operator descriptions are known *a priori*.

Case I: Preconditions Known In this case, the agent knows $\forall a$, a.PRE, but not a.ADD nor a.DEL. Under these conditions, an agent can use the *optimistic* algorithm, STRIPS-EffectLearn (Algorithm 1).

Lemma 1. *STRIPS-EffectLearn can make no more than $O(APm^n)$ PPMs in the "Preconditions known" case.*

Proof sketch. Since the preconditions are known, the agent does not have to worry about action failure. The schemas maintained by STRIPS-EffectLearn are optimistic; they treat any fluent whose membership in a.ADD or a.DEL is uncertain as being in a.ADD. When experience provides information on a fluent w.r.t. an action, it can be marked Yes or No based on update rules 2-5 (but never back to Unknown). PPMs can occur only if a plan invokes a state/action (s,a) where all previous occurrences of a took place with at least one fluent in a.ADD or a.DEL having a different truth value than it has in s. Since the number of Unknowns is initially $O(APm^n)$ and each PPM eliminates at least one Unknown, the total number of PPMs is bounded by $O(APm^n)$. □

Case II: Effects known In this case, $\forall a$, a.ADD and a.DEL are known and the agent must learn a.PRE. Unfortunately, the environment only indicates success or failure (not the cause), so fluents cannot be considered independently as they were in the previous case, leading to the following negative result (proof available in Walsh & Littman (2008)).

Algorithm 1 STRIPS-EffectLearn

1: $\forall a \forall f\, Add[a][f] := Delete[a][f] :=$ Unknown
2: Construct a set of optimistic action schemas, $\mathfrak{A}$, (one for each action) with the given preconditions and where
3: $\quad f \in a$.ADD if $Add[a][f]$ = Unknown OR Yes
4: $\quad f \in a$.DEL if $Delete[a][f]$ = Yes
5: **for** (s_0, $\mathcal{G}$) = nextEpisode() **do**
6: plan = PLANNER.makePlan($\mathfrak{A}$, s_0, $\mathcal{G}$)
7: **for** each $a \in$ plan **do**
8: s = current state
9: s' = plan.nextState(s, a)
10: take action a and receive $newState$
11: **if** $newState \neq s'$ **then**
12: Update Add and $Delete$, in case they changed,
13: Reconstruct the optimistic schema set $\mathfrak{A}$
14: plan = PLANNER.makePlan($\mathfrak{A}$, newState, $\mathcal{G}$)

Theorem 1. *Learning the necessary STRIPS operators to determine the reachability of a goal when the preconditions are not known can require $\Omega(2^P)$ PPMs.*

However, if we restrict the class of formulae permissible as preconditions to conjunctions of length k or less, where k is a constant, we gain the following result.

Lemma 2. *An agent in the "Effects known" setting can efficiently learn precondition conjunctions of k or fewer fluents with no more than $O(A(Pm^n)^k)$ PPMs.*

Proof sketch. We adapt *STRIPS-EffectLearn* by constructing an array for each action, Pre, of size $\sum_{i=1}^{k} \binom{Pm^n}{i} = O((Pm^n)^k)$ with all possible precondition *conjunctions* (valid hypotheses), all initially marked as Unknown. When constructing an action schema, we use one of the hypotheses that are marked as Unknown. If the planner returns "no plan" for this hypothesis, we try another one marked Unknown until we receive a plan, or run out of valid combinations of hypotheses, at which point we correctly predict "no plan". We note this may require an exponential (in A) number of calls to the planner. If we have a plan, the algorithm uses it, always predicting success. Upon failure, all the hypotheses that claimed the current state satisfied all preconditions are marked No and the algorithm replans. Each failure disproves at least one of the hypotheses marked Unknown. This monotonic search yields $O(A(Pm^n)^k)$ PPMs. □

Combining the two positive results above, we can show that STRIPS operators are efficiently learnable with bounded size preconditions even when PRE, ADD, and DEL are all initially unknown.

Theorem 2. *The STRIPS learning problem can be solved efficiently when the preconditions are guaranteed to be conjunctions of k or fewer fluents. Specifically, after $O(A(Pm^n)^{max(k,1)})$ PPMs, an agent can be guaranteed to always return a correct plan to the goal if there is one (or "no plan" if there is none), with no further PPMs.*

Proof sketch. The algorithms for the "Preconditions known" and "Effects known" cases can be combined so

that the agent is always using optimistic schemas. At every step, if an action fails or produces a different state than planned, the agent updates its schema and replans. As in the simpler algorithms, every PPM is guaranteed to monotonically refine one action's array entries. If $k \geq 1$, then the largest array is $|Pre| = O((Pm^n)^k)$, otherwise it is Add or $Delete$ of size $O(Pm^n)$. Schemas must be learned for all A actions, thus we obtain the desired bound. □

We have shown that it is possible to efficiently solve the STRIPS learning problem if $\forall a, |a.\text{PRE}| \leq k$. However, this assumption may not naturally hold in environments with a large number of predicates. Hence, we now consider a different form of interaction with the environment, where an agent can ask a teacher for a trace showing how to reach the goal in the current episode. Prior work has empirically demonstrated such traces speed up learning. We now show that the introduction of a teacher is also theoretically sufficient for efficient STRIPS learning.

STRIPS Learning with a Teacher

We consider a *teacher* that, given s_0 and $\mathcal{G}$, returns either a plan for the current episode that will achieve the goal, or "no plan" if the goal is not reachable. Our strategy in this setting revolves around the use of a *pessimistic* model where a.PRE and a.DEL contain all the fluents that have not been disproved, and no fluents are in a.ADD unless directly evinced. Due to this gloomy outlook, the agent will claim "no plan" anytime it cannot *guarantee* reaching the goal. It will then query the teacher, which responds with a *trace* showing grounded state/action pairs for the current episode. The full learning algorithm, STRIPS-TraceLearn, is described in Algorithm 2. We note that unlike the raw-experience case, all examples provided to this "bottom-up" learner will be positive, allowing us to more deftly prune the precondition hypothesis space (case 1 from the operator update rules), leading to the following theoretical result.

Theorem 3. *STRIPS-TraceLearn solves the STRIPS learning problem efficiently in the presence of a teacher. Specifically, it makes no more than $O(APm^n)$ PPMs.*

Proof sketch. STRIPS-TraceLearn will only make plan-prediction mistakes where the agent claims "no plan" and the teacher returns a valid plan trace. The trace will either indicate that a fluent in some a.ADD should be marked Yes or some fluent in a.PRE or a.DEL should be marked No (otherwise the agent would have discovered a valid plan itself). The resulting pessimistic schema is always consistent with all previous examples, since changes are only "one way". Thus, the number of changes, and therefore disagreements with the teacher, is bounded by $O(APm^n)$. □

This result, which is in line with earlier empirical findings, shows there is a true theoretical benefit, in terms of sample complexity, to interacting with a teacher rather than learning STRIPS operators solely from raw experience. We note that if the goal is always reachable from s_0, the number of requests to the teacher can be similarly bounded.

Algorithm 2 STRIPS-TraceLearn

```
1: ∀a∀f Add[a][f] := Delete[a][f] := Pre[a][f] := Unknown
2: Construct a pessimistic action schema set 𝔄 where
3:     f ∈ a.PRE if Pre[a][f] = Unknown or Yes,
4:     f ∈ a.ADD if Add[a][f] = Yes,
5:     f ∈ a.DEL if Delete[a][f] = Unknown or Yes
6: for each episode (s0, 𝒢) do
7:   plan = PLANNER.makePlan(𝔄, s0, 𝒢)
8:   if plan ≠ "no plan" then
9:     Execute the plan
10:  else
11:    trace = teacher.query(s0, 𝒢)
12:    for each ⟨s, a, s'⟩ ∈ trace do
13:      Update Add, Delete, and Pre.
14:      reconstruct 𝔄
```

Example

We now present an empirical result demonstrating STRIPS-TraceLearn in Blocks World with four blocks (**a**,**b**,**c**,**d**), a table (**t**), four predicates (*On*(X, Y), *Block*(X), *Clear*(X), *Table*(X)), and two actions (*move*(B, From, To) and *moveToTable*(B, From, T)). In the first episode, s_0 has all the blocks on the table and the goal is to stack 3 blocks on one another. The pessimistic agent, which initially thinks every possible pre-condition constrains each action, reports "no plan", but the teacher responds with the plan: [*move*(**a**, **t**, **b**), *move*(**c**, **t**, **a**)] The agent updates its *move* schema to reflect the new trace, as seen in the first schema in Table 1.

Next, the agent is presented with the same goal but an initial state where **a** is on **b** and **c** is on **d**. Because it has not yet correctly learned the preconditions for *move* (it believes blocks can only be moved from the table), the agent reports "no plan", and receives a trace [*move*(**c**, **d**, **a**)], which induces the second schema in Table 1. Notice *move*.DEL still contains several fluents that could never occur if the action succeeds (so their deletion cannot be empirically refuted), but otherwise the schema represents the true dynamics of *move*. Now, the agent receives the same initial state as the previous trace, but with a goal of ∃ X, Y *On*(X, Y), *On*(Y, d). Because the agent has learned the *move* action, it produces the plan [*move*(**a**,**b**,**c**)]. Further experience could refine the *moveToTable* schema.

Web Service Description Learning

We now consider the pragmatic goal of learning web-service descriptions. Protocols like REST and SOAP have standardized web-service syntax, but efforts towards *semantic* standardization have largely failed. While the common Web Services Description Language (WSDL) provides basic type information (integers, strings, etc.), most services do not provide documentation beyond this. Even if they do, the information may be out of date, and if one seeks to link services from multiple sources, matching semantics is unrealistic. For instance, if one service produces *Dog*(X) and another has a precondition *Canine*(X), significant background knowledge is needed to infer a link between them.

This situation is unfortunate, because a number of recent works (e.g. Liu, Ranganathan, & Riabov (2007)) have proposed fairly efficient methods for planning or "composing" web services with known semantics to achieve a goal. The languages used in these works are very similar to the action schemas we described earlier (preconditions, add-lists, delete-lists). We seek to leverage this similarity to extend our STRIPS learning algorithms to the web-service domain. By learning our own semantic interpretation of service descriptions, we parry the complication of missing or mismatched semantic descriptions.

STRIPS+WS for Web Services

One reason the basic STRIPS language is unable to model standard web services, is that service responses often reference objects that were hitherto unknown to the agent. For instance, a service *airlineLookup*(**new_york**, **paris**) might return a grounded fluent *Flight*(**f107**), where **f107** is completely new to the agent. This "object creation" problem has forced planning researchers to enlist linguistic structures such as exemplars (Liu, Ranganathan, & Riabov 2007). We consider a fairly simple modification to STRIPS, the introduction of functional terms, that allows us to model object creation. That is, while STRIPS planners ground every variable as some object, $\mathbf{o} \in \mathcal{O}$, our new language, STRIPS+WS, allows planners to reason about variables bound to functional terms. Schematically, this enhancement only inserts functional terms into a.ADD, so we label each i^{th} function associated with an action a as f_{ai}. For instance, in the airline example, we have *airlineLookup*(Source, Dest) $\rightarrow$ ADD: *Flight*(f_{a1}(Source, Dest)), where a is short for *airlineLookup*. We note that the add-list can contain fluents that reference both new and old objects such as *Flight-From*(f_{a1}(Source, Dest), Source). STRIPS+WS also embodies the following assumptions:

1. We extend the STRIPS scope assumption (objects in a.PRE, a.ADD, and a.DEL must be parameters) to allow functions ($f_{a1}...f_{aN}$) to appear as terms of fluents in a.ADD. These functions refer to objects that have not been seen before. In accordance with the STRIPS scoping assumption, no action schema can reference a function that is tied to another action, although the same function may appear in multiple fluents in a single action's ADD.
2. To keep the number of states finite, STRIPS+WS explicitly outlaws state descriptions where a function is nested in itself. This constraint prevents a planner from considering an infinite sequence of new objects being created.
3. The ordering of fluents referencing new objects in the state description is fixed. That is, if a state after action a contains fluents *P*($f_{a1}(\cdot)$), *P*($f_{a2}(\cdot)$), future occurrences of the action cannot report a state as *P*($f_{a2}(\cdot)$), *P*($f_{a1}(\cdot)$). This assumption is necessary to perform efficient and correct updates of the learned schema.

Web-Service Description Learning

The *web-service description learning problem* takes as input a set of predicates and actions and the maximum number of new objects an action may produce, N. The goal of an agent is again, when provided with s_0 and $\mathcal{G}$, either to produce a valid plan to reach the goal or correctly report "no plan". We note that valid plans may contain functional terms (e.g. *P*(f_{a1}(**obj1**))) that will be replaced by real objects when the agent actually executes the plan.

We note that the functions involved in these learned schemas need *not* be described in terms of their minimal parameters. That is, for an action a(X, Y, Z), all the functions in the add-list can be written as f_{ai}(X, Y, Z), even if they are only dependent on X. However, we *do* need to be concerned with matching functional terms that always produce the same objects. That is, if a(X) always produces *P*(X, NewObj) and *Q*(NewObj), we should represent both occurrences of NewObj with the same function (f_{a1}(X)), as we explore below.

Prior Work on Learning Web Services

A number of prior works have investigated *planning* in domains involving web services. Recent work includes analyses of planning tractability when new constants can be introduced (Hoffman, Bertoli, & Pistore 2007), a planning system using RDF graphs as a representation (Liu, Ranganathan, & Riabov 2007), and a study of the tractability of web-service composition using more general Description Logics (Baader *et al.* 2005). All these languages and planners address the novel problem of representing new objects created by services, usually with placeholders such as exemplars (Liu, Ranganathan, & Riabov 2007) or special output variables (Hoffman, Bertoli, & Pistore 2007). Our work is complementary to these planners, because we build our own operators from experience, and assume a planner exists that can decide if our current model encodes a path to the goal.

Several empirical studies of learning web-service behavior have been performed. Recent work leveraging background knowledge to semantically label inputs and outputs (Lerman, Plangrasopchok, & Knoblock 2006) produced impressive empirical results. In work more akin to ours, ILP techniques have been used to induce web-service descriptions from examples, based on known descriptions of other services (Carman & Knoblock 2007). Unlike their work, which relies on heuristic search and the presence of enough data to infer a description, our learning algorithm provides a method for selecting samples from the environment and bounds the number of such interactions.

Leaning STRIPS+WS with Traces and Experience

We now adapt STRIPS-TraceLearn to efficiently learn operators in STRIPS+WS when the agent has access to a teacher. We focus on cases with access to traces because web-service traces (as XML) are often easy to obtain.

To extend the *STRIPS-TraceLearn* algorithm to STRIPS+WS, we must deal with the ambiguity introduced when function values overlap, as in the NewObj example above. By default, the agent will assume that if it has always seen the same new object in two places, that the same function is producing them. This assumption is optimistic and could lead to the agent claiming a plan will reach the goal when it will not (example below). We will use this optimistic approach (formalized in Algorithm 3)

Algorithm 3 Updating Functional Terms

1: The first time action a is seen, update a.ADD using the same function for objects with the same name.
2: **for** each instance of a that is encountered **do**
3: Compare each fluent containing a new object to its schematic counterpart
4: **if** there are conflicts between the learned and observed mapping of functions to objects **then**
5: Make new functions more finely partitioning the conflicting objects and update a.ADD

to distinguish functions using traces and raw experience, while employing pessimism and traces to discover the function-free contents of the operator lists.

As a concrete example, consider seeing action a(**obj1**) produce grounded fluents *P*(**obj1**, **newObj1**) and *Q*(**newObj1**). This instance would result in the learning algorithm producing *P*(X, f_{a1}(X)) and *Q*(f_{a1}(X)) and inserting them into a(X).ADD. However, in this example there are really two functions, but in the previous instance they just happened to reference the same object. This representation could lead a planner to a PPM if another action b(X, Y) has preconditions *P*(X, Y) $\wedge$ *Q*(Y) and the planner uses f_{a1}(X) to fill Y. Such a mistake may not be caught by the agent until it executes a and sees something like *P*(**c**, **d**), *Q*(**e**). When this is seen, the procedure above will change one f_{a1} term to a new function.

Theorem 4. *Using STRIPS-TraceLearn and Algorithm* 3*, an agent in a STRIPS+WS environment with access to a teacher can make no more than* $O(A(N + Pm^n))$ *PPMs.*

Proof sketch. Learning can be done just as in STRIPS-TraceLearn, and using the algorithm above to deal with new objects. Each time an action succeeds, the same number of predicates involving *new* objects will always appear, and in the same order (by our earlier assumption), so matching the predicates up is trivial. The number of times new functions need to be created is bounded by AN. O(APm^n) PPMs can occur as in the STRIPS case if the algorithm predicts "no plan" at the initial state. But, another AN PPMs can occur when the agent encounters an ambiguous function, as above. When such a mistake is realized through experience, the agent replans (as in the previous raw experience case), and may claim "no plan". At that point, the episode ends, but a trace could be provided using the current state as the initial state. Even if the teacher is only available at s_0, this type of situation only leads to another AN PPMs. □

Conclusions

We proposed the first provably sample-efficient algorithms for learning STRIPS action schemas and derived a language and strategy for porting results to the web-service setting. Future goals include extensions to richer languages and a large scale empirical study. Our initial experiences with Amazon Web Services (amazon.com/aws) suggest our framework can be used successfully to learn to carry out natural tasks.

Acknowledgments

The authors thank DARPA IL and SIFT. We also thank Alex Borgida for helpful discussions.

References

Baader, F.; Lutz, C.; Milicic, M.; Sattler, U.; and Wolter, F. 2005. A description logic based approach to reasoning about web services. In *Proceedings of the WWW 2005 Workshop on Web Service Semantics (WSS2005)*.

Benson, S. 1996. *Learning Action Models for Reactive Autonomous Agents*. Ph.D. Dissertation, Stanford University, Palo Alto, California.

Carman, M. J., and Knoblock, C. A. 2007. Learning semantic definitions of online information sources. *Journal of Artificial Intelligence Research* 30:1–50.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 5:189–208.

Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *ICML-1994*.

Hoffman, J.; Bertoli, P.; and Pistore, M. 2007. Web service composition as planning, revisited: In between background theories and initial state uncertainty. In *AAAI-2007*.

Holmes, M. P., and Isbell Jr., C. L. 2005. Schema learning: Experience-based construction of predictive action models. In *NIPS-2005*.

Kakade, S. 2003. *On the Sample Complexity of Reinforcement Learning*. Ph.D. Dissertation, University College London, UK.

Lerman, K.; Plangrasopchok, A.; and Knoblock, C. A. 2006. Automatically labeling the inputs and outputs of web services. In *AAAI-2006*.

Littlestone, N. 1988. Learning quickly when irrelevant attributes abound. *Machine Learning* 2:285–318.

Liu, Z.; Ranganathan, A.; and Riabov, A. V. 2007. A planning approach for message-oriented semantic web service composition. In *AAAI-2007*.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29:309–352.

Russell, S. J., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall, first edition.

Shahaf, D. 2007. Logical filtering and learning in partially observable worlds. Master's thesis, University of Illinois at Urbana-Champaign.

Walsh, T. J., and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. Technical report, Rutgers University, Piscataway, NJ.

Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *ICML-1995*.

Yang, Q.; Wu, K.; and Jiang, Y. 2005. Learning action models from plan examples with incomplete knowledge. In *ICAPS-2005*.