Journal of Logic and Computation Advance Access published December 20, 2007

Utilizing Natural Language for One-Shot Task Learning

HYUCKCHUL JUNG, JAMES ALLEN and LUCIAN GALESCU, *Florida Institute for Human and Machine Cognition, Pensacola, FL 32502, USA.*
E-mail: {hjung, jallen, lgalescu}@ihmc.us

NATHANAEL CHAMBERS, *Department of Computer Science, Stanford University, Stanford, CA 94305, USA*
E-mail: natec@stanford.edu

MARY SWIFT, *Computer Science Department, University of Rochester, Rochester, NY 14627, USA.*
E-mail: swift@cs.rochester.edu

WILLIAM TAYSOM, *Florida Institute for Human and Machine Cognition, Pensacola, FL 32502, USA.*
E-mail: wtaysom@ihmc.us

Abstract

Learning tasks from a single demonstration presents a significant challenge because the observed sequence is specific to the current situation and is inherently an incomplete representation of the procedure. Observation-based machine-learning techniques are not effective without multiple examples. However, when a demonstration is accompanied by natural language explanation, the language provides a rich source of information about the relationships between the steps in the procedure and the decision-making processes that led to them. In this article, we present a one-shot task learning system built on TRIPS, a dialogue-based collaborative problem solving system, and show how natural language understanding can be used for effective one-shot task learning.

1 Introduction

Our daily activities typically involve the execution of a series of tasks, and we envision personal assistant agents that can help by performing many of these tasks on our behalf. To realize this vision, agents need to have the ability to learn task models. Researchers have attempted to learn these models by observation, creating agents that learn through observing the expert's demonstration [2, 11, 13]. However, these techniques require observing multiple examples of the same task, and the number of required training examples increases with the complexity of the task.

One-shot learning presents a significant challenge because the observed sequence is inherently incomplete—the user only performs the steps required for the current situation. Furthermore, their decision-making processes, which reflect the control structures in the procedure, are not revealed. Natural language (NL) can alleviate these problems by identifying (i) a useful level of abstraction of observed actions; (ii) parameter dependencies; (iii) hierarchical structure; (iv) semantic relationships between the task and the items involved in the actions;nd (v) control constructs not otherwise observable.

doi:10.1093/logcom/exm071

In this article, we present a system that learns task knowledge through observation accompanied by a natural language 'play-by-play'. Various specialized reasoning modules in the system communicate and collaborate with each other to interpret the user's intentions, build a task model based on the interpretation and check consistency between the learned task and a priori knowledge.

The play-by-play approach in NL enables our task learning system to build a task with high-level constructs that are not inferable from observed actions alone. The synergy between the information encoded in NL and the system's basic task knowledge makes it possible to learn a task structure that is completely different from what is demonstrated, yet represents the user's underlying intentions in the demonstration.

In addition to the knowledge about task structure, NL also provides critical information to transform the observed actions into more robust and reliable executable forms. Our system learns how to find objects used in the task, unifying the linguistic information of the objects with the semantic representations of the user's NL descriptions about them. The objects can then be reliably found in dynamic and complex environments.

In the following sections, we introduce a motivating domain and present a task learning system built on top of a dialogue-based collaborative problem-solving system, followed by performance results on real world tasks.

2 Motivating domain

Online purchasing is growing in popularity with advances in e-commerce, and the Web is used as a major information source for both online and offline purposes. However, automating online purchasing and web-information extraction is challenging because it consists of a series of steps within a dynamic web environment that is often populated with irrelevant information.

Figure 1 shows a sample dialogue in which a user teaches the system how to buy a book by submitting a requisition form with information that is found on a web site. After a single session, our system can learn a task model that can be executed almost flawlessly in tests with other books. The dialogue provides the high-level information that is essential to learn a task:

- Task goal: line 1 shows the top-level action (i.e. purchase) and its parameter (i.e. book).
- Task hierarchy: line 8 indicates the start of a new subtask, and lines 25 and 31 indicate the end of a (sub)task.
- Primitive steps: utterances describe the steps required to complete the task (e.g. line 2), and application actions are associated with the steps to realize them (e.g. line 3).
- Parameters: user utterances also provide the abstract information about the parameters of a step and its actions as well as the relationship between parameters.

NL also provides high-level control information that can help a task learning system to acquire complex control constructs. The dialogue in Figure 2 finds hotels near an address by visiting a travel web site and examining its search results, a list or table that is often presented over multiple web pages.

Lines 14–21 provide an example of how the user shows which part of the search result page he/she is interested in (e.g. a hotel list), what information to extract (e.g. hotel name and distance), and how to get to the next page where additional information exists. Line 20 tells a system that the user intends to repeat the steps in lines 14–19. With the system's knowledge that iterative procedures have termination conditions, the system asks for the condition in line 22 and the user defines it in line 23.

After this single demonstration, the system learns an iterative procedure whose behaviour is very different from the sequence of demonstrated actions. Without the information encoded in NL, multiple demonstrations would have been needed to learn iteration with machine-learning techniques: a loop

1. **"Let me teach you to buy a book"**
2. **"You go to the purchase form"**
3. *< Enters a URL in the browser and hits enter >*
4. **"We fill in the author field"**
5. *< Types an author into the author field >*
6. **"And the title field"**
7. *< Types a title in the title field >*
8. **"Now let me show you how to find the other information"**
9. **"Go to Amazon"**
10. *< Opens a new tab, enters a URL in the browser >*
11. **"We select the books tab"**
12. *< Clicks the tab labeled "books">*
13. **"Then select advanced search"**
14. *< Clicks the link labeled "advanced search">*
15. **"Put the title here"**
16. *< Types the title into the title search field >*
17. **"And put the author here"**
18. *< Types the author into the author search field >*
19. **"Then click the search button"**
20. *< Clicks the search button >*
21. **"Now we select the page for the book"**
22. *< Clicks the link with the title of the book >*
23. **"This is the price"**
24. *< Highlights the price >*
25. **"We're done here"**
26. *< Switches back to the book purchase form >*
27. **"We put the price here"**
28. *< Types the price into the price field >*
29. **"Now submit the form"**
30. *< Clicks the submit button >*

FIGURE 1. Request to purchase a book.

1. **"I'll show you how to find hotels near an address"**
2. **"You go to this website"**
3. *< Enters a URL in the browser and hits enter >*
4. **"Put hotels in the place name field"**
5. *< Types "hotels" in the place name field >*
6. **"The street address goes here"**
7. *< Types an address in the address field >*
8. **"Put the city here"**
9. *< Types a city name in the state field >*
10. **"And the state here"**
11. *< Types a state name in the state field >*
12. **"Click the search button"**
13. *< Clicks the button labeled "search">*
14. **"Here is the list of hotels"**
15. *< Select the hotel list in the search result >*
16. **"This is the hotel name"**
17. *< Highlights the hotel name in one item >*
18. **"And this is the distance"**
19. *< Highlights the distance >*
20. **"Then click next to get more results"**
21. *< Clicks the link labeled "NEXT" >*
22. **[System] "When should I stop searching?"**
23. **"When the distance is more than 2 miles"**
24. **"OK I'm finished"**

FIGURE 2. Find hotels near an address.

pattern must be discovered from multiple action traces and not all machine learning techniques can discover such a pattern. Likewise, conditionals can be learned from a single demonstration through NL phrases using 'if' and 'otherwise', whereas they would require several examples to be learned with conventional observation-based techniques.

3 Task learning system architecture

We built a task learning system called PLOW (Procedure Learning On the Web), an extension to TRIPS [7], a dialogue-based collaborative problem solving system that has been tested in many domains.

3.1 The TRIPS system

The TRIPS system provides the architecture and domain-independent capabilities for supporting mixed-initiative dialogue in a range of different applications and domains. Its central components are based on a domain-independent representation, including a linguistically based semantic form, illocutionary acts and a collaborative problem-solving model. Domain-independence is critical for portability between domains: the system can be tailored to individual domains through an ontology mapping between the domain-independent representations and the domain-specific representations.

Figure 3 shows the main components of the complete task learning system. The core components of TRIPS include (i) a toolkit for rapid development of language models for the Sphinx-III speech recognition system, (ii) a robust parsing system that uses a broad coverage grammar and lexicon of spoken language, (iii) an interpretation manager (IM) that provides contextual interpretation based on the current discourse context, including reference resolution, ellipsis processing and the generation of intended speech act hypotheses, (iv) an ontology manager (OM) that translates between representations, and (v) a surface generator that generates system utterances from the domain-independent logical form.

The parser uses a broad coverage, domain-independent lexicon and grammar to produce a detailed description of the semantic content (the Logical Form, or LF) of the input speech. The LF is a flat,

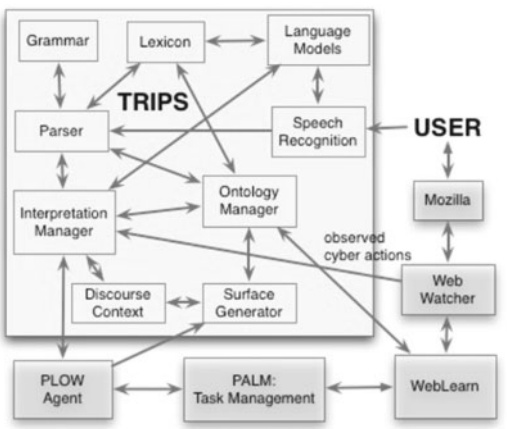

FIGURE 3. PLOW architecture.

unscoped representation that supports robust parsing and interpretation. It consists of a set of terms that describe the objects and relationships evoked by an utterance. The LF includes surface speech act analysis, dependency information, and word senses (semantic types) with semantic roles derived from a domain-independent language ontology. The parser disambiguates word senses based on linguistically motivated selectional restrictions encoded in the semantic types and roles. Word senses that have domain-specific mappings are tried first by the parser to improve efficiency. Figure 4 shows the LF-based linguistic semantics from the parser that includes the mapping between a user's word (e.g. w::book) and its linguistic semantics (e.g. lf::reserve).

The LF enhances system portability by providing a uniform representation of utterance content that is independent of the knowledge representation used by domain reasoners. The IM uses the LF to interpret utterances in context, including resolving referential expressions, recognizing intended speech acts, and interpreting elliptical utterances. These contextual interpretation algorithms are driven by the domain-independent LF representation, but interpretation hypotheses are evaluated with reference to a domain-specific reasoner such as our task learning agent (Figure 5). Note that the hypotheses are represented in a domain-dependent language for the reasoner. An ontology mapping procedure is used to translate the LF into a representational format suitable for the domain reasoners (see Dzikovska *et al.* [6] for details).

The IM also draws from the Discourse Context module to help resolve ambiguities in the input, and coordinates the synchronization of the user's utterances and observed actions. Based on the chosen hypothesis by IM, a domain-specific reasoner takes actions accordingly. For instance, given the interpretation in Figure 5, our task learning agent, PLOW in Figure 3, sends a 'start-learn' request to a related module, PALM, that manages a task model (Figure 6). Section 4.1 provides details of how this request is processed.

```
(w::let w::me w::teach w::you w::to w::book w::a
w::hotel)
⇒ ((lf::speechact v1 w::sa_request :content v2)
    (lf::f v2 (:* lf::allow w::let) :effect v5 :theme v3
    :agent v4 :tma ((w::tense w::pres)))
    (lf::f v5 (:* lf::transfer-information w::teach) :theme
    v6 :addressee v7 :agent v3)
      (lf::f v6 (:* lf::reserve w::book) :theme v8)
        (lf::a v8 (:* lf::accomodation w::hotel)
      (lf::pro v3 (:* lf::person w::me) :context-rel
      w::me)
      (lf::pro v7 (:* lf::person w::you) :context-rel
      w::you)
   (lf::impro v4 lf::person :context-rel w::*you*))
```

FIGURE 4. LF representation for 'Let me teach you to book a hotel'.

```
(cps-act :id cpsa1 :content (propose :who user :to
   *system :id x1 :what v5 :as goal)
   :context ((relational-entity v5 :instance-of teach
   :objective v6 :recipient v7 :agent v3)
   (relational-entity v6 :instance-of acquire :object v8)
   (a v8 :instance-of hotel))
   (the v3 :equals user)
   (the v7 :equals *system))
```

FIGURE 5. Interpretation hypothesis for the parsed information in Figure 4.

```
(request :receiver palm :content (expression :content p1
:context ((relational-entity p1 :instance-of start-learn
:task v6) (relational-entity v6 :instance-of acquire :object
v8) (a v8 :instance-of hotel))) :reply-with p2 :sender
plow)
```

FIGURE 6. Sending a start-learn request based on the interpretation in Figure 5.

The TRIPS system has been described in detail elsewhere [1, 7], so we will focus the rest of the discussion on the PLOW components that enable task learning from language.

3.2 Procedural knowledge learning system

The PLOW Agent provides the core agent capabilities for procedure learning and execution from interactions with the user. While it is specific to the application of procedure learning, it is not specific to any particular type of procedure. It models the interaction in terms of a set of collaborative 'meta' tasks that capture the actions in different modes of user interaction. PLOW maintains a stack of active tasks and is always executing the topmost task until it is completed.

The primary meta-task relevant to this article involves learning a procedure by demonstration. It encodes the typical structure of the interactions involved in demonstrating a new task, starting with explicitly naming the procedure to be learned, then performing a sequence of actions accompanied by a spoken play-by-play to define the procedure, and completed by an explicit ending statement. It will push new tasks onto its agenda in order to deal with situations that interrupt the demonstration temporarily. For instance, one such task involves clarification of a step that was not understood, and another allows the system to ignore actions while the user tries to restore an earlier state after making a mistake. PLOW can also push another learn-by-demonstration task on the stack when it recognizes that the user explicitly defines a subtask before introducing it as a step in the lower task of the stack. PLOW does not perform sophisticated intention recognition, but depends on the user's signals of task initiation and termination.

PLOW uses its stack of active tasks to support the interpretation of new utterances and cyber actions by evaluating hypotheses generated from the IM. The IM uses a set of conventional speech act interpretation rules to suggest possible intended problem solving acts. It uses a set of hand-built heuristics to pick its top hypothesis and requests the PLOW Agent to evaluate its task coherence at every stage in the dialogue. If rejected by PLOW, it tries alternate hypotheses until one makes sense in the current task context. Once PLOW receives input that makes sense given the current models, it uses the models to identify the reasoning actions that it will pass to PALM (Procedure Action Learning Module), which manages the reasoning required to construct the procedures. The task models also specify the content of responses that provides the basis for the system utterances.

Figure 7 shows a highly simplified trace of the messages between the IM, PLOW and PALM where the user demonstrates the next step in a procedure. IDENTIFY-SUBSTEP is a request to build a step described by the user into the task representation and IDENTIFY-EXAMPLE is a request to build an action that realizes the step. For each request, PALM sends PLOW a confirmation reply that includes the ID of a task or a step for future reference, notification of any problems, and assumptions that were made to interpret the request. The next section describes how PALM processes these requests.

To build an executable task model, agents need to understand how a human user perceives objects in an application (e.g. links/buttons/fields in a web browser) and learn how to identify them.

User says "***click on the search button***"
IM → PLOW: (commit (propose :action
(choose :object (button :associated-with search)))
PLOW → PALM: (identify-substep :task P1 :step
(choose :object (button :associated-with search)))
PALM → PLOW: (accepted :condition (step-id S1)
<<*User clicks on button B123*>>
IM → PLOW: (observed (submit-form :id B123))
PLOW → PALM: (identify-example :step S1 :action
(submit-form :object B423))
PALM → PLOW: (accepted ...)

FIGURE 7. Interaction between TRIPS modules.

Much of the semantic knowledge in the natural language descriptions can be used to help automate the identification of those objects that are also described in NL. The WebLearn module learns how to identify such objects by connecting the semantic IDENTIFY-SUBSTEP descriptions with the observed action descriptions from WebWatcher, the listener that observes user behavior on our web browser.

4 Acquisition of task knowledge

A task is built from primitive steps and hierarchical subtasks. To build a task model, the system needs to not only learn each primitive step, but also the relationships between steps and the hierarchical structure of a task. This section explains the process using the dialogue in Figures 1 and 2.

Information in TRIPS is expressed in AKRL (Abstract Knowledge Representation Language), a frame-like representation that describes objects with a domain-specific ontology using cross-domain syntax. AKRL retains the aspects of natural language that must be handled by the reasoning modules. While the details of AKRL are not presented in this article due to limited space, examples in this section are indicative of how it represents the natural language.

4.1 Task representation for learning/execution

The goal of task knowledge acquisition is to construct a task model that is not only executable by agents but also applicable to further learning. To help an agent reason about executable tasks, each learned task is explicitly represented in a form that specifies the task's goals, what triggers the task, and when it is completed:

*(**task :goal** <task description>*
* **:trigger** <triggering condition>*
* **:completion-condition** (state :completed-actions*
* <list of actions required to achieve this task>*
* :propositions <additional conditions>))*

A new task description is created from an utterance that explicitly states what a user intends to demonstrate (e.g. '*let me teach you to buy a book*', line 1 in Figure 1). The IM informs PLOW of the user's intention, and PLOW enters into a learning mode by sending a 'start-learn' request to PALM with a task description such as ((kr::relational-entity kr::v1 :instance-of kr::buy :object kr::2) (kr::a kr::v2 :instance-of kr::book)). The condition of task completion is based on the world state when a user finishes the demonstration. In a simple case, the world state can be the current ordered set of

completed actions. Additional conditions can be added to the completion condition as we will show in Section 4.6.2.

A task may also have a trigger: e.g. when a user says, '*let me teach you what to do when an email arrives*', the email arrival is captured as a trigger. When such an event occurs during execution, the agent invokes the task.

When a step and its actions are identified, PALM creates a rule that describes when the step should be executed. Each rule contains the precondition/name/parameters of the step, and the specific actions that realize the step:

(***rule :precondition***
(state ***:task*** *<task description>*
:ordering-constraint *<step-ordering>*
:propositions *<additional condition>)*
:step *<step description>*
:actions *<actions to realize this step>)*

The precondition of each rule includes the task description that is used as a cue for the step in the rule. When initiating a task, the task is pushed onto the active task stack and popped from the stack when it is finished. For each rule, the task description in the precondition is compared with the top element of the active stack. If there is a rule with a matching precondition (including the task description), the actions of the rule are executed and the world state changes, which may lead to subsequent rule firings until the task is complete. If a rule involves invoking a subtask, the ':step' part in the rule describes the subtask and no ':action' is specified. When such a rule is fired, the subtask is pushed onto the active task stack.

4.2 Learning primitive steps

When a user describes each step in NL, PLOW sends PALM a request to identify the step and build it into a task representation. For instance, when a user says, '*click the search button*' (line 19 in Figure 1), the utterance indicates what action the user intends to perform (a CHOOSE) and the arguments of this action (a BUTTON associated with the action SEARCH). Then, as shown in Figure 8, PLOW sends PALM a request to build the step into the task model, providing the abstract knowledge about the step in AKRL.

Given the request to identify a step, PALM checks the validity of the step based on its knowledge of the names and parameters of legal primitive steps. If valid, PALM records the step name and the value of each parameter. Each primitive step identified by a user has associated actions that realize the step. For instance, the CHOOSE step can be realized by an action of submitting a form. PALM expects the user to demonstrate this action.

When the user clicks on the search button, WebWatcher observes this action and reports it. PLOW links the observed action to the current step and sends PALM a request to learn from this example.

(**akrl-expression** :content V1 :context ((relational-entity V1 :instance-of **identify-substep** :content V2 :procedure-id P1) (relational-entity V2 :instance-of **choose** :agent V3 :object V4) (the V4 :instance-of **button** :associated-with V5) (the V5 :instance-of **search**) (the V3 :equals *system)))

FIGURE 8. Request to identify a step given user utterance 'CLICK THE SEARCH BUTTON' (line 19, Figure 1).

If the action is valid for this step, PALM builds it into the representation of a primitive step. Invalid actions are detected by checking PALM's knowledge base of legal actions for each primitive step: e.g. filling a form is illegal for a step to select a link. Figure 9 shows an example in which PLOW asks PALM to identify a user action 'submit-form' with the form ID as its parameter value.

If the user had performed this step without any verbal description, the learning system would not know the correct abstraction for representing the observed action. For instance, filling in a text field with some text may not provide enough information to infer the abstract information needed (e.g. title or author of a book). Conversely, some steps could be described in NL without real demonstration, but such an approach is not always feasible. NL descriptions without demonstration can be unreasonably verbose, ambiguous, or technically infeasible to interpret because of the difference between the perceptual features a user refers to and the internal representation of the object. To identify a web object (e.g. link, button, etc.), pointing or highlighting is a more natural and effective method, and combined with NL, provides a robust approach to learning.

4.3 Learning to identify objects

Some actions require the identification of application-specific objects (e.g. links, buttons and fields in a web page), especially when they exist in a dynamic environment like the Web. For instance, the SUBMIT-FORM action in Figure 9 refers to a form object with the identifier B123; in future visits to the same web page, the same form object will likely be in a different location.

When adding an action for a step that involves such dynamic objects, PALM uses WebLearn to learn how to identify the objects. WebLearn utilizes the NL description from the user to match with the NL description of the object on the web page. Since web pages are designed for human interpretation, matching the semantic form of the user's description to the language description of the web object proves to be very effective in learning robust descriptions of these objects [4].

PALM sends WebLearn the 'in-order-to' parameter of FIND-OBJECT (in Figure 10), containing the AKRL of the user's utterance. WebLearn utilizes the TRIPS Ontology and Lexicon to search the web object's text for occurrences of lexical realizations of this input AKRL. When found, WebLearn uses this information to build an object description that can be reused during execution. In essence, WebLearn finds the most important feature of the object by utilizing the user's description of it.

As an example, see Figure 11 for the search form of a commercial website. The user may instruct PLOW to, '*fill in the search field*'. The input to WebLearn is the interpreted AKRL

```
(akrl-expression :content V6 :context ((relational-entity V6 :instance-of identify-example :content V7 :step-id S1
:procedure-id P1) (relational-entity V7 :instance-of submit-form :object B123)))
```

FIGURE 9. Request to identify an example for the step in Figure 8.

```
(rule :precondition (state :completed-action (.....) :task (.....))
 :step  (act :name choose :parameter ((arg :name object :value ((the V4 :instance-of button :associated-with V4)
                                                                (the V5 :instance-of search))))
 :actions ((act :name find-object :parameter ((arg :name type :value form)
                                              (arg :name in-order-to :value < the description of this step in AKRL >)))
                                  :returns (value :type textfield :set-to X1))
          (act :name submit-form :parameter ((arg :name object :value (value-of X1))))))
```

FIGURE 10. PALM task representation for the step built on the information in Figures 8 and 9.

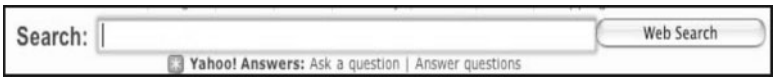

FIGURE 11. Input field at yahoo.com.

form, ((relational-entity v1 :instance-of fill :object v2) (relational-entity v2 :instance-of text-field :associated-with v3) (kind v3 :instance-of search)). WebLearn uses this representation with the Ontology and Lexicon to learn that the NL text next to the textfield, 'Search', describes the content of the user's intention. The user is using this field because of this NL on the web page, not because the field is of a certain size, in a certain position, or configured in some specific manner. By utilizing the user's description, we can then find this textfield again even if the web page changes its location. Note that this is not just text matching. Even if the input field title changes from 'Search' to 'Searching' or 'Find', WebLearn still finds the field using its ontological and linguistic knowledge.

4.4 Learning parameters and their dependencies

For an object used in a demonstration, the system has to be able to determine whether it should be a constant or a variable in the procedure and whether it has some relational dependency to other parameters already identified in the procedure. The system also has to determine if the object is an input or an output parameter. With traditional observation-based learning techniques, it would require numerous training examples to identify such information. However, with proper interpretation of the knowledge encoded in NL, we can effectively identify parameters and determine their bindings from one example.

Much information can be obtained from language through the definiteness feature. For instance, in Figure 2, a noun phrase such as 'an address' in line 1 is very likely to be an input parameter since it is indefinite and there is no deictic action involved. In the same utterance, we can identify an output parameter (i.e. hotels) and its type (i.e. a set) since the noun phrase 'hotels' are plural and it is an object to find.

Definite noun phrases are generally resolved using TRIPS' reference resolution capability, connecting the same instances of the parameters as they are used in the task. In Figure 1, the primitive steps that are identified by the user utterances, '*we fill in the author field*' (line 4) and '*put the author here*' (line 17) have the same parameter that refers to the author of the book to buy. The learned task model should have information that the two steps share the same value. When the IM analyses the utterances, it infers that both refer to the same object, namely the AUTHOR of B1, where B1 is the book to be bought, which was introduced in the first utterance, '*let me teach you to buy* ***a book***'. The same reasoning can be applied to other roles of a book such as its title.

4.5 Managing the task hierarchy

User utterances can be used as cues for the hierarchical segmentation of a task. For instance, in Figure 1, '*now let me show you how to find the other information*' (line 8) indicates that a user starts to teach a new (sub) task (i.e. finding other information), and the subsequent utterances will define this subtask. Given such interpretation, PLOW sends PALM a request to push a subtask under the current task. Then, PALM defines a new task and stores it in the task representation, adding it to the precondition of subsequent steps. When a user provides utterances that indicate completion in their linguistic semantics such as, '*we're done here*' (line 25) or '*that's it*' (line 31),

PLOW sends PALM a request to pop the current task and return to a pending task, if any. In this process, PALM builds a completion condition into the definition of the finished task based on the current world state. While decomposition or segmentation of a task is difficult or almost impossible to learn using observation-based techniques, the use of NL makes it feasible, moreover with a single demonstration.

4.6 Learning control structures

Natural language can greatly simplify learning complex control structure (e.g. conditions, iteration and recursion) that otherwise would be infeasible for some machine learning techniques and/or require multiple examples for complete learning [11]. In this section, we first present a simple control structure of conditions and show how iteration in Figure 2 can be learned through a specialized interface in a mixed-initiative interaction.

4.6.1 Conditionals

Conditionals have a basic structure of 'if X, then do Y', optionally followed by 'otherwise do Z'. Assume an email task of handling a book purchase request in which a user says, '*if the subject field contains this (clicking the email subject which has the text of 'book request'), then find the book information*'. Given the utterance, PLOW sends PALM a request to identify a conditional step. The request is similar to the 'identify-substep' request but with additional condition information that is inserted into the precondition of a rule for the step:

*(**rule :precondition** (state …*
* **:propositions** ((relational-entity V1 :instance-of*
* contain :container V2 :value "book request")*
* (the V2 :instance-of text-field :associated-with V3)*
* (kind V4 :instance-of subject)))**…**)*

Learning more complex conditionals (e.g. with an 'else' condition) typically requires an additional example. For instance, in extending the above condition, a user can say '*otherwise, move the email to this folder (select the email and drag it to a mailbox)*'. The system adds a new step from this utterance with the negation of the proposition used above.

4.6.2 Iteration

Learning iteration is essential since many real world applications involve the repetition of the same or similar tasks. Learning iteration with high-level control information from a single demonstration can greatly improve the learning system's efficiency because otherwise a large number of steps have to be demonstrated and the resulting learned task may not be flexible.

However, learning iteration requires special capabilities from the learning system because the sequence of the observed actions is often a partial instance of an iterative procedure (Figure 12). Given limited information, NL plays an important role in supplementing necessary elements of iteration as described below. In this section, we use a part of the dialogue in Figure 2 as an illustrative example.

4.6.2.1 Identifying iterative subtasks. In observation-based techniques, it is difficult for a system to correctly identify a segment of actions related to iteration without observing multiple examples.

FIGURE 12. One-shot iteration learning.

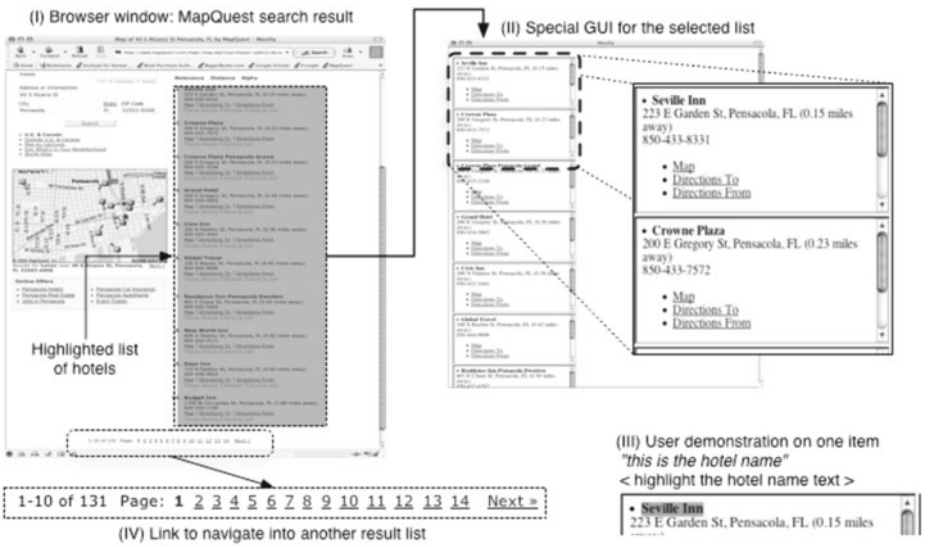

FIGURE 13. Search result and special GUI for iteration.

However, an explicit NL introduction for iteration such as '*here is the list of hotels*' (line 14 in Figure 2) shows the system the start of an iteration and the type of object to iterate over (i.e. a list). When a user highlights the list of search results in a browser window (Figure 13-I),[1] WebLearn learns how to identify such a list again in the future. Then, WebLearn uses a tree similarity metric over the document structure to identify the individual objects (rows) in the list. Rows in any table or list form (or even just a series of <p> objects) are typically similar in structure. It remains as a future work to allow a user to teach how to parse a list (e.g. how to divide or aggregate list elements when WebLearn's parsing is incorrect).

Since lists are often complex objects and the browser contains limitations to manipulating them, PLOW takes a novel approach that extracts a table and presents the table's rows in a special GUI window called TableGUI (Figure 13-II) in which a user can then continue demonstration. Using this approach, a user can verify if the table was correctly parsed and easily teach the system what to do with each table row since irrelevant information (in the original web page) has been filtered out by WebLearn's table parsing.

[1] PLOW uses an instrumented Mozilla browser that allows a user to select objects such as lists using special keys.

4.6.2.2 Actions to repeat in the iteration body. In presenting the parsed table that is already divided into individual rows by WebLearn, PLOW takes the initiative to ask the user to define steps/subtasks and show demonstration (e.g. lines 16–19 in Figure 2) on at least one of the rows in the TableGUI (Figure 13-III).

Here, PLOW learns an iterative subtask (i.e. 'for-loop' in Figure 12) with the steps defined in the TableGUI. That is, the user demonstrates what information to extract from one item in the list, and the system then build an iterative procedure to extract the same type of information from all list items. The iteration over a list is a special type of iteration in which iterator and termination conditions (i.e. until there is no more item) are implicit and can be automatically built into a task model.

Within an iteration, the user can define an arbitrary hierarchical subtask as described in Section 4.5. For instance, the user can teach how to navigate from the 'Directions To' link (opening a new browser) and find the distance from each hotel to a meeting place.

4.6.2.3 Identifying more lists and when to stop? Most search results on the Web are presented over multiple pages with a fixed number of items on each page. However, depending on the search terms, the number of pages can vary (Figure 13-IV). A user can specify whether to visit the next page in the iteration by explicitly saying, '*click next to get more results*' (line 20 in Figure 2) and click the link labelled 'Next'.

The system interprets '*get more results*' as a cue that the steps in line 14–21 need to be repeated on the next page, creating a different iterative task structure (e.g. 'do-until' in Figure 12). This type of iteration is explicitly demonstrated and its termination condition needs to be defined by the user. While the default termination condition for iteration could be when there is no more list item or page to examine, PLOW asks a user to explicitly define more specific conditions.

The user defined termination condition is set as a completion condition for the iteration subtask. During execution, when all the actions in the iteration body are performed, PLOW checks the termination condition (i.e. task completion condition) and, if unsatisfied, repeats the actions again. Currently, PLOW allows the following types of termination conditions:

- Number of pages: '*get three pages*'
- Number of results: '*get ten results*' (static max result number limit) or '*get a certain number of results*' (parameterized max limit)
- Value comparison: '*when the distance is more than two miles*'
  - The underlying NL system makes it possible to compare values. For instance, when a text '0.23 miles' is extracted, the text is parsed and saved in a computable representation so that the distance metric value can be checked against the value defined in the termination condition: *'0.23 miles'* $\rightarrow$ *((the V1 :instance-of length-value :cardinal-value V2) (a V2 :instance-of cardinal :numeric-value 0.23 :unit mile))*

Termination conditions can be placed in different locations within a task model, depending on their types. For instance, in Figure 12, the 'number of pages' termination condition is with the outer 'do-until' loop that iterates over page. In contrast, termination conditions with the result number and the value comparison are inserted into the inner for-loop that iterates over list items.

Figure 14 is an example of a partial result of this task execution: hotel list items associated with their extracted data (e.g. hotel name & distance). PLOW extracted the hotel name and the distance from each item in the hotel list. Learning iteration demonstrates how the knowledge encoded in NL helps a system to acquire complex task knowledge. By exploiting the properties of NL, PLOW transforms the observed actions into a totally different form, and this process is generic and domain-independent. Without specialized domain-dependent knowledge, it is infeasible for observation-based techniques to learn iteration from a single demonstration.

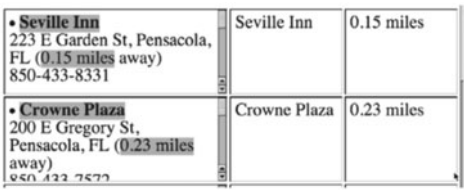

FIGURE 14. Task execution result example

4.7 Revising a task model

Currently, our system has limited capabilities to revise a task model. For instance, during learning, a user can tell the system to undo the last step in the learned task model by saying, '*scratch that*' or '*undo the last step*'. While the undo has an effect only on the task model, the user can change the application environment without affecting the learning process by telling the system to ignore some actions (e.g. 'let me reset the browser') and resuming demonstration later (e.g. 'let's resume'). More flexible methods to revise a task model (e.g. combination of verbalization and GUI actions on a window that shows a learned task) remain as future work.

4.8 Transfer of knowledge

Because language identifies what actions are being performed, we can relate the current action to prior instances of the same action. The language description can be parameterized so that the learned knowledge can be used for similar tasks by providing different values for the parameters. This enables transfer of knowledge to improve learning. In fact, PALM and WebLearn actually learn how to find and fill in fields that the user has never described.

Furthermore, since PLOW can identify whether it already knows actions corresponding to a step, it can learn a hierarchical task involving previously learned tasks without actually seeing a complete demonstration. For instance, after teaching the system how to buy a book (Figure 1), a user may give an instruction 'now you need to buy the book' in a dialogue for a different task (e.g. handling book request email). Given the instruction, PLOW checks the description of each learned task to find out whether it knows how to perform the described book purchase task. If PLOW finds a task model for book purchasing in its task repository, it inserts the task (as a subtask) into the current task.

5 Evaluation and discussion

5.1 Book search

The PLOW system learned the procedure to buy a book shown in Figure 1 with only one demonstration, and similar dialogues have also been tested on Amazon.com and other online book retailer web sites.

To measure the impact of natural language, we evaluated the performance of a 'macro-recorder' that simply recorded the web actions, ignoring user utterances. When the user selects a link (or text field, form, etc.) during training, it records the index of that link on the page (e.g. the 23rd link). Due to the limited capability of macro-style learning, we tested only the part of the task in Figure 1 that involves searching for a book (lines 9–22 in Figure 1), excluding the steps that fill in a requisition

form. For testing, the macro is a little more advanced than a basic macro in that it is parameterized with author and title.

We evaluated whether each system completed all the learned steps and found the correct book detail page for 162 randomly selected books at Amazon.com and 155 books at BarnesAndNoble.com (subset of Amazon's 162 books). The execution of the task model acquired by PLOW showed 97.5% and 96% success while the macro showed 61.3% and 36.4%, respectively. The PLOW error was due to incorrect links identified by WebLearn even though the learned task model was correct.

PLOW was also compared with a commercial tool for web information extraction, AgentBuilder by Fetch Technologies, which uses syntax-based web object identification and provides a GUI demonstration interface to the user. The task models built by AgentBuilder showed similar performance of 96% and 99% success on the two sites, respectively. However, compared with PLOW, AgentBuilder requires a user to navigate multiple GUIs and wizards, fill in names for schema items and connectors, and make choices from multiple radio buttons and popup menus, leading to more than five times the number of interactions to learn the same task.

5.2 Institution search

To test the correctness and the robustness of a learned iterative task, we did experiments with a small variation to the dialogue in Figure 2: line 4 '*put the institution name here*'. In execution, PLOW asks for four fields to be used in the MapQuest search: <institution, street address, city, state>. In the experiments, we randomly selected 25 real world examples with different institutions and addresses (e.g. <plumbers, 2200 Stonegate Lane, Wheaton, IL> and <art galleries, 55 Brunswick St., Rochester, NY>) and, given a task learned from one training example, we ran the task for the 25 examples with 20 as a max result number (i.e. 'get 20 results' as a termination condition). There were 350 (not 500) list items to be extracted from the MapQuest search results for the 25 examples (henceforth, called 'base search results') because MapQuest search results had less than 20 items for some institutions. It is out of the scope of this experimentation to check if a search result (e.g. a mattress store) falls into the specified institution (e.g. Bed & Breakfasts).

The experimental results show that, for every example, PLOW correctly followed all the steps and checked the termination condition in the learned task. However, it missed 34 items (e.g. the information about 'motel 6' is totally missed in finding hotels) and misparsed 2 items (i.e. two blank items in the parsed table, both are included in our results). Thus, the recall rate (the ratio of correctly identified results to the base search results) was 89.7% (= 314/350). The main source of failure is the error in list parsing due to inconsistent HTML format in the search results. The list parsing is based on a syntactic tree similarity approach rather than semantics from NL for which WebLearn is mainly designed. Despite missing 34 items among the base search results, PLOW's final return had 350 items (including two false empty items) by retrieving 34 additional items because PLOW's learned task iteration makes PLOW visit extra pages until the termination condition is satisfied. The percentage of correctly extracted results in the final return was 99.4% (= 348/350).

5.3 Discussion

PLOW is a general-purpose system that can be applied to other web tasks and can be easily extended to non-web tasks with instrumented applications that provide an API for PLOW to observe the user's actions (e.g. copy and paste in a text editor) and perform the actions. PLOW has been successfully used for one-shot learning of a couple dozen different tasks—and multiple variants thereof,

using different web sites, inputs and/or outputs—ranging from simple navigation and information extraction to complex nested iterative procedures.

Domains include weather forecasting, shopping (e.g. finding vendors for a product, finding products that fit a specification, finding products based on their numbers of positive reviews, etc.), travel (e.g. finding nonstop flights, finding hotels or restaurants near an address, deciding whether to rent a car or take a taxi from an airport to an address, etc.) and publications (e.g. finding someone's publications, finding citations or references for a publication, etc.).

In order to judge the effectiveness of PLOW with naive users, we recorded five subjects as they taught a mock-up system how to buy a book on the Amazon web site. The system recorded each verbal instruction and replied with verbal acknowledgments. The users were surprisingly similar in their approaches to searching. Seventy three of the 95 utterances can be interpreted by the current PLOW system after very small lexicon and grammar updates. The other 22 utterances can be classified into three categories:

(i) Dynamic Text: The user talks about text on the web page, creating complex noun phrases that are difficult to recognize and parse.
(ii) Instruction Repair: The user makes a mistake during instruction and verbally repairs the situation, such as, '*I might have spelled the author wrong*.'
(iii) Complex Instruction: The user gives a complex description to explain his actions, such as, '*we insert an n into jean <pause> ette.*'

6 Related work

6.1 Task learning system

Most task learning methods take advantage of existing knowledge by reusing, integrating and generalizing it. There are two extreme approaches for task learning: one is for an expert to encode his/her expertise in detail and communicate the knowledge to agents, and the other is for an agent to learn tasks without any supervision or guidance from an expert. As a middle ground to both approaches, researchers have explored techniques that are more practical and do not require too much effort from experts.

One major technique is observation-based task learning in which agents learn task models through observation of the actions performed by an expert [2, 11, 13]. Lent and Laird developed a system called KnoMic (Knowledge Mimic) that learns actions and their conditions from observation traces annotated by an expert. While KnoMic learns rules for actions, the Sheepdog system developed by Lau *et al.* applies a probabilistic approach based on Hidden Markov Models in aligning state-action pairs from traces. Lau and Weld applied the inductive logic programming approach to find actions and their pre-conditions from traces. Their approach shows the effectiveness in learning conditional loops over the version-space learning.

Angros *et al.* extended an observation-based approach by letting agents autonomously experiment with learned tasks and modify learned task models. However, their experimentation that simply omits different steps at each trial is not guided by general or domain knowledge. A significant drawback of these approaches is that they require multiple examples, infeasible for one-shot learning.

An alternative approach that does not rely on observation is the annotation-based technique that learns task models based on experts' annotations on selected examples [8]. While it requires an expert to encode expertise, machine-learning techniques often help agents to discover parameter bindings and constraints. Lee and Anderson also proposed a technique that does not require observation [12].

In their approach, given a task, declarative knowledge relevant to the task is retrieved, combined as a procedure, and translated into production rules based on general/domain knowledge. General means-end planning production rules are also used to improve the task model.

For the annotation-based approach, to help a human expert provide agents with task knowledge, tools have been developed [3, 9]. Gil *et al.* developed a system called EXPECT which checks dependency/consistency of user input, guides users to specify constraints and procedures using GUI. Blythe's Tailor system helps a user modify the procedure that is presented in NL and reasons about the effect of the change. However, neither of these systems can learn from observation.

The applicability of general machine learning techniques for task learning is limited due to an enormous search space. However, researchers have investigated approaches close to "programming by demonstration". For instance, Nicolescu and Mataric developed a technique which enables a robot to learn high level complex tasks by observing actions from a human, building tasks in the form of a behavior network [15]. Yang et al. developed a propositional satisfiability (SAT) based algorithm to find action sequences from plan examples [18].

While these task-learning systems are designed and customized for different applications, they all require multiple examples, making one-shot learning infeasible. Furthermore, unlike in PLOW, the role of NL in the systems that use NL (e.g. annotation-based learning and Tailor) is limited in that either the NL does not provide any semantic information or the encoded knowledge in NL is applied to modify only a single step under scrutiny rather than to build a hierarchical task with complex control constructs.

6.2 Language processing

Our broad-coverage, linguistically motivated parser produces sophisticated semantic representations of natural language that are suitable for the contextual interpretation of an utterance. For use in reasoning tasks, these representations are transformed into a format that can be used by domain reasoners via an ontology mapping procedure.

Other language interpretation systems that produce detailed semantic representations, such as LINGO ERG [5] and XLE [14], rely on information that can be extracted from syntactic structure. They do not provide semantic types and roles that can be mapped to domain actions and objects, which limits their use in complex reasoning tasks.

Applications that require more complex reasoning typically use specifically engineered grammars that directly link language and domain concepts (e.g. [16, 17]). Such language processing components lack portability and are not well equipped to handle contextually dependent interpretation phenomena such as pronouns or ellipsis.

7 Conclusion

We have presented a one-shot task learning system that learns complex tasks based on demonstration accompanied by a spoken 'play-by-play' and our system evaluation shows its effectiveness in learning real-world tasks. The knowledge encoded in language enables agents to learn complex tasks in a more accurate, robust and natural way. The task representation in this article not only retains the information from a user's verbal description but also stores it in an executable and computable form. Furthermore, the acquired knowledge can be reused. The key ideas from this approach could shed more light on the usefulness of natural language in one-shot task learning. Future work includes learning more

complex constructs (e.g. recursion), reasoning about learned knowledge (e.g. completeness and error checking, optimization), and generalization in further learning.

References

[1] J. Allen, N. Blaylock and G. Ferguson. A problem solving model for collaborative agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, ACM Press, New York, NY, USA, 2002.

[2] R. Angros, L. Johnson, J. Rickel and A. Scholer. Learning domain knowledge for teaching procedural skills. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, New York, NY, USA, 2002.

[3] J. Blythe. Task learning by instruction in tailor. In *Proceedings of the International Conference on Intelligent User Interfaces*, ACM Press, New York, NY, USA, 2005.

[4] N. Chambers, J. Allen, L. Galescu, H. Jung and W. Taysom. Using semantics to identify web objects. In *Proceedings of the National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, USA, 2006.

[5] Copestake and D. Flickinger. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second International Workshop on Language Resources and Evaluation*, ELDA, Paris, France, 2000.

[6] M. Dzikovska, J. Allen and M. Swift. Linking semantic and knowledge representations in a multi-domain dialogue system. *Journal of Logic and Computation*, Oxford University Press, Oxford, UK, in press.

[7] G. Ferguson and J. Allen TRIPS: an integrated intelligent problem-solving assistant. In *Proceedings of the National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, USA, 1998.

[8] Garland, K. Ryall and C. Rich . Learning hierarchical task models by defining and refining examples. In *Proceedings of the International Conference on Knowledge Capture*, ACM Press, New York, NY, USA, 2001.

[9] Y. Gil, J. Blythe, J. Kimand S. Ramachandran. Acquring procedural knowledge in EXPECT. In *Proceedings of AAAI Fall Symposium,* AAAI Press, Menlo Park, CA, USA, 2000.

[10] T. Lau, L. Bergman, V. Castelliand and D. Oblinger . Sheep Dog: learning procedures for technical support. In *Proceedings of the International Conference on Intelligent User Interface*, ACM Press, New York, NY, USA, 2004.

[11] T. Lau and D. Weld. Programming by demonstration: an inductive learning formulation. In *Proceedings of the International Conference on Intelligent User Interfaces,* ACM Press, New York, NY, USA, 1999.

[12] F. Lee and J. Anderson Learning to act: acquisition and optimization of procedural skill. In *Proceedings of the Annual Conference of the Cognitive Science Society*, Psychology Press, Hove, UK, 1997.

[13] M. Lent and J. Laird. Learning procedural knowledge through observation. In *Proceedings of the International Conference on Knowledge Capture*, ACM Press, New York, NY, USA, 2001.

[14] J. Maxwell and R. Kaplan. An efficient parser for LFG. In *Proceedings of the First Lexical Functional Grammar Conference*, University of Chicago Press, Chicago, IL, USA, 1996.

[15] M. Nicolescu and M. Mataric Experienced-based representation construction: learning from human and robot teachers. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE Computer Society, Los Alamitos, CA, USA, 2001.

[16] M. Rayner, B. A. Hockey and G. Dowding. Grammar specialization meets language modelling. In *Proceedings of the 7th International Conference on Spoken Language Processing*, ISCA Archive, Bonn, Germany, 2002.

[17] S. Seneff. TINA: a natural language system for spoken language applications. *Computational Linguistics*, **18**, 61–86, MIT Press, MA, USA, 1992.

[18] Q. Yang, K. Wu and Y. Jiang Learning action models from plan examples with incomplete knowledge. In *Proceedings of the International Conference on Automated Planning and Scheduling*, AAAI Press, Menlo Park, USA, 2005.

Received 31 July 2006