Toward Open Knowledge Enabling for Human-Robot Interaction

Xiaoping Chen, Jiongkun Xie, Jianmin Ji, and Zhiqiang Sui
Computer School, University of Science and Technology of China

This paper presents an effort on enabling robots to utilize open-source knowledge resources autonomously for Human-Robot Interaction. The main challenges include how to extract knowledge in semi-/unstructured natural languages, make use of multiple types of knowledge in decision-making, and identify the missing knowledge. Techniques for multi-mode Natural Language Processing, integrated decision-making and open knowledge searching are proposed. The OK-KeJia robot prototype is implemented and evaluated, especially with two tests on 11,615 user tasks and 467 user desires, respectively. The experiments show that the overall performance improves remarkably due to the use of appropriate open knowledge. In the later test, the percentage of fulfilled user desires increases from 0.86% to 28.69% when 15,020 open knowledge rules plus WordNet synonymies are used.

Keywords: Human-robot interaction, open knowledge, NLP, decision-making, social robotics

1. Introduction

Human-Robot Interaction (HRI) draws more and more interest from researchers (Cantrell et al., 2012; Rosenthal et al., 2011; Kaupp et al., 2010; Chen et al., 2010; Tenorth & Beetz, 2009; Doshi & Roy, 2007; Thrun, 2004; Fong et al., 2003; Burgard et al., 1999). In HRI settings, robots should be able to communicate with users, understand users' requests and provide services for users accordingly by taking physical and other actions. For these purposes, a robot needs a lot of knowledge. For instance, when a user tells a robot "I am thirsty", the robot is expected to do something to meet the user's desire. If the robot knows that this desire may be met by serving a drink to the user, then it can plan and execute some actions towards this goal. In many cases, however, it is too hard to equip a robot with complete knowledge before it is put to use. In these cases, the challenge is to develop robots that can acquire, ideally automatically, missing knowledge from somewhere for the tasks requested by users at running time.

There are open-source knowledge resources available on the web, such as Cyc[1], the Open Mind Indoor Common Sense database (OMICS) (Gupta & Kochenderfer, 2004), ontologies, and household appliances manuals. Robots can also get knowledge through human-robot dialogue. In this paper, we call the knowledge from these resources *open knowledge* and report an effort on online acquiring and making use of open knowledge for HRI. We consider three requirements in this effort. (i) The robots should be able to "understand" knowledge in natural language, since a great proportion of open knowledge

[1] http://www.opencyc.org/

is expressed in natural language. (ii) The robots should be capable of using different types of knowledge, because a single user task may involve multiple types of knowledge that exist in more than one open-source knowledge resource. (iii) A robot should be aware of what knowledge it lacks for a given task and can search for the missing knowledge from open-source resources. When a robot encounters knowledge shortages, it should be able to recognize what gaps exist between the knowledge it already possesses and the task at hand, and then try to find the relevant pieces of knowledge.

We have made a continual effort on developing intelligent service robots that can meet these requirements jointly in the KeJia project. The main ideas are sketched below. Firstly, we develop multi-mode NLP techniques for comprehension and extraction of knowledge in different modes of natural language expressions, and transformation of the extracted knowledge into an intermediate representation. Secondly, we propose an integrated decision-making mechanism based on a uniform representation of knowledge, so that different types of knowledge can be made use of. Thirdly, we put forth a principle for detecting knowledge gaps between the current task and the local knowledge of a robot, and introduce mechanisms for searching missing knowledge from open-source resources.

There are projects sharing a part of common concerns with us, but not all mentioned above. KnowRob (Tenorth & Beetz, 2009; Lemaignan et al., 2012) also use open knowledge such as Cyc ontology and OMICS in HRI. The authors study how extraction, representation and use of the knowledge can enable a grounded and shared model of the world suitable for later high-level tasks such as dialogue understanding. A specialized symbolic knowledge representation system based on Description Logics is employed. While their approach is action-centric, where a robot collects and reasons about knowledge around action models, our approach is open-knowledge-centric, focusing on automatically acquiring and utilizing open knowledge for on-line planning with general-purpose decision-making mechanisms. (Cantrell et al., 2012) is similar to our work in that open knowledge in natural language is formalized in order to update the planner model and enhance the planning capability of a robot. The major difference lies in the resources and scale of open knowledge, since we use large-scale knowledge resources (e.g., OMICS). In addition, our formalization of open knowledge concerns both unstructured and semi-structured natural language expressions. (Rosenthal et al., 2010) proposes a symbiotic relationship between robots and humans, where the robots and humans benefit each other by requesting and receiving help on actions they could not performed alone due to limitations of abilities. The Cognitive Systems for Cognitive Assistants (CoSy) project makes a continual contribution on the human-robot dialogue processing for HRI. (Kruijff et al., 2010) postulates a bi-directionality hypothesis which relates the linguistic processing with the robots' experiences associated with a situated context. They show how such bi-directionality between language and perception influences the situated dialogue processing for HRI. Like CoSy, the situated dialogue processing for HRI is also needed in our system, which provides our robot a foundation for understanding the requests from users and searching for open knowledge. The integrated decision-making mechanism proposed here is similar to Golog (Levesque et al., 1997) to a great extent. However, missing steps in a sequence as constraint are allowed and can be filled in by our mechanism, but not in Golog. In fact, Golog programs are intended to be high-level control programs of robots written by the designer, while our approach aims to make robots work by user requests and open knowledge. (Talamadupula et al., 2010) tries to adapt planning technology to Urban Search And Rescue (USAR) with a human-robot team, paying special attention to enabling existing planners, which work under closed-world assumptions, to cope with the open worlds in

USAR scenarios. We try to adapt planning technology to HRI, especially by using open knowledge.

In Section 2 we present the framework and architecture of the OK-KeJia robot, describes the main ideas of the whole work. The implementing techniques for two main modules of the robot, multimode NLP and integrated decision-making, are addressed in Section 3 and 4, respectively. Some case studies are reported in Section 5 and conclusions are given in Section 6.

2. System Overview

To describe the framework of OK-KeJia robots, we adopt a simple and general language for knowledge representation in this paper. The vocabulary of the language includes objects, predicates and action names. A predicate expresses an attribute of an object, environment or human. For instance, $small(obj)$ expresses that object obj is small. A predicate or its negation is called a literal.

We assume that a robot is equipped with a set of primitive actions. A *primitive action* (action for short) is specified as an ordered pair $\langle pre\text{-}cond(a), \mathit{eff}(a)\rangle$, called an action description, where a is a action name, $pre\text{-}cond(a)$ and $\mathit{eff}(a)$ are sets of literals. Intuitively, $\mathit{eff}(a)$ is the effects of executing action a and $pre\text{-}cond(a)$ is the pre-conditions under which a can be executed and $\mathit{eff}(a)$ can be reached through execution of a. For instance, $pre\text{-}cond(move) = \{nav\text{-}target(l), \neg robot\text{-}at\text{-}loc(l)\}$ and $\mathit{eff}(move) = \{robot\text{-}at\text{-}loc(l)\}$. The set of all the action descriptions is called the *action model* of the robot and taken as its built-in knowledge. We require that every action a is implemented by a routine on a real robot, with $\langle pre\text{-}cond(a), \mathit{eff}(a)\rangle$ being the expected action model.

In HRI settings, an action model is generally insufficient for a robot to provide services to users, since user requests may contain predicates that do not appear in the action model. For example, "thirsty" does not generally appear in the action model of a robot. In this paper, we consider three more types of knowledge. (i) *Conceptual Knowledge*, i.e., knowledge about relationships between concepts. Typically, ontologies specify such relationships, like super and equivalent classes of concepts. (ii) *Procedural Knowledge*, i.e., knowledge describing the steps of how to accomplish a task (relationship between a task and its sub-tasks). (iii) *Functional Knowledge*, i.e., knowledge about effects of tasks, sometimes called *goals*. For instance, the effects of task "*put object 1 on table 2*" can be expressed as a set of literals, $\{on(object_1, table_2), empty(grip)\}$. Manual instructions include functional knowledge, e.g., about the functions of buttons of a microwave oven.

Theoretically, a *growing model* is defined as $M = \langle A, C^*, P^*, F^*\rangle$, where A is the action model, C^*, P^* and F^* store the conceptual, procedural and functional knowledge that the robot has obtained, respectively. We assume in this paper that an element of C^* maps a concept into its superclass/subclass concepts, P^* a task into its sub-tasks, and F^* a task into its effects, respectively. C^*, P^* and F^* can expand during the robot's running. The model provides a framework for analyzing the main research issues. In particular, the integrated decision-making mechanism should be so constructed that knowledge from C^*, P^* and F^* can be made use of by it. Moreover, knowledge gaps can be identified against the differences between a user task and the local knowledge in M.

The overall architecture of our robot OK-KeJia is shown in Figure 1. The robot is driven by input from human-robot dialogue. The information extracted from the dialogue is further processed by the multi-mode NLP module. The integrated decision-making module tries to generate a plan for the current user task. If succeeds, the plan consisting of primitive

actions is fed to the low-level control module to execute. The robot tries to acquire open knowledge when it detects knowledge gaps for the current task. Then the open knowledge searching module is triggered to obtain relevant pieces of knowledge from some open-source knowledge resources. Some meta-control mechanism (Chen et al., 2012) is also needed for the coordination of these modules, but not shown in Figure 1. The robot's sensors include a laser range finder, a stereo camera and a 2D camera. The robot has an arm for manipulating portable items. The on-board computational resource consists of two laptops.

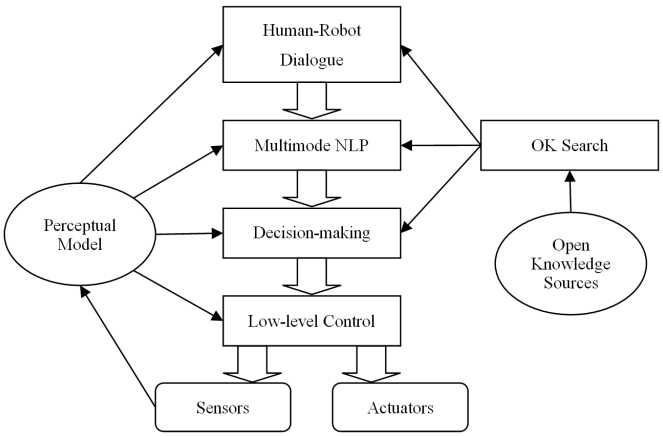

Figure 1. The overall architecture of OK-KeJia

The *human-robot dialogue* (HRD) component provides the interface for communication between users and the robot. The Speech Application Programming Interface (SAPI) developed by Microsoft is used for the speech recognition and synthesis. Once a user's utterance is captured by the recognizer, it will be converted to a sequence of words. The embedded *dialogue manager* (Figure 2) then classifies the dialogue contribution of the input utterance by keeping track of the *dialogue moves* of the user. According to the dialogue move, the HRD component decides to update the *world model*, which contains the information from the perceptual model and of the robot's internal state, and/or to invoke the *decision-making* module for the task planning, with the semantic representation of the input utterance produced by the *multi-mode NLP* module (Section 3). At present, the structure of the dialogue is represented as a finite state transition network. Figure 2 shows our implementation (i.e., finite state machine) of managing a simple human-robot dialogue in which the user tells the robot facts that he/she has observed or tasks, and the robot asks for more information. A mixed-initiative dialogue management will be developed in our future work.

Assume that a robot's perception of the current environmental and internal state is expressed as a set of literals, called an *observation*. And user tasks are transformed into (dummy, sometimes) goals. Given an observation o and a goal g, a *plan* for $\langle o, g\rangle$ is defined as a sequence $\langle o, a_1, \ldots, a_n, g\rangle$, where $a_1, \ldots, a_n$ are primitive actions, such that the goal g will be reached after the execution of $a_1, \ldots, a_n$ under any initial state satisfying o. In the literature, there are two basic schemas of decision-making for autonomous agents and robots, which can be employed in open knowledge settings. One is *goal-directed plan-*

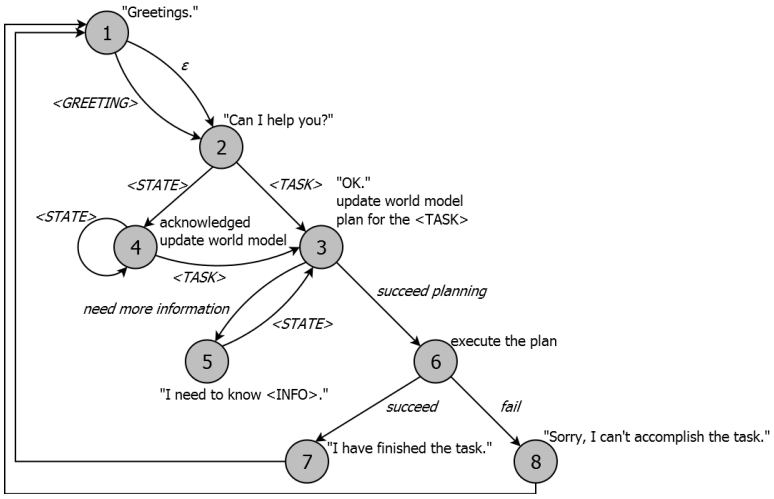

Figure 2. The finite state machine for a simple human-robot dialogue

ning, the procedure of generating a plan $\langle o, a_1, \ldots, a_n, g\rangle$ for any $\langle o, g\rangle$ with functional knowledge. The other one is *task-directed action selection*. This schema employs procedural knowledge iteratively, until the task is decomposed into a sequence of primitive actions. In Section 4, we present a hybrid schema that integrates some mechanism of task decomposition into a goal-directed planning system.

The ability of acquiring open knowledge depends on the detection of knowledge gaps between the current task and the robot's local knowledge. Let $M = \langle A, C^*, P^*, F^*\rangle$ be the growing model of the robot, and $\mathit{eff}(A) = \{p | p \in \mathit{eff}(a)$ for some $a \in A\}$, i.e., the union of all $\mathit{eff}(a)$ where a is a primitive action in A. A predicate Q is *grounded in* M if following conditions hold: (i) $Q \in \mathit{eff}(A)$; or (ii) Q is "reduced" (see Section 4) to $Q_1, \ldots, Q_n$ by $C^* \cup P^* \cup F^*$ such that each Q_i $(i = 1, \ldots, n)$ is grounded in M. A task is grounded in M if every predicate in the description of the task is grounded in M. This leads to the definition of knowledge gaps: There is a *knowledge gap* between a user task p and the robot's growing model M, if and only if there is a predicate Q in p such that Q is not grounded in M. Now we describe *the principle of open knowledge searching* as follows: Given a task p and a growing model M such that there is a knowledge gap between p and M. Search open-source knowledge resources to find C^+, P^+ and/or F^+ (i.e., new knowledge) so that there is no knowledge gap between p and $M^+ = \langle A, C^* \cup C^+, P^* \cup P^+, F^* \cup F^+\rangle$. We develop searching algorithms in accordance with each chosen resource of open knowledge based on this principle. In each case study we conducted, the robot accumulated knowledge during the process of one task set, but does not across task sets, since that would involve consistency issue of the acquired knowledge.

3. Multi-mode NLP

In this section, we demonstrate how to formalize the knowledge in unstructured natural language (e.g., in human-robot dialogue and manual instructions) and in semi-structured natural language (tuples in OMICS) to an intermediate language called Human-Robot Dialogue Structure (HRDS). HRDS captures the semantics of natural language sentences. Its

syntax is Lisp-like (see Appendix A) and it can be translated further into the ASP language (Section 4). HRDS has not been fully developed for handling with all the situations in human-robot dialogue, yet it is sufficient for our needs in this paper. Handlings of unstructured and semi-structured knowledge share the same underlying formalization (i.e., syntactic parsing and semantic interpretation), though the semantic interpretation of the OMICS knowledge needs further processing. By the same mechanism, both Chinese and English are processed with a slight difference in configuration, particularly the lexicon. Therefore, our robot can use open knowledge in both languages for one and the same task. However, grammar plays a less important role in Chinese language, which weakens the performance of the same mechanism in processing Chinese to some extent. This paper presents the multi-mode NLP techniques for English expressions of knowledge.

3.1 Formalizing the knowledge in natural language

The translation process consists of the syntactic parsing and the semantic interpretation. In the syntactic parsing, the Stanford parser (Klein & Manning, 2003) is employed to obtain the syntax tree of a sentence. The semantic interpretation using λ-calculus (Blackburn & Bos, 2005) is then applied on the syntax tree to construct the semantics. For this purpose, a lexicon with semantically annotated lexemes and a collection of *augmented syntax rules* (as-rules, for short) are hand crafted in our current implementation. However, (Ge & Mooney, 2009) provides a promising machine learning approach to automatic constructing such a lexicon with the as-rules, which will be introduced into our work in the future.

Table 1: Part of the lexicon

Word	**Category**	**Semantics**
if	IN	$\lambda p.\lambda q.(\mathbf{cause}\ p\ q)$
the	DT	$\lambda x.\lambda y.y@x$
you	PRP	$\lambda x.x@robot$
will	MD	$\lambda r.r@(t+1)$
press	VBP	$\lambda p.\lambda x.(\mathbf{fluent}\ (\mathbf{pred}\ press\ l\ x\ Y)$ $(\mathbf{conds}\ (\mathbf{pred}\ at\_time\ l\ t)\ p@(\lambda y.y@Y)))$
begin	VB	$\lambda q.\lambda r.\lambda p.(\mathbf{fluent}\ (\mathbf{pred}\ begin\ l\ X\ Y)$ $(\mathbf{conds}\ (\mathbf{pred}\ at\_time\ l\ r)\ p@X\ q@Y))$
microwave	NN	$\lambda x.((\mathbf{pred}\ oven\ x)\ (\mathbf{pred}\ microwave\ x))$
microwave	NN	$\lambda p.\lambda q.(p@q\ (\mathbf{pred}\ microwave\ q))$
oven	NN	$\lambda x.(\mathbf{pred}\ oven\ x)$
START/RESUME	NN	$\lambda p.\lambda q.(p@q\ (\mathbf{pred}\ start\_resume\ q))$
button	NN	$\lambda x.(\mathbf{pred}\ button\ x)$
cooking	NN	$\lambda x.(\mathbf{pred}\ cooking\ x)$
...	...	...

A semantically annotated lexeme lists the syntactic category of a word and its semantics represented by a λ-calculus formula, as shown in Table 1. One word could have multiple senses (i.e., λ formula). For example, the word *microwave* has two senses: one represents the concept *microwave oven*, the other gives the partial semantics of a compound noun (e.g., microwave oven). The combinations of all senses of each word in a sentence will be

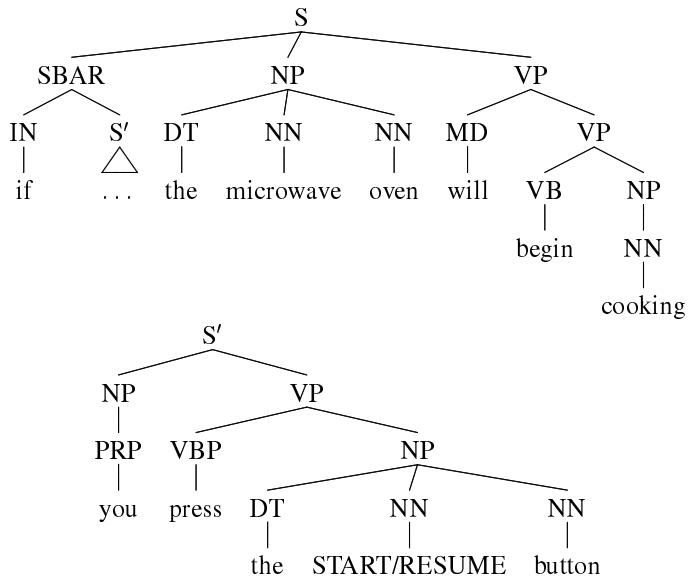

Figure 3. Syntax tree of sentence "*if you press the START/RESUME button, the microwave oven will begin cooking*".

analyzed in the semantic interpretation.

An as-rule augments the corresponding syntax rule with an extra slot which combines the semantic interpretations of the parts of the rule's right hand side. For example, the as-rule VP($vp := vb@np$) $\rightarrow$ VB(vb)NP(np) specifies that the semantic interpretation of VP results from applying the semantic interpretation of VB to that of NP. The notation '@' denotes such application. Some of the as-rules are shown below:

$$\text{S}(s := sbar@(np@vp)) \rightarrow \text{SBAR}(sbar)\ \text{NP}(np)\ \text{VP}(vp)$$
$$\text{S}(s := np@vp) \rightarrow \text{NP}(np)\ \text{VP}(vp)$$
$$\text{SBAR}(sbar := in@s) \rightarrow \text{IN}(in)\ \text{S}(s)$$
$$\text{NP}(np := dt@(nn1@nn2)) \rightarrow \text{DT}(dt)\ \text{NN}(nn1)\ \text{NN}(nn2)$$
$$\text{NP}(np := nn) \rightarrow \text{NN}(nn)$$
$$\text{NP}(np := prp) \rightarrow \text{PRP}(prp)$$
$$\text{VP}(vp := md@vp) \rightarrow \text{MD}(md)\ \text{VP}(vp)$$
$$\text{VP}(vp := vbp@np) \rightarrow \text{VBP}(vbp)\ \text{NP}(np)$$
$$\text{VP}(vp := vb@np) \rightarrow \text{VB}(vb)\ \text{NP}(np)$$

Once the syntax tree of a sentence is generated by the Stanford parser, its semantics is computed using the β-conversion with corresponding lexemes and as-rules. Given the sentence *if you press the START/RESUME button, the microwave oven will begin cooking.* Its syntax tree is shown in Figure 3. The semantics of words *START/RESUME* of category NN and *button* of category NN, retrieved from the lexicon, are $\lambda p.\lambda q.(p@q\ (\mathbf{pred}\ start\_resume\ q))$ and $\lambda x.(\mathbf{pred}\ button\ x)$, respectively. According to the as-rule NP($np :=$

$dt@(nn1@nn2)) \rightarrow \text{DT}(dt)\ \text{NN}(nn1)\ \text{NN}(nn2)$, the phrase *START/RESUME button* is converted to its semantics: $\lambda q.((\textbf{pred}\ button\ q)\ (\textbf{pred}\ start\_resume\ q))$. Applying the β-conversion on the syntax tree in Figure 3 from the bottom up to the root, the semantics of the sentence is gained and expressed as the following HRDS:

$$
\begin{aligned}
&(\textbf{cause}\ (\textbf{fluent}\ (\textbf{pred}\ press\ l_1\ robot\ Y_1)\\
&\qquad (\textbf{conds}\ (\textbf{pred}\ at\_time\ l_1\ t)\ (\textbf{pred}\ button\ Y_1)\ (\textbf{pred}\ start\_resume\ Y_1)))\\
&\quad (\textbf{fluent}\ (\textbf{pred}\ begin\ l_2\ X_2\ Y_2)\\
&\qquad (\textbf{conds}\ (\textbf{pred}\ at\_time\ l_2\ t+1)\ (\textbf{pred}\ oven\ X_2)\ (\textbf{pred}\ microwave\ X_2)\\
&\qquad\quad (\textbf{pred}\ cooking\ Y_2))))
\end{aligned}
$$

It expresses the causation between two fluents: $(\textbf{pred}\ press\ l_1\ robot\ Y_1)$ and $(\textbf{pred}\ begin\ l_2\ X_2\ Y_2)$. The predicates in the field marked by **conds** of the fluent $press$, as well as $begin$, are the conditions which the terms (e.g., l_1 and Y_1) in the fluent should satisfy when the fluent is valid. For example, the fluent $(\textbf{pred}\ press\ l_1\ robot\ Y_1)$ holds in the conditions that Y_1 is a *button* and it functions to *start* or to *resume* the running of a microwave.

3.2 Semantic interpretation of OMICS

In the OMICS project (Gupta & Kochenderfer, 2004), an extensive collection of knowledge is collected from Internet users, in order to enhance the capability of autonomously accomplishing tasks for indoor robots. The knowledge was input into sentence templates by users, censored by administrators, and then converted into and stored as tuples, of which most elements are English phrases. There are 48 tables in OMICS at present capturing different sorts of knowledge, including *Help* table (each tuple mapping a user desire to a task that may meet it), *Tasks* table (containing the names of tasks input by internet users), *Steps* table (each tuple decomposing a task into steps). Since some of the tables are related, some pieces of knowledge in OMICS can be represented as tuples in a joint table generated by a SQL query. The elements of such a tuple are semantically related according to the corresponding sentence template. Therefore, we introduce the *semantically-augmented rules*, one for each tuple sort, to capture the semantic information of tuples. Some semantically-augmented rules are listed below:

$$
\begin{aligned}
&\text{Location}(loc := (\textbf{state}\ (\textbf{fluent}\ (\textbf{pred}\ in\ X\ Y)\ (\textbf{conds}\ obj@X\ room@Y))))\\
&\qquad \rightarrow \text{Object}(obj)\ \text{Room}(room)\\
&\text{TaksSteps}(taksteps := (\textbf{dec}\ task\ step_0)) \rightarrow \text{Task}(task)\ \text{Step}(step_0)\\
&\text{TaksSteps}(taksteps := (\textbf{dec}\ task\ (\textbf{seq}\ step_0\ step_1))) \rightarrow \text{Task}(task)\ \text{Step}(step_0)\ \text{Step}(step_1)\\
&\text{Task}(task := vp) \rightarrow \text{VP}(vp)\\
&\text{Step}(step := vp) \rightarrow \text{VP}(vp)
\end{aligned}
$$

The semantic interpretation of OMICS will be demonstrated with the following example. The *Tasks* table and the *Steps* table combine to produce a joint table *Tasks/Steps*, where each tuple specifies steps of a task and also the definition of the task (Table 2). Accordingly, the semantically-augmented rule

$$
\begin{aligned}
&\text{TaksSteps}(taksteps := (\textbf{dec}\ task\ (\textbf{seq}\ step_0\ step_1\ step_2\ step_3)))\\
&\qquad \rightarrow \text{Task}(task)\ \text{Step}(step_0)\ \text{Step}(step_1)\ \text{Step}(step_2)\ \text{Step}(step_3) \qquad (1)
\end{aligned}
$$

Table 2: A part of *Tasks/Steps* table

$task$	$stepnum$	$step$
fetch an object	0	locate the object
fetch an object	1	go to the object
fetch an object	2	take the object
fetch an object	3	go back to where you were

Table 3: The semantic interpretation of tuple elements

Tuple Element	**Category**	**Semantics**
fetch an object	Task	$(\mathbf{task}\ fetch\ X\ (\mathbf{conds}\ (\mathbf{pred}\ object\ X)))$
locate the object	Step	$(\mathbf{task}\ locate\ X\ (\mathbf{conds}\ (\mathbf{pred}\ object\ X)))$
go to the object	Step	$(\mathbf{task}\ go\ X\ (\mathbf{conds}\ (\mathbf{pred}\ object\ X)))$
take the object	Step	$(\mathbf{task}\ take\ X\ (\mathbf{conds}\ (\mathbf{pred}\ object\ X)))$
go back to where you were	Step	$(\mathbf{task}\ go\_back\ X\ (\mathbf{conds}\ (\mathbf{pred}\ location\ X)))$

defines the semantics of the tuple.

The semantic interpretation procedure works in a bottom-up fashion. The tuple elements are first interpreted as shown in the table 3. Following the rule (1), the semantics of tuple elements are combined together piece by piece. Then we get the HRDS representation of the task *fetch an object*:

$$
\begin{array}{l}
(\mathbf{dec}\ (\mathbf{task}\ fetch\ X_1\ (\mathbf{conds}\ (\mathbf{pred}\ object\ X_1))) \\
\quad (\mathbf{seq}\ (\mathbf{task}\ locate\ X_2\ (\mathbf{conds}\ (\mathbf{pred}\ object\ X_2))) \\
\qquad (\mathbf{task}\ go\ X\ (\mathbf{conds}\ (\mathbf{pred}\ object\ X))) \\
\qquad (\mathbf{task}\ take\ X\ (\mathbf{conds}\ (\mathbf{pred}\ object\ X))) \\
\qquad (\mathbf{task}\ go\_back\ X\ (\mathbf{conds}\ (\mathbf{pred}\ location\ X)))))
\end{array}
$$

4. Integrated Decision-making

In the KeJia project, the integrated decision-making module is implemented using Answer Set Programming (ASP), a logic programming language with Prolog-like syntax under stable model semantics originally proposed in (Gelfond & Lifschitz, 1988). The module implements a growing model $M = \langle A, C^*, P^*, F^* \rangle$, the integrated decision-making mechanism, as well as some auxiliary mechanisms as an ASP program M^Π. The integrated decision-making in M is then reduced to computing answer sets of M^Π through some ASP solver. When the robot's multi-mode NLP module extracts a new piece of knowledge and stores it into M, it will be transformed further into ASP-rules and added into the corresponding part of M^Π.

4.1 Representing growing models in ASP

Given any growing model $M = \langle A, C^*, P^*, F^* \rangle$, all the components, A, C^*, P^* and F^* can be represented in ASP with following conventions. Firstly, the underlying language includes three pairwise-disjoint symbol sets: a set of *action* names, a set of *fluent* names,

and a set of *time* names. The atoms of the language are expressions of the form $occurs(a,t)$ or $true(f,t)$, where a, f, and t are action, fluent, and time name, respectively. Intuitively, $occurs(a,t)$ is true if and only if the action a occurs at time t, and $true(f,t)$ is true if and only if the fluent f holds at time t. Based on these conventions, an ASP-rule is of the form:

$$H \leftarrow p_1, \ldots, p_k, not\ q_1, \ldots, not\ q_m.$$

where p_i, $1 \leq i \leq k$, and q_j, $1 \leq j \leq m$, are literals, and H is either empty or an literal. A *literal* is a formula of the form p or $\neg p$, where p is an atom. If H is empty, then this rule is also called a *constraint*. An ASP-rule consisting of only H is called an *ASP-fact*. An ASP program is a finite set of ASP-rules. There are two kinds of negation in ASP, the classical negation $\neg$ and non-classical negation not. Roughly, $not\ q$ means that q is not derivable from the ASP program. Similarly, a constraint that $\leftarrow p_1, \ldots, p_k$ specifies that $p_1, \ldots, p_k$ are not derivable jointly the ASP program. We take the action *grasp* for example to show how primitive actions are represented as ASP-rules. The related fluents are

- $grasp(X)$: the action of griping the object X and picking it up.
- $holding(X)$: the fluent that the object X is held in the grip of the robot.
- $on(X,Y)$: the fluent that the object X is on the object Y.

The effect of executing $grasp(X)$ is $holding(X)$ and described by following ASP-rules:

$$\begin{aligned} true(holding(X), t+1) &\leftarrow occurs(grasp(X), t), \\ \neg true(on(X,Y), t+1) &\leftarrow occurs(grasp(X), t), true(on(X,Y), t). \end{aligned}$$

And the precondition of $grasp(X)$ is $not\ holding(Y)$ for any Y, i.e., the grip holds nothing, which is described in ASP as a constraint

$$\leftarrow occurs(grasp(X), t), true(holding(Y), t).$$

Other primitive actions are represented as ASP-rules similarly. In addition, the occurrence of any primitive action is forced to conform to the following restrictions

$$\begin{aligned} occurs(grasp(X), t) &\leftarrow not\ \neg occurs(grasp(X), t), \\ \neg occurs(grasp(X), t) &\leftarrow not\ occurs(grasp(X), t). \end{aligned}$$

Each element of P^* decomposes a task into some sub-tasks or actions. For the general case, see the details in Section 4.2. When a task T is decomposed to an action sequence $\langle a_1, a_2, \ldots, a_n \rangle$, we add an ASP-rule into the ASP program

$$process(T, t, t') \leftarrow occurs(a_1, t), occurs(a_2, t+1), \ldots, occurs(a_n, t+n),\ t' = t+n.$$

where $process(T,t,t')$ denotes that the task T is accomplished during time t to t'. Accordingly, the definitions of $process(T,t,t')$ are also included in the ASP program. Similarly, for each element of F^* designating a set of literals $\{l_1, \ldots, l_m\}$ to a task T, we add an ASP-rule

$$process(T, t, t') \leftarrow true(l_1, t'), \ldots, true(l_m, t'),\ t < t'$$

into the ASP program. It is the case of C^* elements, which are transformed into ASP-rules similarly. Moreover, we use

$$\begin{aligned} &\leftarrow true(holding(X), t), true(falling(X), t), \\ true(falling(X), t) &\leftarrow true(on(X,Y), t), true(falling(Y), t). \end{aligned}$$

to specify that $falling(X)$ is an indirect effect of some action that causes $falling(Y)$ while X is on Y. The *frame problem* is resolved by "inertia lows" of the form

$$true(\sigma, t+1) \leftarrow true(\sigma, t), not\ \neg true(\sigma, t+1),$$
$$\neg true(\sigma, t+1) \leftarrow \neg true(\sigma, t), not\ true(\sigma, t+1).$$

where σ is a meta-variable ranging over fluent names. The inertia lows guarantee the minimal change condition. In the Situation Calculus (Reiter, 2001), the successor state axioms are used to "solve" the frame problem. According to this solution, the designer must enumerate as effect axioms all the ways in which the value of a particular fluent can be changed, and add axioms to capture the assumption that "these effect axioms characterize all the conditions under which an action causes a fluent to become true (respectively, false) in the successor situation". Once, such an effect axiom is modified then the corresponding successor state axiom needs also to be modified. However, in the ASP program, the effect axiom is the only rule that needs to be modified, while others remain unchanged. The initial state (at time 0) of the environment can be expressed in facts of the form $true(\sigma, 0)$.

4.2 Integrated decision-making in ASP

Since any ASP solver innately possesses a general-purpose goal-directed planning schema, we embed a general-purpose task-directed action selection schema into the existing schema, so that the augmentation becomes a general-purpose decision-making mechanism that integrates both schemas and guarantees the executability of every plan it generates. Technically, the augmentation is built on the basis of M^{Π}.

First of all, we name a class of entities called "sequence" this way: (i) an action a is a sequence; (ii) a task T is a sequence; and (iii) if p_i $(1 \leq i \leq m)$ are sequences, then $p_1; \ldots; p_m$ is a sequence. Let $\tau = \langle s_0, a_0, s_1, \ldots, a_{n-1}, s_n \rangle$ be any trajectory. That τ *satisfies a sequence* p is defined recursively as follows:

(1) If $p = a$, where a is an action, then $a_0 = a$;

(2) If $p = T$, where T is a task such that there is a HRDS rule in P^* that decomposes T into a sequence of sub-tasks, then τ satisfies this sequence of sub-tasks; or T is a task such that it is designated a set of literals in F^*, then this set is a subset of the state s_n;

(3) If $p = p_1; \ldots; p_m$, where pi $(1 \leq i \leq m)$ are sequences, then there exists $0 \leq n^1 \leq n^2 \leq \cdots \leq n^{m-1} \leq n$ such that:

– the trajectory $\langle s_0, a_0, \ldots, s_{n^1} \rangle$ satisfies p_1;

– the trajectory $\langle s_{n^1}, a_{n^1}, \ldots, s_{n^2} \rangle$ satisfies p_2;

– $\cdots$;

– the trajectory $\langle s_{n^{m-1}}, a_{n^{m-1}}, \ldots, s_n \rangle$ satisfies p_n.

According to the definitions above, if a trajectory $\langle s_0, a_0, s_1, \ldots, a_{n-1}, s_n \rangle$ satisfies a sequence $a; a'$ where a and a' are actions, and a' is not executable in s_1, then $a_0 = a$ and there exists a state s_m $(1 \leq m \leq n)$ such that s_i satisfies the preconditions of a' and $a_m = a'$. In other words, the sub-trajectory $\langle s_1, a_1, \ldots, s_m \rangle$ fills up the "gap" between a and a'.

A sequence s specifies how to complete a task T step by step. If a trajectory contains a sub-trajectory which satisfies s, then the corresponding task T is also completed in this trajectory. Now we consider how to specify a procedure s in ASP. Given an growing model M, we want to obtain a set of ASP-rules of s, Π_s, such that a trajectory $\langle s_0, a_0, s_1, \ldots, a_{n-1}, s_n \rangle$ satisfies both M and s if and only if $\{true(\sigma, i) | \sigma \in s_i, 0 \leq i \leq n\} \cup \{\neg true(\sigma, i) | \neg\sigma \in s_i, 0 \leq i \leq n\} \cup \{occurs(a_i, i) | 0 \leq i \leq n-1\}$ is an answer set of $M^{\Pi} \cup \Pi_s$.

Any sequence defined above can be specified in HRDS as $\langle sequence \rangle$, see Appendix A for details. Given such a sequence S, we define the set Π_s of ASP-rules recursively as follows (where $t, t', t_1, \ldots, t_{m-1}$ are meta-variables ranging over time):

(1) If $S = (\mathbf{act}\ name\_a\ X_1 \ldots X_m\ (\mathbf{conds}\ (\mathbf{pred}\ cond_1\ X_1 \ldots X_m) \ldots (\mathbf{pred}\ cond_n\ X_1 \ldots X_m)))$ where $name\_a$ is an action name, $X_1 \ldots X_m$ are its parameters, and predicates $cond_1(X_1, \ldots, X_m), \ldots, cond_n(X_1, \ldots, X_m)$ represent domains of these parameters, then Π_S is the set of following ASP-rules

$$complete(p, t, t+1) \leftarrow occurs(name\_a(X_1, \ldots, X_m), t), \\ true(cond_1(X_1, \ldots, X_m), t), \ldots, true(cond_n(X_1, \ldots, X_m), t).$$

(2) If $S = (\mathbf{task}\ name\_t\ X_1 \ldots X_m\ (\mathbf{conds}\ (\mathbf{pred}\ cond_1\ X_1 \ldots X_m) \ldots (\mathbf{pred}\ cond_n\ X_1 \ldots X_m)))$ where $name\_t$ is an task name, $X_1 \ldots X_m$ are its parameters, and predicates $cond_1(X_1, \ldots, X_m), \ldots, cond_n(X_1, \ldots, X_m)$ represent domains of these parameters, then Π_S contains

$$complete(p, t, t') \leftarrow complete(name\_t(X_1, \ldots, X_m), t, t'), \\ true(cond_1(X_1, \ldots, X_m), t), \ldots, true(cond_n(X_1, \ldots, X_m), t).$$

(3) If P^* designates a sequence S' to accomplish the task $name\_t(X_1, \ldots, X_m)$, then Π_S also contains $\Pi_{S'}$ and

$$complete(name\_t(X_1, \ldots, X_m), t, t') \leftarrow complete(S', t, t').$$

(4) If F^* designates a set of literals $\{\sigma_1, \ldots, \sigma_o, \neg\sigma_{o+1}, \ldots, \neg\sigma_m\}$ for the task, then Π_S also contains

$$complete(name\_t(X_1, \ldots, X_m), t, t') \leftarrow true(\sigma_1, t'), \ldots, true(\sigma_o, t'), \\ \neg true(\sigma_{o+1}, t'), \ldots, \neg true(\sigma_m, t'),\ t < t'.$$

(5) If $S = (\mathbf{seq}\ S_1 \ldots S_m)$, where S_i $(1 \leq i \leq m)$ are sequences, Π_S contains $\Pi_{S_1}, \ldots, \Pi_{S_m}$ and

$$process(S; t; t') \leftarrow process(S_1, t, t_1), process(S_2, t_1, t_2), \ldots, process(S_m, t_{m-1}, t').$$

5. Case-studies

We have conducted a variety of case studies of open knowledge enabling on KeJia robots. In a case study (Chen et al., 2010), we tested the KeJia robot's ability of acquiring causality knowledge through human-robot dialogue. A testing instance is shown in Figure 4. There was a board putting on the edge of a table, with one end sticking out. A red can was put on the sticking out end of the board, and a green can was on the other end. The task for KeJia is to pick up the green can under a default presupposition of avoiding anything falling. We took a version of KeJia without any built-in knowledge about "balance", "fall", or any other equivalents. A human told the robot "*An object will fall if it is on the sticking out end of a board and there is nothing on the other end of the board*". KeJia's

NLP module extracted the knowledge and transformed it into an ASP-rule, and with the rule the decision-making module generated a plan, in which the robot moved the red can to the table first and then picked up the green one. The robot accomplished the task by executing this plan[2]. It is worthwhile emphasizing the fact that KeJia could not have accomplished the task without the causality knowledge acquired. This indicates that KeJia's ability is substantially enhanced by knowledge acquisition through human-robot dialogue. In another case study (Xie et al., 2012), a user requested for heating up some popcorn with a microwave oven, while the robot had not known the functions of the buttons on the control panel before the experiment. The robot extracted knowledge of the buttons' function from the manual in English (Section 3.1 shows the translation of one of the sentences from the manual). With the extracted knowledge, the robot generated a plan consisting of 22 primitive actions in 3.4 seconds. It executed the plan and accomplished the task in 11.3 minutes[3].

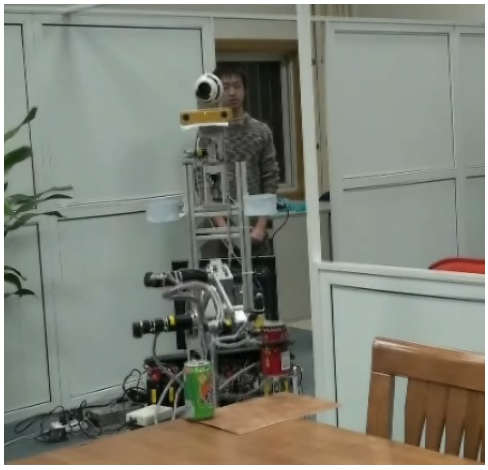

Figure 4. Testing instance of acquiring causality knowledge through human-robot dialogue

In this paper, we further report first system evaluation of the proposed techniques through two tests on two large task sets collected from OMICS. Meanwhile, we took a version of the OK-KeJia robot that contains only two sorts of built-in knowledge. One is the action models, and the other one is the semantically annotated lexemes of words in the lexicon, a type of linguistic knowledge (see Section 3.1) which is only used in the multi-mode NLP module. The open knowledge the robot could get in the experiments was limited to two tables of OMICS and the synonymies of *WordNet*[4], without any handcrafted knowledge.

Each test varies in two dimensions, the action model and the open knowledge base. Five action models, $AM_1 = \{move\}$, $AM_2 = \{move, find\}$, $AM_3 = \{move, find, pick\_up\}$, $AM_4 = \{move, find, pick\_up, put\_down\}$ and $AM_5 = \{move, find, pick\_up, put\_down, open, close\}$, were chosen in order to examine the impact of the different action capabilities of a robot on its overall performance. Each test

[2]Demo video available on http://ai.ustc.edu.cn/en/demo/Cause_Teach.php

[3]Demo video available on http://ai.ustc.edu.cn/en/demo/ServiceRobot_oven.php

[4]http://wordnet.princeton.edu/wordnet/

consisted of three rounds with different open knowledge bases, in order to show the impact of open knowledge on performance.

The task set in Test 1 is defined as follows. There are 11,615 different tuples in *Tasks/Steps*, each consisting of a task name T and a sequence of steps (sub-tasks), s. We take s as the definition of T. For example, there are two tuples: <*help someone carry something, 0. pick up the item, 1. walk with the item to where the person needs it*> and <*help someone carry something, 0. get object, 1. follow person*>. Obviously, these two tuples with the same task name define two different tasks, because there is no guarantee that any action sequence that fulfill one of the two tasks must fulfill the other one. Therefore, we collected all the 11,615 task definitions in the task set for Test 1.

In the first-round of Test 1, only the action models were used in the planning procedure with no open knowledge (i.e., $C^* = P^* = F^* = \emptyset$). Every task in the task set was input into the robot and the robot tried to fulfill it. A task is solvable by the robot's planner if and only if each step of the task is a primitive action. In the second-round, only the 11,615 *Tasks/Steps* tuples were used as a sort of procedural knowledge, now taken as task-decomposition rules (td-rules, for short). Hence $C^* = F^* = \emptyset$ and $P^* \subseteq$ *Tasks/Steps*. It follows that a task can be fulfilled if all of its steps can be reduced to primitive actions through some td-rules. In the third-round, the robot tried to get open knowledge from both *Tasks/Steps* and WordNet (i.e., $P^* \subseteq$ *Tasks/Steps*, $C^* \subseteq$ WordNet synonymies, and $F^* = \emptyset$). Consequently, "*move to a location*" and "*go to a location*" were identified as equivalent and executable by the robot, although only the former can be matched to the robot's primitive action. Obviously, the open knowledge used in this test is extremely sparse—only the definitions of the tasks were used as the procedural knowledge.

A set of algorithms were developed. Algorithm 1 is only for understanding the main ideas of the actual algorithm for the second-round experiment of Test 1. In the iteration, a new td-rule from the *Tasks/Steps* table is selected, processed by the multi-mode NLP module, and added into the growing model. Then the planner is called to compute an action sequence for the user task. For the third-round, synonymies of WordNet are used to substitute concepts appeared in td-rules with their equivalent primitive actions. The planner $taskPlan$ in the algorithm is a simplified version of the integrated decision-making module due to some reasons. Most importantly, since this case study was conducted on large task sets collected from OMICS, we did not succeed in associating with each task an observation, which were obtained by our real robots in other case studies mention above. Consequently, the executability condition was simplified as "consisting of primitive actions".

The task set in Test 2 consists of 467 different desires appeared in the *Help* table of OMICS, with duplicate ones discarded. Some example *Help* tuples are <*are sick, by giving medicine*>, <*are cold, by making hot tea*>, and <*feel thirsty, by offering drink*>. In each tuple, the first element is taken as a user desire, while the second element, a task, is taken as a means to meet the desire, not as the definition of the desire. In the first-round of Test 2, no open knowledge was used. In the second-round, the 3,405 unduplicated tuples in *Help* were taken as functional knowledge and *Tasks/Steps* as procedural knowledge (i.e., $F^* \subseteq$ *Help*, $P^* \subseteq$ *Tasks/Steps*, and $C^* = \emptyset$). WordNet synonymies were used in the third-round.

The experimental results are shown in Table 4. On every action model in each round, the number of tasks or desires that were fulfilled by the robot is listed in the table. In addition, the percentages of fulfilled tasks or desires with respect to the size of the task sets on AM_5 are listed in the last column. We make following observations.

(1) **The overall performance increases remarkably due to the use of a moderate**

Algorithm 1 $getActionSequence$(phrase ph)

```
1: /* generate an action sequence for task ph */
2: p := parsePredicates(ph)
   /* semantically parses ph to internal representation p */
3: Subtask := subTask(p)
   /* initiate Subtask with sub-tasks of p */
4: P+ := P*
   /* initiate P+ with the action model P* */
5: Res := taskPlan(p)
   /* taskPlan returns an action sequence computed by the integrated decision-making
   module */
6: if Res ≠ null then
7:    return Res
8: end if
9: while there is a new tuple t from Tasks/Steps that matches an element of Subtask do
10:   q := parsePredicates(t)
11:   Subtask := subTask(q)
12:   P+ := P+ ∪ q
13:   Res := taskPlan(p)
14:   if Res ≠ null then
15:      return Res
16:   end if
17: end while
18: return Failure
```

Table 4: Experimental results

Open Knowledge	AM_1	AM_2	AM_3	AM_4	AM_5	Percentage on AM_5
Test 1 (11,615 user tasks)						
Null	6	24	45	164	207	1.78%
Tasks/Steps (11,615 rules)	7	28	51	174	219	1.89%
Tasks/Steps+WordNet	16	43	71	233	297	2.56%
Test 2 (467 user desires)						
Null	0	1	1	4	4	0.86%
Help+*Tasks/Steps* (15,020 rules)	29	63	83	107	117	25.05%
Help+*Tasks/Steps*+WordNet	43	73	87	119	134	28.69%

amount of open knowledge. The percentage of fulfilled tasks increases from 1.78% to 2.56% in Test 1 with very sparse open knowledge of two types, and from 0.86% to 28.69% in Test 2 with a moderate amount of open knowledge of three types, respectively. The difference in the performance improvements in the two tests further reveals the significant function of the amount or "density" of the open knowledge.

(2) **General-purpose techniques play an essential role in supporting the use of open knowledge**. It is worthwhile emphasizing that the improvements were made through us-

ing open knowledge by some general-purpose mechanisms (multi-mode NLP, integrated decision-making and open knowledge searching), without any manual assistance. Actually, it is difficult to utilize manual assistance in case of large task sets.

(3) **A robot's basic ability (primitive actions) is still a key factor for the overall performances**. As expected, the robot met more user requests with more primitive actions in all the cases. More detailed examination indicates that there are lots of user requests which are not met just due to the powerlessness of the action models used in the experiments, though KeJia robots can do more actions that are not contained in the action models.

6. Conclusions

Along with remarkable progress in HRI-related areas, it is interesting to consider the possibility of improving the performance of robots using open knowledge autonomously, instead of handcraft coding of knowledge. The KeJia project is a long-term effort toward this goal. We focus on three issues in this paper: (i) How does a robot "understand" and extract open knowledge in unstructured and semi-structured natural language; (ii) How does a robot make use of different types of knowledge in decision-making in order to meet user requests; (iii) How can a robot be aware of what knowledge it lacks for a given task and search for the missing knowledge from open-source resources. A set of techniques are proposed, including mechanisms for multi-mode NLP, integrated decision-making and open knowledge searching, and the OK-KeJia robot prototype is developed and tested in a variety of case studies. In particular, first system evaluation is conducted with two large task sets, consisting of 11,615 user tasks and 467 user desires collected from OMICS, respectively. Different action models and multiple types of open knowledge from OMICS and WordNet are used in the evaluation. The experiments show that overall performances will increase remarkably due to the use of appropriate open knowledge. In the test with 11,615 user tasks and very sparse open knowledge of two types, the percentage of fulfilled tasks increases only from 1.78% to 2.56%. In the test with 467 user desires and a moderate amount of open knowledge of three types, the percentage increases remarkably from 0.86% to 28.69%. Considering that only a very small proportion of knowledge in OMICS (i.e., 2 tables from the 48 in total) was used in this case study, there would be some room for further progress.

Many challenges remain in the current work. The tests with large task sets conducted so far did not involve the context of HRI, where sometimes a user request can only be understood together with the observation of the scenario. This leads to an investigation into the connection of open knowledge and contexts of HRI. How to make use of a larger amount of open knowledge is also an interesting issue, especially when context is considered. We believe these two issues are related. They may demand more on techniques for multi-mode NLP, integrated decision-making, and identification of knowledge gaps. Another question is about the relation between open knowledge and action model learning. Integration of both techniques may help improve the performance of HRI further.

Acknowledgements

There are about 30 Chinese professors and graduate students working on the KeJia Project, which has been supported by the National Hi-Tech Project of China under grant 2008AA01Z150 and the Natural Science Foundation of China under grant 60745002 and 61175057, as well as the USTC Key Direction Project and the USTC 985 Project. The authors are grateful to the other team members, the sponsors, and the following persons for

helpful discussions about research issues in the project with them: Michael Gelfond, David Hsu, Fengzhen Lin, Daniele Nardi, Peter Stone, Manuela Veloso, Mary-Anne Williams, Yan Zhang and Shlomo Zilberstein. We also thank the anonymous reviewers for their valuable comments and suggestions on the manuscript of this work.

References

Blackburn, P., & Bos, J. (2005). *Representation and inference for natural language: A first course in computational semantics*. Center for the Study of Language and Information.

Burgard, W., Cremers, A., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., et al. (1999). Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, *114*(1-2), 3–55.

Cantrell, R., Talamadupula, K., Schermerhorn, P., Benton, J., Kambhampati, S., & Scheutz, M. (2012). Tell me when and why to do it!: run-time planner model updates via natural language instruction. In *Proceedings of the 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI-12)* (pp. 471–478). (To appear)

Chen, X., Ji, J., Jiang, J., Jin, G., Wang, F., & Xie, J. (2010). Developing high-level cognitive functions for service robots. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)* (pp. 989–996).

Chen, X., Sui, Z., & Ji, J. (2012). Towards metareasoning for human-robot interaction. In *Proceedings of the 12th International Conference on Intelligent Autonomous System (IAS-12)*.

Doshi, F., & Roy, N. (2007). Efficient model learning for dialog management. In *Proceedings of the 2nd ACM/IEEE International Conference on Human Robot Interaction (HRI-07)* (pp. 65–72).

Fong, T., Thorpe, C., & Baur, C. (2003). Robot, asker of questions. *Robotics and Autonomous Systems*, *42*(3), 235–243.

Ge, R., & Mooney, R. (2009). Learning a compositional semantic parser using an existing syntactic parser. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP-09)* (pp. 611–619).

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming (ICLP-88)* (pp. 1070–1080).

Gupta, R., & Kochenderfer, M. (2004). Common sense data acquisition for indoor mobile robots. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)* (pp. 605–610).

Kaupp, T., Makarenko, A., & Durrant-Whyte, H. (2010). Human-robot communication for collaborative decision making–a probabilistic approach. *Robotics and Autonomous Systems*, *58*(5), 444–456.

Klein, D., & Manning, C. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics (ACL-03)* (pp. 423–430).

Kruijff, G., Lison, P., Benjamin, T., Jacobsson, H., Zender, H., Kruijff-Korbayová, I., et al. (2010). Situated dialogue processing for human-robot interaction. *Cognitive Systems*, 311–364.

Lemaignan, S., Ros, R., Sisbot, E., Alami, R., & Beetz, M. (2012). Grounding the interaction: Anchoring situated discourse in everyday human-robot interaction. *International Journal of Social Robotics*, 1–19.

Levesque, H., Reiter, R., Lesperance, Y., Lin, F., & Scherl, R. (1997). Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, *31*(1-3), 59–83.

Reiter, R. (2001). *Knowledge in action: logical foundations for specifying and implementing dynamical systems* (Vol. 16). Cambridge Univ Press.

Rosenthal, S., Biswas, J., & Veloso, M. (2010). An effective personal mobile robot agent through symbiotic human-robot interaction. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)* (pp. 915–922).

Rosenthal, S., Veloso, M., & Dey, A. (2011). Learning accuracy and availability of humans who help mobile robots. In *Proceedings of the 25th Conference on Artificial Intelligence (AAAI-11)* (pp. 60–74).

Talamadupula, K., Benton, J., Kambhampati, S., Schermerhorn, P., & Scheutz, M. (2010). Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *1*(2), 14.

Tenorth, M., & Beetz, M. (2009). KnowRob-knowledge processing for autonomous personal robots. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-09)* (pp. 4261–4266).

Thrun, S. (2004). Toward a framework for human-robot interaction. *Human–Computer Interaction*, *19*(1-2), 9–24.

Xie, J., Chen, X., Ji, J., & Sui, Z. (2012). Multi-mode natural language processing for extracting open knowledge. In *Proceedings of the 2012 IEEE/WIC/ACM International Conferences on Intelligent Agent Technology (IAT-12)*. (To appear)

Appendix A

The BNF definition for the syntax of Human-Robot Dialogue Structure

$\langle variable \rangle ::=$ a string where the first character is upper-case, e.g., X

$\langle constant \rangle ::=$ a string where the first character is lower-case or a underline, e.g., a, $\_b$, etc.

$\langle term \rangle ::= \langle constant \rangle \mid \langle variable \rangle$

$\langle predicate \rangle ::= (\textbf{pred}\ \langle name \rangle\ \langle term \rangle^+)$

$\langle fluent \rangle ::= (\textbf{fluent}\ \langle predicate \rangle\ (\textbf{conds}\ \langle predicate \rangle^+))$

$\langle formula \rangle ::= \langle fluent \rangle \mid (\textbf{neg}\ \langle formula \rangle) \mid (\textbf{conj}\ \langle formula \rangle^+) \mid (\textbf{disj}\ \langle formula \rangle^+)$

$\langle action \rangle ::= (\textbf{act}\ \langle action\_name \rangle\ \langle term \rangle^+\ (\textbf{conds}\ \langle predicate \rangle^+))$

$\langle task \rangle ::= (\textbf{task}\ \langle task\_name \rangle\ \langle term \rangle^+\ (\textbf{conds}\ \langle predicate \rangle^+))$

$\langle sequence \rangle ::= \langle action \rangle \mid \langle task \rangle \mid (\textbf{seq}\ \langle sequence \rangle^+)$

$\langle causation \rangle ::= (\textbf{cause}\ \langle fluent \rangle\ \langle fluent \rangle)$

$\langle decomposition \rangle ::= (\textbf{dec}\ \langle task \rangle\ \langle sequence \rangle)$

$\langle effect \rangle ::= (\textbf{eff}\ \langle task \rangle\ \langle formula \rangle)$

$\langle statement \rangle ::= (\textbf{state}\ \langle formula \rangle)$

$\langle request \rangle ::= (\textbf{req}\ \langle task \rangle)$