Adapting knowledge-level planning for natural language dialogue

Ron Petrick
School of Informatics
University of Edinburgh
Edinburgh, Scotland, UK
rpetrick@inf.ed.ac.uk

University of Strathclyde, Glasgow, Scotland, UK

19 March 2010
Travelling to Glasgow (Scenario I)

“I want to take the train from Edinburgh Waverley to Glasgow Queen Street Station.”

Go to the station, buy a ticket, check the departure board for track information, go to the track, board the train, . . . , enjoy Glasgow!
“I want to take the train from Edinburgh Waverley to Glasgow Queen Street Station.”

Go to the station, buy a ticket, ask someone for track information, go to the track, board the train, . . . , enjoy Glasgow!
### Two plans

<table>
<thead>
<tr>
<th>Scenario I</th>
<th>Scenario II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to the station</td>
<td>Go to the station</td>
</tr>
<tr>
<td>Buy a ticket</td>
<td>Buy a ticket</td>
</tr>
<tr>
<td>Check departure board</td>
<td>Ask someone for information</td>
</tr>
<tr>
<td>Go to the track</td>
<td>Go to the track</td>
</tr>
<tr>
<td>Board the train</td>
<td>Board the train</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Enjoy Glasgow!</td>
<td>Enjoy Glasgow!</td>
</tr>
</tbody>
</table>

**Observational action step**  **Dialogue step (speech act)**

⇒ Both actions serve as information gathering steps in the plan.

⇒ Can we reason about dialogue acts in the same way as “ordinary” actions? Can we use the same machinery for planning?
Outline

1. Natural language dialogue systems
2. STRIPS planning
3. Planning with Knowledge and Sensing (PKS)
4. Extending PKS for dialogue planning
5. Dialogue planning in PACO-PLUS

Message

⇒ The same mechanisms used for ordinary task planning can often be applied to dialogue planning.
⇒ Speech acts can be modelled as planning actions, and plans generated by reasoning about the knowledge state.
⇒ Modern planning techniques offer potential solutions to challenging problems in natural language.
⇒ Problems in natural language offer challenging benchmarks for automated planning research.
Natural language dialogue systems

• **Dialogue systems** are computer systems designed to carry out natural language conversations with human users.

• To do this, a dialogue system needs a rich model of its domain of operation, a description of the user’s (and its own) goals, and a model of the actions it has available to it.

• Deciding on the behaviour of a dialogue system involves choosing the actions that will change the user’s (or its own) state in order to bring about its goals.

• Dialogue systems have been studied in many challenging domains, e.g., tutorial systems (Dzikovska *et al.* 2008), systems that coordinate joint activities (Foster *et al.* 2009), and those that engage in information seeking dialogues (Rieser & Lemon 2008), among others.
Example: JAST robot (Foster et al. 2009)
Our target: PACO-PLUS ARMAR robot

Image: Asfour et al., Karlsruhe Institute of Technology

code: graspD-fromTable(obj2)

Image: Asfour et al., Karlsruhe Institute of Technology
Natural language and planning


• Early approaches suffered due to inefficient planning techniques.

• Recent work has tended to separate action planning and dialogue planning and has focused on specialized approaches, e.g., finite state machines, information state, rule-based approaches to speech act theories, dialogue games, . . .

• There has been a renewed interest in applying modern planning techniques to natural language problems, e.g., Koller & Stone (2007), Benotti (2008), Brenner & Kruijff-Korbayová (2008), Koller & Petrick (2008, 2010).
Examples: GIVE, text adventure games

(Koller et al. 2007; Koller & Petrick 2008, 2010)

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>trykey(x, y)</td>
<td>$K(\text{accessible}(x))$</td>
<td>$\text{add}(K_w, \text{fits_in}(x, y))$</td>
</tr>
<tr>
<td></td>
<td>$K(\text{locked}(x))$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$K(\text{key}(y))$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$K(\text{inventory_obj}(y))$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(Benotti 2007, 2008, 2009)

take(silver\_key, table),
take(golden\_key, table),
trykey(silver\_key, door),
branch(fits\_in(silver\_key, door))

$K^+$:
unlock(doors, silver\_key),
open(doors)

$K^-$:
unlock(doors, golden\_key),
open(doors).
This work: what about dialogue planning?

⇒ Recent approaches have mostly ignored the application of planning techniques to dialogue.
Automated planning

• Automated **planning** techniques are good at building goal-directed plans of action under many challenging conditions, given a suitable description of the domain.

• A **planning problem** consists of:
  1. A representation of the properties and objects in the world and/or the agent’s knowledge, usually described in a logical language,
  2. A set of state transforming actions,
  3. A description of the initial world/knowledge state,
  4. A set of goal conditions to be achieved.

• A **plan** is a sequence of actions that when applied to the initial state transforms the state in such a way that the resulting state satisfies the goal conditions.
### STRIPS (Fikes & Nilsson 1971)

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Add list</th>
<th>Delete list</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pickup(x)</code></td>
<td><code>handEmpty</code></td>
<td><code>holding(x)</code></td>
<td><code>handEmpty</code></td>
</tr>
<tr>
<td></td>
<td><code>onTable(x)</code></td>
<td></td>
<td><code>onTable(x)</code></td>
</tr>
<tr>
<td><code>dropInBox(x, y)</code></td>
<td><code>holding(x)</code></td>
<td><code>inBox(x, y)</code></td>
<td><code>holding(x)</code></td>
</tr>
<tr>
<td></td>
<td><code>box(y)</code></td>
<td><code>handEmpty</code></td>
<td><code>empty(y)</code></td>
</tr>
</tbody>
</table>

- **A world state** is represented by a **closed world** database \( \mathcal{D} \).
- An action’s **preconditions** specify the conditions under which an action can be applied, evaluated against \( \mathcal{D} \) (qualification problem).
- An action’s **effects** specify the changes the action makes to the world, applied by updating \( \mathcal{D} \) (and offer a solution to the **frame problem**).
Planning with STRIPS actions

- We can generate **plans** by chaining together fully instantiated STRIPS actions.
- Example: achieve a state where $\text{inBox}(o1,b1)$ holds.
- STRIPS forms the core of PDDL (McDermott *et al.*, 1998), the language of many modern planners and the International Planning Competition (see [http://ipc.icaps-conference.org/](http://ipc.icaps-conference.org/)).
Planning with incomplete information

- Problem: classical STRIPS planning assumes complete knowledge and deterministic action effects, which are not realistic in many domains (e.g., typical dialogue contexts).

- In general, an agent operating in a dynamic world must do so with incomplete information about its environment, and
  - Make decisions based on what it knows or believes,
  - Reason about the effects of its actions,
  - Gather information about the world (through sensing).

- This is certainly the case with dialogue.

\[\Rightarrow\] Reasoning about sensing requires the ability to reason effectively about the agent’s knowledge/beliefs.
Planning with Knowledge and Sensing

- PKS is a “knowledge-level” planner that builds plans based on what is known (Petrick & Bacchus 2002, 2004).

- Knowledge is updated in a STRIPS-like manner, however, actions are modelled by their effects on the planner’s knowledge state.

- Plans can be constructed with conditional branches to manage indefinite information (contingent planning).

- Representation supports non-propositional features like functions, run-time variables, and simple program structures.

- PKS has previously been applied to traditional benchmarks, robot systems, web services, operating system scenarios.
Representing knowledge in PKS

• PKS uses a collection of five databases, each of which is restricted to a particular types of knowledge: $K_f$, $K_v$, $K_w$, $K_x$, $LCW$.

• Knowledge is assumed to be correct but incomplete.

• The contents of the databases ($DB$) have a fixed formal translation to formulae in a modal logic of knowledge which formally defines the planner’s knowledge state ($KB$).

• Rather than modelling sets of possible worlds, the modal formulae directly represent the planner’s knowledge state.

• Planning: actions update $DB \Rightarrow$ update $KB$. 
PKS databases

- $K_f$: knowledge of positive and negative facts (but not closed world!)
  \[ p(c) \quad \neg q(b, c) \quad f(a) = c \quad g(b, c) \neq d \]

- $K_w$: knowledge of binary sensing effects
  \[ \phi : \text{the planner “knows whether” } \phi \]

- $K_v$: knowledge of function values, multi-valued sensing effects
  \[ f : \text{the planner “knows the value” of } f \]

- $K_x$: exclusive-or knowledge
  \[ (\ell_1 | \ell_2 | \ldots | \ell_n) : \text{exactly one of the } \ell_i \text{ must be true} \]

- $LCW$: local closed world information (Etzioni et al. 1994)
Reasoning in PKS

- A primitive query language is used to ask simple questions about the planner’s knowledge state:
  - $K(\alpha)$, is $\alpha$ known to be true?
  - $K_v(t)$, is the value of $t$ known?
  - $K_w(\alpha)$, is $\alpha$ known to be true or known to be false?
  - The negation of the above queries.

- A sound, but incomplete, inference procedure checks the database contents to determine the truth of a query.
### PKS actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>readPaper</td>
<td>$K(\text{havePaper})$</td>
<td>$\text{add}(K_v, \text{phoneNumber})$</td>
</tr>
</tbody>
</table>
| dial | $K_v(\text{phoneNumber})$ | $\text{add}(K_f, \text{dialledOk})$
| | | $\text{add}(K_w, \text{connected})$ |

- PKS actions are based on an extension of STRIPS.
- Easy to compute new knowledge states by forward chaining:
  - Evaluate preconditions against a set of databases $\mathbf{DB}$ ($\mathbf{KB}$),
  - Effects update $\mathbf{DB} \Rightarrow$ update $\mathbf{KB}$.
- Plans are generated by searching the space of database states.
PKS planning

- **Conditional branches** are formed from $K_w$ (and $K_v$) formulae.
- Actions can be parametrized with ground $K_v$ terms.
- Goal conditions must be satisfied along every plan branch.
Can we apply this approach to dialogue?
Dialogue planning with speech acts

• Motivation: some actions like *ask* and *tell* can be modelling as sensing actions. We can model certain dialogue problems as instances of planning with incomplete information and sensing.

• Can we apply knowledge-level planning techniques to this problem?
  – Dialogues involve multiple participants.
  – Actions correspond to *speech acts*, e.g., *ask*, *tell*, . . . .
  – Plans specify mixed-initiative discourse among participants.


⇒ What kinds of changes do we need to make to PKS?
How tractable is the reasoning?
In what kinds of domains can we apply these techniques?
Participants and common ground

- We use labels (modalities) for referencing dialogue participants and common ground:

  \[
  \begin{align*}
  \text{[S]} & \quad \text{Speaker supposition} \\
  \text{[H]} & \quad \text{Hearer supposition} \\
  \text{[X], [Y], \ldots} & \quad \text{Other participant/agent suppositions} \\
  \text{[C}_{XY}] & \quad \text{Common ground between X and Y}
  \end{align*}
  \]

Examples

| \text{[S]} p | “The speaker supposes } p.” |
| \text{[S]} [H] p | “The speaker supposes the hearer supposes } p.” |
| \text{[H]} [C}_{SH} [S] p | “The hearer supposes it’s common ground between the speaker and hearer that the speaker supposes } p.” |
Knowledge assertions

• Use restricted PKS knowledge assertions as the basis (content) of knowledge expressions:

\[ K_p \] “Know \( p \)”
\[ K_v t \] “Know the value of \( t \)”
\[ K_w p \] “Know whether \( p \)”

• Use \( K_v \) and \( K_w \) to represent indefinite information ⇒ information returned by sensing actions.

• Combine labels with knowledge assertions.

- Examples

| [S] \( \neg K_{\text{open}}(\text{obj1}) \) |
| [S] [H] \( K_v \text{,train} \) |
| [S] [C_{SH}] \( K_w \text{,connected} \) |
Reasoning with labelled knowledge

• Rules for reasoning about speaker-hearer suppositions and common ground modalities (Steedman & Petrick 2007):

A1. \([X] \phi \Rightarrow \phi\)  
Supposition Veridicality

A2. \([X] \neg \phi \Rightarrow \neg [X] \phi\)  
Supposition Consistency

A3. \(\neg [X] \phi \Rightarrow [X] \neg [X] \phi\)  
Negative Introspection

A4. \([C] \phi \iff ([S] [C] \phi \land [H] [C] \phi)\)  
Common Ground

A5. \([X] [C] \phi \Rightarrow [X] \phi\)  
Common Ground Veridicality

• We require restricted versions of rules similar to these in order to augment PKS’s standard inference procedure.

⇒ No specific conversational rules or intent recognition rules are used.
Example: taking a train
Initial knowledge

F1. “If I know what time it is then I’ll know what track my train is on.”

\[[S] K_v \text{time} \Rightarrow add([S] K_v \text{train})\]

F2. “I don’t know what track my train leaves from.”

\[[S] \neg K_v \text{train}\]

F3. “I suppose you know what time it is.”

\[[S] [H] K_v \text{time}\]

F4. “I suppose it’s not common ground I don’t know what time it is.”

\[[S] \neg [C_{SH}] \neg [S] K_v \text{time}\]
Knowledge requirements of *ask* and *tell*

R1. “If X doesn’t know \( p \) and X knows Y does, X can ask Y about it.”

⇒ Knowledge-level preconditions for *ask*.

R2. “If X asks Y about \( p \), it makes it common ground X doesn’t know it.”

⇒ Knowledge-level effects for *ask*.

R3. “If X knows \( p \), and X knows \( p \) is not common ground, X can tell Y \( p \).”

⇒ Knowledge-level preconditions for *tell*.

R4. “If X tells Y \( p \), Y stops not knowing it and starts to know it.”

⇒ Knowledge-level effects for *tell*.
Knowledge-level dialogue actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ask(X, Y, p)</code></td>
<td>¬ [X] p</td>
<td><code>add(K_f, [C_{XY}] ¬ [X] p)</code></td>
</tr>
<tr>
<td></td>
<td>[X] [Y] p</td>
<td></td>
</tr>
<tr>
<td><code>tell(X, Y, p)</code></td>
<td>[X] p</td>
<td><code>add(K_f, [Y] p)</code></td>
</tr>
<tr>
<td></td>
<td>[X] ¬ [C_{XY}] p</td>
<td></td>
</tr>
</tbody>
</table>

- We can encode the knowledge requirements for dialogue actions like `ask` and `tell` in terms of their preconditions and effects.
- We can build plans by chaining together actions using our extra rules for reasoning about modalities.
Plan generation I

Goal: $[S] K_vtrain$ (I need to know which track to go to)

(D1) $\Rightarrow [H] K_vtime$  (F3),(A1)
(D2) $\Rightarrow \neg [S] K_vtime$  (F2),(A2),(F1)
(D3) $\Rightarrow$ Apply action: $ask(S, H, K_vtime)$
(D4) $\Rightarrow [C_{SH}] \neg [S] K_vtime$  (D3),(R2)
(D5) $\Rightarrow$ Apply action: $tell(H, S, K_vtime)$
(D6) $\Rightarrow [S] K_vtime$  (D5),(D2),(R4)
(D7) $\Rightarrow [S] K_vtrain$  (D6),(F1)

Plan: $[ask(S, H, K_vtime), tell(H, S, K_vtime)]$
Plan generation II

**Goal:** $[S] K_{vtrain}$ (I need to know which track to go to)

(D1) $\Rightarrow [S] \neg [S] K_{vtime}$

(D2) $\Rightarrow [S] \neg [C_{SH}] \neg [S] K_{vtime}$

(D3) $\Rightarrow$ Apply action: $tell(S, H, \neg [S] K_{vtime})$

(D4) $\Rightarrow [C_{SH}] \neg [S] K_{vtime}$

$\Rightarrow \ldots$

$\Rightarrow$ Apply action: $tell(H, S, K_{vtime})$

$\Rightarrow \ldots$

**Plan:** $[tell(S, H, \neg [S] K_{vtime}), tell(H, S, K_{vtime})]$
Dialogue and plan generation

<table>
<thead>
<tr>
<th>Plan I</th>
<th>Plan II</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ask(S, H, K_vtime)}</td>
<td>\texttt{tell(S, H, \neg [S] K_vtime)}</td>
</tr>
<tr>
<td>\texttt{tell(H, S, K_vtime)}</td>
<td>\texttt{tell(H, S, K_vtime)}</td>
</tr>
</tbody>
</table>

• Plan generation takes place in the space of multi-agent plans:
  – No reasoning about other agents’ goals or intentions.
  – Cannot guarantee other agents’ actions.
  – Planning is offline.

• Approach is driven by the knowledge state, i.e., what the planning agent knows about the world and the other agents’ beliefs.

• Both direct (Plan I) and indirect (Plan II) speech acts result from the same underlying planning and reasoning mechanisms.
Dialogue planning in PKS

• Approach I: encode problems directly in PKS.
• Approach II: extend PKS.
  – Index queries and updates by participant and common ground modalities, using existing PKS representations and reasoning mechanisms.
  – Restrict the form of the allowable knowledge assertions, e.g.,

  \[ [X] [Y] \cdots [X] \phi_{PKS}. \]

  – Implement new inference rules for reasoning with modalities.
  – Manage common ground as an instance of local closed world (LCW) information (Etzioni et al. 1994).

• We can easily generate simple plans, however, efficiency is a problem for larger plans.

• Current work: improving PKS’s search strategy.
Application: PACO-PLUS project (EU FP6)

“Perception, Action, and Cognition through learning of Object-Action Complexes”

http://www.paco-plus.org/

• Multiple robot platforms.
• Portions of the robot’s high-level representation are induced from its interaction with the real world.
• Task and dialogue planning using PKS.
Task planning in PACO-PLUS

Action descriptions

- **grasp(?o,?l,?h)**: Grasp object ?o from ?l using gripper ?h.
- **grasp-fromEdge(?o,?l,?h)**: Grasp object ?o from the edge of ?l using gripper ?h.
- **move(?l1,?l2)**: Move the robot from location ?l1 to location ?l2.
- **nudge-toEdge(?o,?l,?h)**: Nudge flat object ?o to the edge of ?l using gripper ?h.
- **open(?l,?h)**: Open ?l with gripper ?h.
- **open-partial(?l,?h)**: Partially open ?l with gripper ?h.
- **open-complete(?l,?h)**: Finish opening ?l with gripper ?h.
- **close(?l,?h)**: Close ?l with gripper ?h.
- **pass-object(?o,?h1,?h2)**: Pass object ?o from gripper ?h1 to ?h2.
- **place-upright(?o,?l,?h)**: Put object ?o upright at ?l using gripper ?h.
- **put-down(?o,?l,?h)**: Put object ?o down at ?l using gripper ?h.
- **put-in(?o,?l,?h)**: Put object ?o into ?l using gripper ?h.
- **remove-from(?o,?l,?h)**: Remove object ?o from ?l using gripper ?h.
- **sense-open(?o)**: Sense whether object ?o is open or not.

Example plan: ensure the applejuice is in the fridge

- place-upright(applejuice,sideboard,lefthand)
- grasp(applejuice,sideboard,righthand)
- move(sideboard,fridge)
- open-partial(fridge,lefthand)
- pass-object(applejuice,righthand,lefthand)
- open-complete(fridge,righthand)
- put-in(applejuice,fridge,lefthand)
- close(fridge,lefthand)

(Petrick et al. 2009)
Dialogue planning in PACO-PLUS

Robot1: Let’s make breakfast.
Robot2: I don’t know how to make breakfast.
Robot1: To make breakfast we must bring the cereal and the milk to the sideboard.
Robot2: Is the cereal at the sideboard?
Robot1: No.
Robot2: Where is the cereal?
Robot1: The cereal is in the cupboard.
Robot2: Is the milk at the sideboard?
Robot1: No.
Robot2: Where is the milk?
Robot1: The milk is in the fridge.
Robot2: Okay. I suggest I go to the cupboard, pickup the cereal, bring it to the sideboard, then go to the fridge, pickup the milk, and bring it to the sideboard.

⇒ The nature of the domain limits the dialogue context somewhat.
⇒ The same underlying plan generation mechanisms will be used for both task planning and dialogue planning.
Summary and future work

- Certain dialogue actions like *ask* and *tell* can be encoded as knowledge-level planning operators with sensing effects.

- Plans can be generated by reasoning about the planner’s knowledge state, without reference to specific conversational rules or intent/goal recognition. (This work does not preclude the use of such techniques.)

- Direct and indirect speech acts can be generated from the same underlying planning process.

- We are extending PKS to improve its ability to generate dialogue plans, and are building a set of tools for integration with robot systems.

- Future work: evaluation. To what extent can we use this approach for “real” dialogue problems?

- Natural language problems offer suitable challenges for driving research in the planning community.
References


