Planning for natural language: adventures in instruction giving and robot dialogue

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

Rutgers, New Brunswick, New Jersey, USA
8 September 2010
Taking the train (Scenario I)

“I want to take the train from Newark to New Brunswick.”

Go to the station, buy a ticket, check the departure board for track information, go to the track, board the train, . . ., enjoy New Brunswick!
Taking the train (Scenario II)

“I want to take the train from Newark to New Brunswick.”

Go to the station, buy a ticket, ask someone for track information, go to the track, board the train, . . . , enjoy New Brunswick!
Taking the train (Scenarios I & II)

<table>
<thead>
<tr>
<th>Plan I</th>
<th>Plan 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 New Brunswick!</td>
<td>Enjoy New Brunswick!</td>
</tr>
</tbody>
</table>

⇒ 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?
Instruction giving (Scenario III)

<table>
<thead>
<tr>
<th>Plan I</th>
<th>Plan 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 New Brunswick!</td>
<td>Enjoy New Brunswick!</td>
</tr>
</tbody>
</table>

⇒ Both plans serve as good instruction sets for directing other agents to achieve the goal.

⇒ Can we use the same machinery for generating action plans to build instructional plans?
Outline

1. Automated planning
2. Natural language and planning
3. Planning in instruction giving (GIVE)
4. Planning in robot dialogue (PACO-PLUS)
5. Conclusions

⇒ Joint work with Alexander Koller (Universität des Saarlandes) on GIVE and Mark Steedman (University of Edinburgh) on PACO-PLUS.
Target scenarios

GIVE

(Koller & Petrick 2008)

PACO-PLUS

Image: Asfour et al., Karlsruhe Institute of Technology
Automated planning

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

- Classical planning
- Planning with incomplete information and sensing
- Hierarchical planning
- Probabilistic planning
- Planning with costs and preferences
- Action learning
- Spatial reasoning
- Temporal planning
- Planning with control knowledge
- SAT planning
- Heuristic search
- Plan execution
- ...

Ron Petrick / Planning for natural language: instruction giving and robot dialogue / Rutgers / 2010-09-08
Natural language and planning

• **Natural Language Generation (NLG)** is a major subfield of natural language processing, concerned with computing natural language sentences or texts that convey a piece of information to a user.

• **Dialogue systems** are computer systems designed to carry out natural language conversations with human users. A central component of most dialogue systems is the **dialogue manager** which is responsible for making appropriate conversational moves.

• Can be viewed as problems involving actions, beliefs, and goals:

  A speaker tries to change the mental state of the hearer by applying actions that correspond to the utterance of words or sentences.

⇒ Obvious parallels to planning.
Natural language and planning...


• Early approaches suffered due to inefficient planning techniques.

• Recent work has tended to separate task planning from other types of natural language 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).

⇒ To what extent can we apply **general purpose planning techniques** to problems in natural language?
Planning in instruction giving
GIVE Challenge

• “Generating Instructions in Virtual Environments” (Koller et al. 2007).
• A challenge for the NLG community

Build an NLG system capable of producing natural language instructions to guide a human user in performing some task in a virtual environment.

• A theory-neutral task that tests all components of an NLG system.
• Virtual 3D client; evaluation possible over the Internet.
• GIVE-1 ran from March 2008 to February 2009
  – 5 systems were evaluated using data from over 1100 game runs.
  – Largest ever NLG evaluation effort in terms of the number of experimental subjects (claim).
• GIVE-2 evaluation was scheduled to start in February 2010
  (http://www.give-challenge.org/research/).
Example GIVE map

Virtual 3D GIVE client

(Koller & Petrick 2008)
GIVE as a planning problem

- A GIVE problem is very similar to a Grid planning problem (IPC 1998).
  - Discretised tiles of equal size,
  - Users can turn 90° left or right, or move forward one tile,
  - Additional requirement to press buttons in the right order, reason about large numbers of world objects, and navigate complicated room shapes.

⇒ Model the task domain as a classical planning problem in PDDL (McDermott et al. 1998) and generate plans using existing off-the-shelf planners.
GIVE domain state space

### Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{move}(p_1, p_2, o))</td>
<td>Move from position (p_1) to position (p_2) in orientation direction (o).</td>
</tr>
<tr>
<td>(\text{turn-left}(o_1, o_2))</td>
<td>Turn left from orientation (o_1) to orientation (o_2).</td>
</tr>
<tr>
<td>(\text{turn-right}(o_1, o_2))</td>
<td>Turn right from orientation (o_1) to orientation (o_2).</td>
</tr>
<tr>
<td>(\text{manipulate-button-on-off}(b, p))</td>
<td>Manipulate button (b) in position (p) from on to off.</td>
</tr>
<tr>
<td>(\text{manipulate-button-off-on}(b, p))</td>
<td>Manipulate button (b) in position (p) from off to on.</td>
</tr>
</tbody>
</table>

### Properties (relations)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{adjacent}(p_1, p_2, o))</td>
<td>Position (p_1) is adjacent to position (p_2) in orientation direction (o).</td>
</tr>
<tr>
<td>(\text{alarmed}(p))</td>
<td>Position (p) is alarmed.</td>
</tr>
<tr>
<td>(\text{next-orientation}(o_1, o_2))</td>
<td>Next orientation from direction (o_1) in a clockwise direction is (o_2).</td>
</tr>
<tr>
<td>(\text{object-position}(b, p))</td>
<td>Button (b) is at position (p).</td>
</tr>
<tr>
<td>(\text{object-state}(b, s))</td>
<td>Button (b) is in state (s).</td>
</tr>
<tr>
<td>(\text{player-position}(p))</td>
<td>Player is at position (p).</td>
</tr>
<tr>
<td>(\text{player-orientation}(o))</td>
<td>Player has orientation (o).</td>
</tr>
<tr>
<td>(\text{releases}(b, p))</td>
<td>Button (b) releases (disables) the alarm in position (p).</td>
</tr>
</tbody>
</table>

### Objects/constants

<table>
<thead>
<tr>
<th>Object/constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{north, south, east, west})</td>
<td>Orientations</td>
</tr>
<tr>
<td>(\text{on, off})</td>
<td>Button states</td>
</tr>
<tr>
<td>(u_1, l_1, u_2, l_2, \ldots)</td>
<td>Button names</td>
</tr>
<tr>
<td>(\text{pos.0.0, pos.0.1, \ldots pos.7.23, \ldots})</td>
<td>Grid positions</td>
</tr>
</tbody>
</table>
GIVE actions and plans

- Classical planning actions describe task-level operations, e.g., “move”, “turn”, “press button”, etc. (Koller & Petrick 2008).

```
(:action move
  :parameters (?from - position
                ?to - position
                ?ori - orientation)
  :precondition
    (and (player-position ?from)
         (adjacent ?from ?to ?ori)
         (player-orientation ?ori)
         (not (alarmed ?to)))
  :effect
    (and (not (player-position ?from))
         (player-position ?to)))
```

- A plan for a $2 \times 2$ GIVE world with 2 buttons:

```
move(pos_1_1, pos_1_2, north),
manipulate-button-off-on(u1, pos_1_2),
turn-right(north, east),
move(pos_1_2, pos_2_2, east),
turn-right(east, south),
move(pos_2_2, pos_2_1, south),
manipulate-button-off-on(l1, pos_2_1).
```
Challenges for NLG

• Task plans often need to be post-processed by other components of the NLG system.

• Plan summarisation
  
  “Turn left; turn left; walk forward”
  \[\Rightarrow\] “Turn around and walk through the door”

• Plan elaboration

  “Press the green button on the wall to your right”
  \[\Rightarrow\] “Walk to the centre of the room; turn right; now press the green button in front of you”
Challenges for planning

• Strict run-time requirements
  – Planning must happen in (near) real time,
  – User can interact with the world while the system is deliberating.

• System must monitor a user’s actions and compare them against the generated instruction set
  – Mental state of the user is not known,
  – User’s actions may not match the generated instructions exactly but still meet the intended goal,
  – User can communicate intentions through action and inaction.

• Plans are often non-trivial, e.g., 108 steps in sample domain.
Experiment 1: minimal GIVE worlds

- Simplified world consisting of an $N \times h$ grid with buttons in alternating positions along the north and south walls.
- Player starts in position (1,1).
- All buttons must be pressed to successfully complete the game.
  - Unordered: buttons can be pressed in any order.
  - Ordered: buttons must be pressed in an alternating fashion from west to east, e.g., $u_1, l_1, u_2, l_2$ in the above grid.
Experiment 1: results \((h = 20)\)

(a) Unordered

(b) Ordered

(Koller & Petrick 2010)
Experiment 1: runtime v. grounding time

(Koller & Petrick 2010)
Experiment 2: GIVE with extra grid cells

• We vary the structure of the world by adding an extra $w \times h$ empty grid cells to the right of the minimal world.
Experiment 2: results \((h = 20, N = 20)\)

(a) Unordered

(b) Ordered

(Koller & Petrick 2010)
Experiment 3: variation on Experiment 2

- Use a fixed grid size.
- Vary the number of buttons (and, hence, the free grid space).
Experiment 3: results \( (h = 20, N = 40) \)

(Koller & Petrick 2010)
Observations on GIVE

• Full evaluation results in (Koller & Petrick 2010).

• Mixed results on certain GIVE problems
  – Planning times over a couple seconds can negatively affect the overall response time of an NLG system.
  – Restricting bottleneck is often the initial preprocessing stage performed by many modern planners.

• Planners like FF (Hoffmann & Nebel 2001) and SGPLAN (Hsu et al. 2006) are good at controlling search in the GIVE domain but the total planning time is sometimes dominated by grounding time. Large problem instances remain a challenge.

• Changing the representation can alter planner performance.

• It’s not all bad news for planning! The GIVE challenge wouldn’t have been possible 15 years ago...
Planning for robot dialogue
PACO-PLUS project (EU FP6)

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

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

- Multiple robot platforms.
- Object manipulation in a kitchen environment.
- Same underlying mechanism is used for both task and dialogue planning.

Image: Asfour et al., Karlsruhe Institute of Technology

Image: Kraft & Krüger, University of Southern Denmark
PACO-PLUS: task planning in a kitchen domain

“Classical” planning actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grasp(o, l, h)</td>
<td>Grasp object o from l using gripper h.</td>
</tr>
<tr>
<td>graspFromEdge(o, l, h)</td>
<td>Grasp object o from the edge of l using gripper h.</td>
</tr>
<tr>
<td>move(l₁, l₂)</td>
<td>Move the robot from location l₁ to location l₂.</td>
</tr>
<tr>
<td>nudgeToEdge(o, l, h)</td>
<td>Nudge flat object o to the edge of l using gripper h.</td>
</tr>
<tr>
<td>open(l, h)</td>
<td>Open l with gripper h.</td>
</tr>
<tr>
<td>openPartial(l, h)</td>
<td>Partially open l with gripper h.</td>
</tr>
<tr>
<td>openComplete(l, h)</td>
<td>Finish opening l with gripper h.</td>
</tr>
<tr>
<td>close(l, h)</td>
<td>Close l with gripper h.</td>
</tr>
<tr>
<td>passObject(o, h₁, h₂)</td>
<td>Pass object o from gripper h₁ to h₂.</td>
</tr>
<tr>
<td>placeUpright(o, l, h)</td>
<td>Put object o upright at l using gripper h.</td>
</tr>
<tr>
<td>putDown(o, l, h)</td>
<td>Put object o down at l using gripper h.</td>
</tr>
<tr>
<td>putIn(o, l, h)</td>
<td>Put object o into l using gripper h.</td>
</tr>
<tr>
<td>removeFrom(o, l, h)</td>
<td>Remove object o from l using gripper h.</td>
</tr>
</tbody>
</table>

Example plan: ensure the applejuice is in the fridge:

placeUpright(applejuice, sideboard, lefthand),
grasp(applejuice, sideboard, righthand),
move(sideboard, fridge),
openPartial(fridge, lefthand),
passObject(applejuice, righthand, lefthand),
openComplete(fridge, righthand),
putIn(applejuice, fridge, lefthand),
close(fridge, lefthand).

(Petrick et al. 2009)
PACO-PLUS: task planning in a kitchen domain...

⇒ Classical STRIPS-style planning is often sufficient for many task-based problems in PACO-PLUS.
**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>pickup(x)</td>
<td>handEmpty</td>
<td>holding(x)</td>
<td>handEmpty</td>
</tr>
<tr>
<td></td>
<td>onTable(x)</td>
<td></td>
<td>onTable(x)</td>
</tr>
<tr>
<td>dropInBox(x, y)</td>
<td>holding(x)</td>
<td>inBox(x, y)</td>
<td>holding(x)</td>
</tr>
<tr>
<td></td>
<td>box(y)</td>
<td>handEmpty</td>
<td>empty(y)</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.
- STRIPS forms the core of PDDL (McDermott et al. 1998), the language of many modern planners and the International Planning Competition.
Planning with incomplete information

• Problem: classical STRIPS planning assumes complete knowledge and deterministic action effects, which is not always realistic.

• 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).

• Reasoning about sensing requires the ability to reason effectively about the agent’s knowledge/beliefs.

⇒ This is often the case with dialogue.

⇒ This work: use the PKS (Planning with Knowledge and Sensing) planner for dialogue planning.
Planning with Knowledge and Sensing

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

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

• 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 ($KB$).

• Actions are defined in terms of the changes they make to the planner’s knowledge state (i.e., the databases), rather than the world state.

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

• PKS has previously been applied to traditional planning benchmarks, robot systems, web services, operating system applications.
Knowledge representation in PKS

- $K_f$: 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.
Representing actions in PKS

<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>$add(K_v, \text{phoneNumber})$</td>
</tr>
<tr>
<td>dial</td>
<td>$K_v(\text{phoneNumber})$</td>
<td>$add(K_f, \text{dialledOk})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$add(K_w, \text{connected})$</td>
</tr>
</tbody>
</table>

- 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 $DB$ (corresponding to some $KB$)
  - Effects update $DB \Rightarrow$ update $KB$
- Plans are generated by searching the space of database states.
Example: planning in PKS

\[ K_f \text{ havePaper} \quad \text{readPaper} \quad K_f \text{ havePaper} \quad \text{dial} \quad K_f \text{ havePaper dialledOk} \]

\[ K_v \text{ phoneNumber} \]

\[ K^+ \]

\[ K^- \]

\[ K_f \text{ havePaper dialledOk connected} \]

\[ K_v \text{ phoneNumber} \]

\[ K_v \text{ phoneNumber} \]

\[ K_v \text{ phoneNumber} \]
What about dialogue?
Participants and common ground

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

[S] Speaker supposition
[H] Hearer supposition
[X], [Y], ... Other participant/agent suppositions
[C_{XY}] Common ground between X and Y

Examples

[S] p “The speaker supposes p.”
[S] [H] p “The speaker supposes the hearer supposes p.”
[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 underlying basis of knowledge expressions

\[
K_p \quad \text{“Know } p\text{”}
\]
\[
K_v t \quad \text{“Know the value of } t\text{”}
\]
\[
K_w p \quad \text{“Know whether } p\text{”}
\]

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

• Combine labels with knowledge assertions.

Examples

| [S] \( \neg K_{open}(obj1) \) |
| [S] [H] \( K_{v\_train} \) |
| [S] [C_{SH}] \( K_{w\_connected} \) |
Reasoning with labelled knowledge

• Common 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 these rules in order to augment PKS’s ordinary inference procedure.

⇒ No specific conversational rules or intent recognition rules are used.
Plan generation with dialogue actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
</table>
| \textit{ask}(X, Y, p) | $\neg [X] p$  
$[X] [Y] p$ | \textit{add}(K_f, [C_{XY}] \neg [X] p) |
| \textit{tell}(X, Y, p)  | $[X] p$       
$[X] \neg [C_{XY}] p$ | \textit{add}(K_f, [Y] p) |

- We can encode the knowledge requirements of speech acts like \textit{ask} and \textit{tell} in terms of their preconditions and effects.
- We can build plans by chaining together actions using PKS’s plan generation engine, extended to reason about agent modalities (e.g., $[X]$, $[Y]$, and $[C]$).
Example 1: Asking for track information

- Initial knowledge

\[
[S] \ K_v^{\text{track}} \Rightarrow \text{add}(K_f, [S] \ K_v^{\text{train}}) \\
[S] \ \neg K_v^{\text{train}} \\
[S] \ [H] \ K_v^{\text{track}} \\
[S] \ \neg [C_{SH}] \ \neg [S] \ K_v^{\text{track}}
\]

- Actions: ask, tell

- Possible plans:

<table>
<thead>
<tr>
<th>Plan 1</th>
<th>Plan 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{ask}(S, H, K_v^{\text{track}})</td>
<td>\textit{tell}(S, H, \neg [S] \ K_v^{\text{track}})</td>
</tr>
<tr>
<td>\textit{tell}(H, S, K_v^{\text{track}})</td>
<td>\textit{tell}(H, S, K_v^{\text{track}})</td>
</tr>
</tbody>
</table>

Plan 1 is an example of a \textbf{direct speech act}.
Plan 2 is an example of an \textbf{indirect speech act}. 
Example 2: PACO-PLUS kitchen domain

Robot1: Let’s make breakfast.
Robot2: Do you know where the milk is?
Robot1: The milk is in the fridge.
Robot2: Is the cereal at the sideboard?
Robot1: No.
Robot2: Where is the cereal?
Robot1: The cereal is in the cupboard.
Robot2: Okay. I suggest I go to the fridge, pickup the milk, bring it to the sideboard, then go the cupboard, pickup the cereal, and bring it to the sideboard.

⇒ Generate both physical plans and dialogue plans in the same domain.
Example plan: bring the milk to the sideboard

ask-location(robot1,milk)
receive-location(robot1,milk)
move(sideboard,location(milk))
branch(location(milk))
  K(location(milk) = fridge):
    open-partial(fridge,lefthand)
    open-complete(fridge,righthand)
    remove-from(milk,fridge,righthand)
    close(fridge,lefthand)
  K(location(milk) = stove):
    grasp(milk,stove,righthand)
  ...
move(location(milk),sideboard)
put-down(milk,sideboard,righthand).
Example plan: bring the milk to the sideboard

1. ask-location(robot1, milk)
2. receive-location(robot1, milk)
3. move(sideboard, location(milk))
4. location(milk)?
   - location(milk) = fridge
     - open-partial(fridge, lefthand)
     - open-complete(fridge, righthand)
     - remove-from(milk, fridge, righthand)
     - close(fridge, lefthand)
     - move(location(milk), sideboard)
     - put-down(milk, sideboard, righthand)
   - location(milk) = stove
     - grasp(milk, stove, righthand)
   - location(milk) = ...

5. ...

Ron Petrick / Planning for natural language: instruction giving and robot dialogue / Rutgers / 2010-09-08
Observations about PKS dialogue planning

- Plan generation takes place in the space of multi-agent plans
  - No reasoning about other agents’ goals or intentions,
  - Cannot guarantee other agents’ actions.

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

- Both direct and indirect speech acts can be generated from the same mechanisms for reasoning about knowledge and common ground, without reference to specific conversational rules.

- Evaluation (forthcoming).

Conclusions

• Modern planning techniques offer potential solutions to many challenging problems in natural language.

• The same mechanisms used for ordinary task planning can often be applied to problems like instruction giving and dialogue planning.

• Open question: to what extent can automated planning techniques be used for “real” problems in natural language?

• Choosing the “right” planner and deciding what the “right” problem is can sometimes be difficult.

⇒ Natural language problems offer suitable challenges that can help drive research in the planning community.
References


C. W. Hsu, B. W. Wah, R. Huang, and Y. X. Chen. 2006. New features in SGPlan for handling soft constraints and goal preferences in PDDL 3.0. *5th International Planning Competition at ICAPS 2006*.


Ron Petrick / Planning for natural language: instruction giving and robot dialogue / Rutgers / 2010-09-08


