Planning for natural language: instruction giving, robot dialogue, and social interaction

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

Heriot-Watt University, Edinburgh, Scotland, United Kingdom
12 January 2011
Taking the train (Scenario I)

“I want to take the train from Edinburgh to Glasgow.”

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

“I want to take the train from Edinburgh to Glasgow.”

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

<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 Glasgow!</td>
<td>Enjoy Glasgow!</td>
</tr>
</tbody>
</table>

**Observation 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?
## Instruction giving

<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 Glasgow!</td>
<td>Enjoy Glasgow!</td>
</tr>
</tbody>
</table>

⇒ Both plans serve as good instruction sets for directing agents.

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

<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 Glasgow!</td>
<td>Enjoy Glasgow!</td>
</tr>
</tbody>
</table>

⇒ How do the requirements of social interaction affect the planning process?
Outline

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

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

GIVE

PACO-PLUS

JAMES
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: GIVE, PACO-PLUS, JAMES / Heriot-Watt University / 2011-01-12
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 onTable(x)</td>
<td>holding(x)</td>
<td>handEmpty onTable(x)</td>
</tr>
<tr>
<td>dropInBox(x, y)</td>
<td>holding(x) box(y)</td>
<td>inBox(x, y)</td>
<td>holding(x) 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.
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...

• The link between natural language and planning has a long tradition, e.g., (Perrault & Allen 1980, Appelt 1985, Clark 1996, Stone 2000), including early BDI-based approaches, e.g., (Litman & Allen 1987, Cohen & Levesque 1990, Grosz & Sidner 1990), \ldots

• 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 specialised approaches, e.g., finite state machines, information state, rule-based approaches to speech act theories, dialogue games, \ldots

• 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 arising from natural language?
Planning in instruction giving
GIVE Challenge

• “Generating Instructions in Virtual Environments” (Koller et al. 2007).

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

  http://www.give-challenge.org/research/.

• A theory-neutral task that tests all components of an NLG system.

• Virtual 3D client; evaluation possible over the Internet.

• Largest ever NLG evaluation effort in terms of experimental subjects.
  – GIVE-2 (completed mid-2010): 7 systems, data from 1800+ game runs, participation from 39 countries.
  – GIVE-2.5 is planned for 2010–2011.
Example GIVE map

Virtual 3D GIVE client

(Koller & Petrick 2008)
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.
Actions

- \textit{move}(p_1, p_2, o)\quad\text{Move from position } p_1 \text{ to position } p_2 \text{ in orientation direction } o.\)
- \textit{turn-left}(o_1, o_2)\quad\text{Turn left from orientation } o_1 \text{ to orientation } o_2.\)
- \textit{turn-right}(o_1, o_2)\quad\text{Turn right from orientation } o_1 \text{ to orientation } o_2.\)
- \textit{manipulate-button-on-off}(b, p)\quad\text{Manipulate button } b \text{ in position } p \text{ from on to off.}\)
- \textit{manipulate-button-off-on}(b, p)\quad\text{Manipulate button } b \text{ in position } p \text{ from off to on.}\)

Properties (relations)

- \textit{adjacent}(p_1, p_2, o)\quad\text{Position } p_1 \text{ is adjacent to position } p_2 \text{ in orientation direction } o.\)
- \textit{alarmed}(p)\quad\text{Position } p \text{ is alarmed.}\)
- \textit{next-orientation}(o_1, o_2)\quad\text{Next orientation from direction } o_1 \text{ in a clockwise direction is } o_2.\)
- \textit{object-position}(b, p)\quad\text{Button } b \text{ is at position } p.\)
- \textit{object-state}(b, s)\quad\text{Button } b \text{ is in state } s.\)
- \textit{player-position}(p)\quad\text{Player is at position } p.\)
- \textit{player-orientation}(o)\quad\text{Player has orientation } o.\)
- \textit{releases}(b, p)\quad\text{Button } b \text{ releases (disables) the alarm in position } p.\)

Objects/Constants

- \textit{north, south, east, west}\quad\text{Orientations}\)
- \textit{on, off}\quad\text{Button states}\)
- \textit{u_1, l_1, u_2, l_2, \ldots}\quad\text{Button names}\)
- \textit{pos_0_0, pos_0_1, \ldots pos_7_23, \ldots}\quad\text{Grid positions}\)
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”
  ⇒ “Turn around and walk through the door”

• Plan elaboration
  “Press the green button on the wall to your right”
  ⇒ “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 the 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 2011)
Experiment 1: runtime v. grounding time

(a) Unordered

(b) Ordered

(Koller & Petrick 2011)
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 \((h = 20, N = 20)\)

(Koller & Petrick 2011)
Observations on GIVE

• Mixed results on certain problems (full evaluation in (Koller & Petrick 2011)).
  – 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
“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><code>grasp(o, l, h)</code></td>
<td>Grasp object o from l using gripper h.</td>
</tr>
<tr>
<td><code>graspFromEdge(o, l, h)</code></td>
<td>Grasp object o from the edge of l using gripper h.</td>
</tr>
<tr>
<td><code>move(l_1, l_2)</code></td>
<td>Move the robot from location l_1 to location l_2.</td>
</tr>
<tr>
<td><code>nudgeToEdge(o, l, h)</code></td>
<td>Nudge flat object o to the edge of l using gripper h.</td>
</tr>
<tr>
<td><code>open(l, h)</code></td>
<td>Open l with gripper h.</td>
</tr>
<tr>
<td><code>openPartial(l, h)</code></td>
<td>Partially open l with gripper h.</td>
</tr>
<tr>
<td><code>openComplete(l, h)</code></td>
<td>Finish opening l with gripper h.</td>
</tr>
<tr>
<td><code>close(l, h)</code></td>
<td>Close l with gripper h.</td>
</tr>
<tr>
<td><code>passObject(o, h_1, h_2)</code></td>
<td>Pass object o from gripper h_1 to h_2.</td>
</tr>
<tr>
<td><code>placeUpright(o, l, h)</code></td>
<td>Put object o upright at l using gripper h.</td>
</tr>
<tr>
<td><code>putDown(o, l, h)</code></td>
<td>Put object o down at l using gripper h.</td>
</tr>
<tr>
<td><code>putIn(o, l, h)</code></td>
<td>Put object o into l using gripper h.</td>
</tr>
<tr>
<td><code>removeFrom(o, l, h)</code></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.
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 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 \cite{Etzioni1994}
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>$\text{add}(K_v, \text{phoneNumber})$</td>
</tr>
<tr>
<td>dial</td>
<td>$K_v(\text{phoneNumber})$</td>
<td>$\text{add}(K_f, \text{dialledOk})$ $\text{add}(K_w, \text{connected})$</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 $\text{DB}$ (corresponding to some $\text{KB}$)
  - Effects update $\text{DB} \Rightarrow \text{update } \text{KB}$

• Plans are generated by searching the space of database states.
Example: planning in PKS

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

\[ K_v \text{ phoneNumber} \]

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

\[ K_v \text{ phoneNumber} \]

\[ K_v \text{ connected} \]

\[ K^- \]

\[ K^+ \]
What about dialogue?
Participants and common ground

• Use labels (modalities) for referencing dialogue participants and common ground:

  - \([S]\) Speaker supposition
  - \([H]\) Hearer supposition
  - \([X, Y, \ldots]\) Other participant/agent suppositions
  - \([C_{XY}]\) Common ground between \(X\) and \(Y\)

• Combine with restricted PKS knowledge assertions:

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

• Enhance PKS reasoning with a set of rules about speaker-hearer suppositions and common ground modalities (Steedman & Petrick 2007).
Plan generation with dialogue actions

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

- We can encode the knowledge requirements of speech acts like *ask* and *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: Asking for track information

• Initial knowledge

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

• Example plans:

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

Plan 1 is an example of a **direct speech act**.
Plan 2 is an example of an **indirect speech act**.
Example: 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.
Example plan: bring the milk to the sideboard

```
ask-location(robot1,milk)
receive-location(robot1,milk)
move(sideboard,location(milk))

location(milk)?

location(milk) = fridge
open-partial(fridge,lethand)
open-complete(fridge,righthand)
remove-from(milk,fridge,righthand)
close(fridge,lethand)
move(location(milk),sideboard)
put-down(milk,sideboard,righthand)
...

location(milk) = stove
grasp(milk,stove,righthand)
...

location(milk) = ...
...
```

Ron Petrick / Planning for natural language: GIVE, PACO-PLUS, JAMES / Heriot-Watt University / 2011-01-12
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 is forthcoming.

⇒ XPERIENCE / EU FP7 (2011 - 2015)
Robots Bootstrapped through Learning from Experience
http://xperience.org/
Planning in social interaction
Joint Action for Multimodal Embodied Social Systems

http://james-project.eu/

- Develop an embodied artificial agent that supports socially appropriate, multi-party, multimodal interactions.
- Endow a robot that can recognise, understand, and generate appropriate multimodal social signals in real-world, task-oriented contexts.
- Demonstrate in a bartending scenario.
Motivating scenario

Interaction 1 (Socially inappropriate)

*Two people, A and B, each individually approach the robot bartender*

Robot (to A): How can I help you?
A: A pint of cider, please.

A *third person, C, approaches the robot and attracts its attention by gesturing*

Robot (to C): How can I help you?
C: I’d like a pint of bitter.
Robot: (Serves C)
Robot (to B): What will you have?
B: A pint of Guinness.
Robot: (Serves B)
Robot: (Serves A)

Interaction 2 (Socially appropriate)

Robot (to A): How can I help you?
A: A pint of cider, please.

Robot (to C): Wait a moment please
Robot: (Serves A)
Robot (to B): What will you have?
B: A pint of Guinness.
Robot: (Serves B)
Robot (to C): Thanks for waiting.
How can I help you?
C: I’d like a pint of bitter.
Robot: (Serves C)
From JAST to JAMES

• Starting point: JAST / EU FP6 (2005 – 2009)
  Joint Action Science and Technology

• JAMES: focus on “getting the social interaction right”.

Approach

1. Record and analyse multimodal social and task-based behaviour of humans engaged in joint task-based interactions.

2. Use the data to train models able to understand human affective, social, and task-related behaviour, and to generate socially appropriate responses.

3. Integrate the implemented models onto a robot platform and evaluate with human users in a robot bartender scenario.
Architecture and research areas

- Visual Perception of Human Activity
- Natural Language Communication
- Social State Processing
- Planning and Reasoning
- Machine Learning for Social Skills Execution
- Social Robotics and Embodiment

Diagram:
- Planner/Execution Monitor
- Social State Recogniser
- Visual Processing
- Parser
- Speech Recogniser
- Talking-Head Controller
- Motion Planner
- Output Planner
- Robot Simulator
- Learned Policies

Real World
PKS planning in JAMES

- Extend models of knowledge and action to the joint representation of task and social acts.
- Adapt PKS’s plan generation techniques to the social state spaces arising in JAMES, as a means of addressing scalability concerns.
- Use results from studies on communicative intention and machine learning experiments, for reasoning about uncertainty and providing domain-specific heuristics for planning in social state spaces.
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.

• We believe these techniques will also extend to social state spaces.

• 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*.


References...


