Human-Computer Interaction

Collaborative Discourse and Plan Recognition

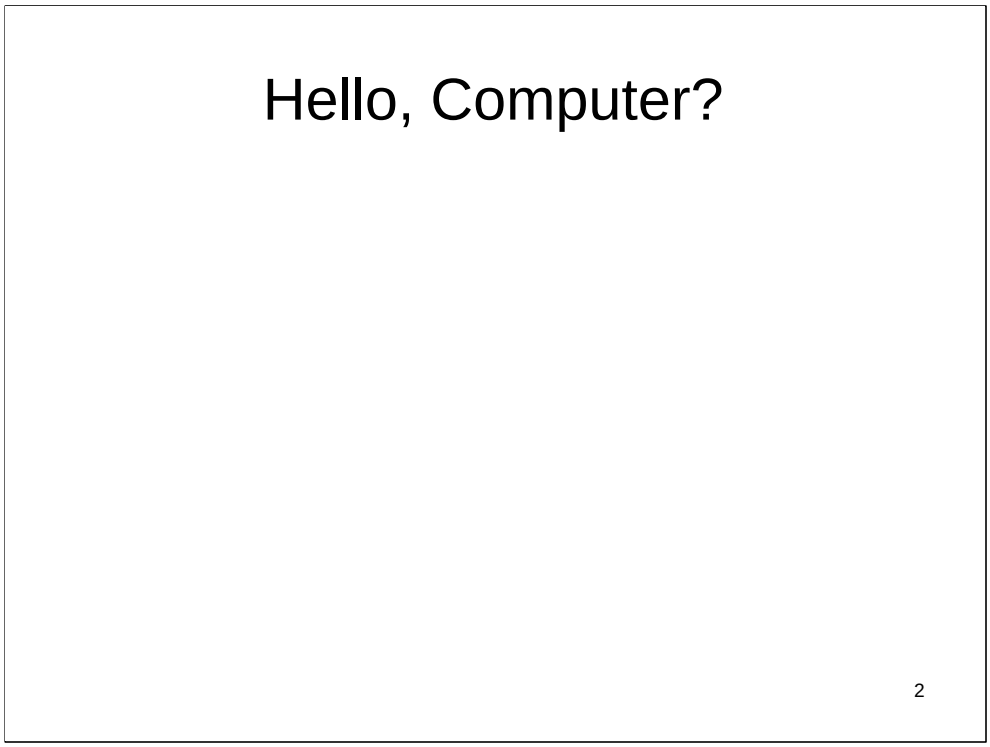
Thomas Coffee

Shannon Dong

Shen Qu

4/13/2005

Through this lecture, we aim to convey the usefulness of intelligent systems capable of understanding and collaborating with humans towards the accomplishment of a common goal. Specifically, we will focus on the theory of collaborative discourse and plan recognition and their role in current collaborative assistants.



"Hello, computer?" These were the words uttered by Professor Scotts (a.k.a. Scotty) to the mouse of a 20th century computer when the crew of the Enterprise traveled back in time to year 1986. This scene is probably a familiar sight to all Star Trek fans, but one may wonder "what does it have to do with cognitive robotics?"

Well, anyone who's seen any form of Star Trek probably encountered scenario(s) where a crewmember works with the onboard computer system to solve various types of complex problems. And although not immediately obvious, this problem solving process not only involves command inputs from the human party but also demands a computer software system capable of keeping track of the problem's state, any progress made, and what still needs to be done while providing inputs and quires as needed. When Scotty said "hello computer," he was not simply seeking a "how do you do" reply but was trying to start a collaborative session in which the computer would assist him in the accomplishment of a goal (in this case, constructing the molecular structure of transparent aluminum).

Through this lecture, we will demonstrate the importance/usefulness of such computer/software agents, present a couple of such agents currently under development, and dive into the theory behind a couple of key building blocks of these agents.

TRIPS Demo

3

Before going into technical details, we would first like to show a demonstration of a collaborative agent TRIPS applied to a military scenario. The demo is self explanatory in terms of the problem statement and final goals. We are showing this demo early on to give people a sense of what is available today.

Note that in this scenario the computer received a high level goal from the user, provided relevant information without being specifically prompted to do so, made recommendations towards the course of action, kept track of parallel solution approaches to the goal, retraced steps when an approach did not work out, asked for clarification when information provided is unclear or incomplete, and provided both visual and vocal feedback to the user. These are all important properties for this type of collaborative agents, and towards the end of this lecture we will show a similar demo in which we link these key features to the theory that we will soon present.

This demo can be found at the following url:
http://www.cs.rochester.edu/research/cisd/projects/trips/movies/TRIPS_CPoF/

How can we create computer or robotic systems that not only follow user commands but also collaborate with humans toward the achievement of a common goal?

4

This question outlines the main objective of this lecture. Key words to note are “collaborate” and “common goal.”

Technical Approach

- Derive intentions from observed or communicated information
- Decompose goals into organized structure
- Refine goal structures by seeking additional information
- Identify shared elements of agents' goals
- Execute communications and actions to achieve multi-agent objectives

5

Human-computer/robot interaction is a wide field. Even a specific type of interactive agent such as TRIPS is very complex. This slide only outlines the technical approach to what we consider to be the most important components to this type of agents.

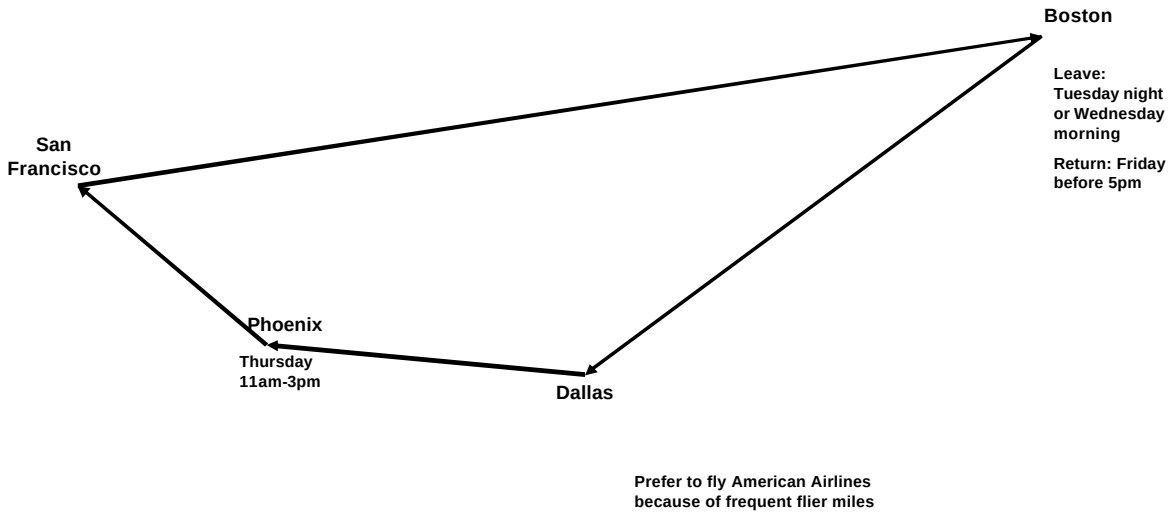
Outline

- **Travel Scheduling Example**
- Collaborative Discourse
- Plan Recognition
- TRIPS
- COLLAGEN

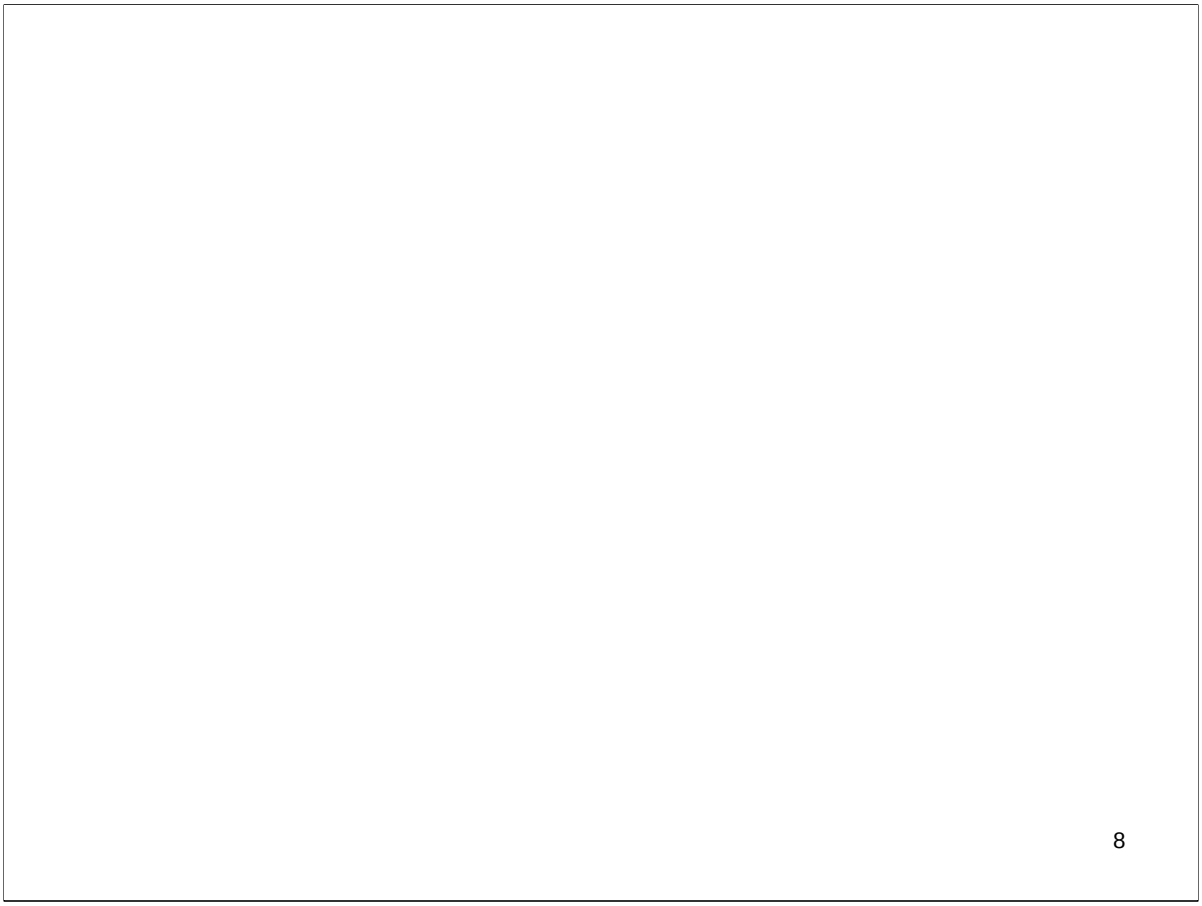
6

For the reminder of this lecture, we will first introduce a travel planning example in which collaborative agents can be applied. And we will continue to refer back to this example through our technical discussion on Collaborative Discourse and Plan Recognition. Finally we will end the lecture with a more detailed analysis of how the 2 leading collaborative agents TRIPS and COLLAGEN embody the theory presented along with a demo for each agent.

Air Travel Scheduling Scenario



Many people are familiar with the complexity and headaches associated with travel scheduling and ticket purchasing. Above is a scenario where a Boston based traveler wants to make a trip to Dallas, Phoenix, and San Francisco. There are some hard constraints such as he must be in Phoenix on Thursday between 11am and 3pm and return to Boston by Friday 5pm. There are also lose constraints and preferences such as he would like to leave Wednesday morning but can leave Tuesday night, and he would like to fly American Airlines as much as possible (but this is not a requirement).



If he visits a website such as Yahoo! Travel to plan his trip, one can immediately see the problem. The simple “to” and “from” fields along with date constraints as no where near sophisticated enough to solve this multi-destination trip with varying levels of constraints.

Problems with Travel Planning

- Order of actions may not be flexible
- Difficult to recover from mistakes
- Easy to get stuck or get lost
- May over or under constrain
- Lack of support for the user's problem solving process as it unfolds over time

9

Here are some of the common problems with travel planning systems, or even planners and schedulers in general.

1) Order of actions may not be flexible: The user may be required to enter the destination before the date and time information.

2) Difficult to recover from mistakes: Due to the inflexibility of actions changing the destination may erase other data already entered.

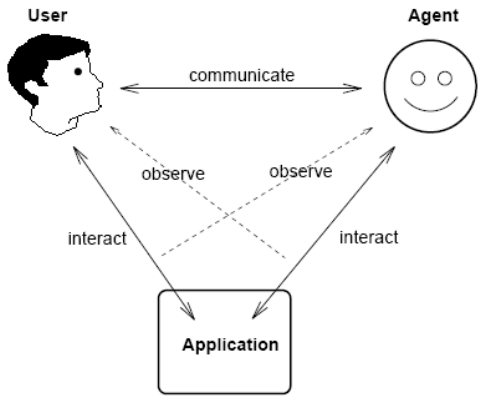
3) Easy to get stuck or get lost: After trying 20 or 30 combinations of departure and destination location and time, various airlines, perhaps even multiple websites, it would be very easy to loose track of the combinations that have been tried and the possible itineraries listed.

4) May over or under constrain: One can easily over or under constrain the problem and receive 100 possible itineraries or no itinerary at all.

5) And the main reason for all this confusion is that software like the one used on Yahoo! Travel lack support for the user's problem solving process as it unfolds over time: it does not keep track of what has been and what still needs to be accomplished.

Collaborative Interface Agent

- User and agent share a common goal
- Both parties work in close collaboration
- All relevant actions are either observed or reported.

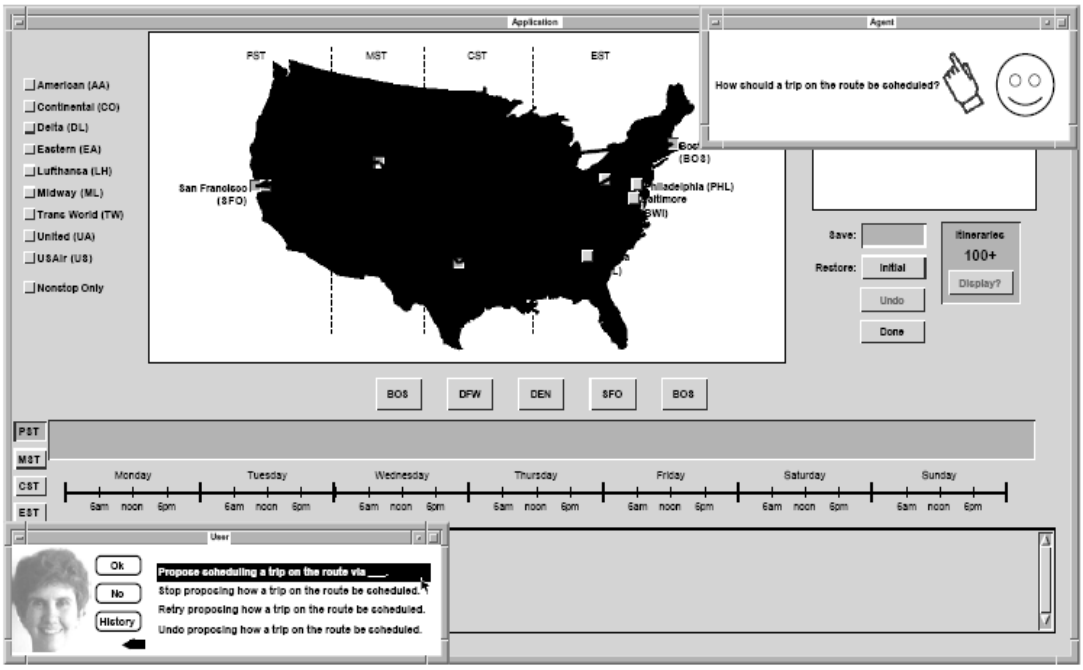


10

To solve these problems and make the planning process simpler, consider a collaborative agent in addition to the basic software application. Notice that aside from direct communication between user and agent, both the user and agent can observe the other's interaction with the specific application. With direct communication and observations both user and agent are fully aware of the other's actions and intentions; this is a property of such systems important to the theory that we will present.

We also want to emphasize that the topic of this lecture does not involve the design of the application on the bottom of the figure, but rather the collaborative agent on the right, which is modular and independent of the application.

Collaborative Travel Assistant



Here is an example of a travel assistant with a collaborative agent. The back ground is the actual travel planning application where the user can directly select options such as time, destination(s), and the airline. On the bottom left corner is the user interface window that reflects all verbal inputs of the users. And on the upper right corner is the dialogue box of the agent containing verbal communication from the agent. Both text displayed in the dialogue boxes and any direct actions that the user makes upon the application are considered by the agent through the entire planning process. Later, we will cover an example of such a transaction and how collaborative discourse and plan recognition analyze and extract relevant information from these transactions real-time.

Outline

- Travel Scheduling Example
- **Collaborative Discourse**
- Plan Recognition
- TRIPS
- COLLAGEN

12

As illustrated in the travel scheduling example above, current interactive automated agents lack a structured framework for interpreting and contributing to user goals, leading them to rely on restrictive, formulaic command sets and operation sequences.

A more sophisticated agent should engage in flexible discourse with the user, extracting and supplying information relevant to shared objectives. The challenge is to limit the overhead associated with interaction so that it does not outweigh the advantages of collaboration between agents of differing capabilities.

Here we outline a framework for collaborative discourse.

Elements of Collaborative Discourse

(Grosz and Sidner)

Intentional Structure

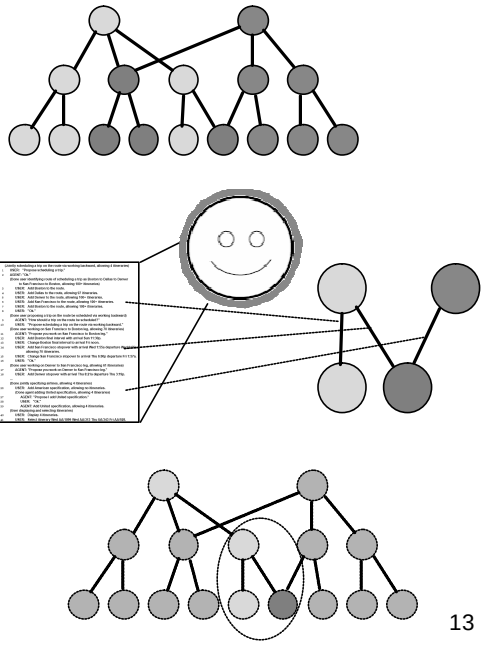
? model agent goals and intersections to identify collaboration opportunities

Linguistic Structure

? resolve agent interactions to build intentional model

Attentional Structure

? maintain immediate context to interpret agent interactions



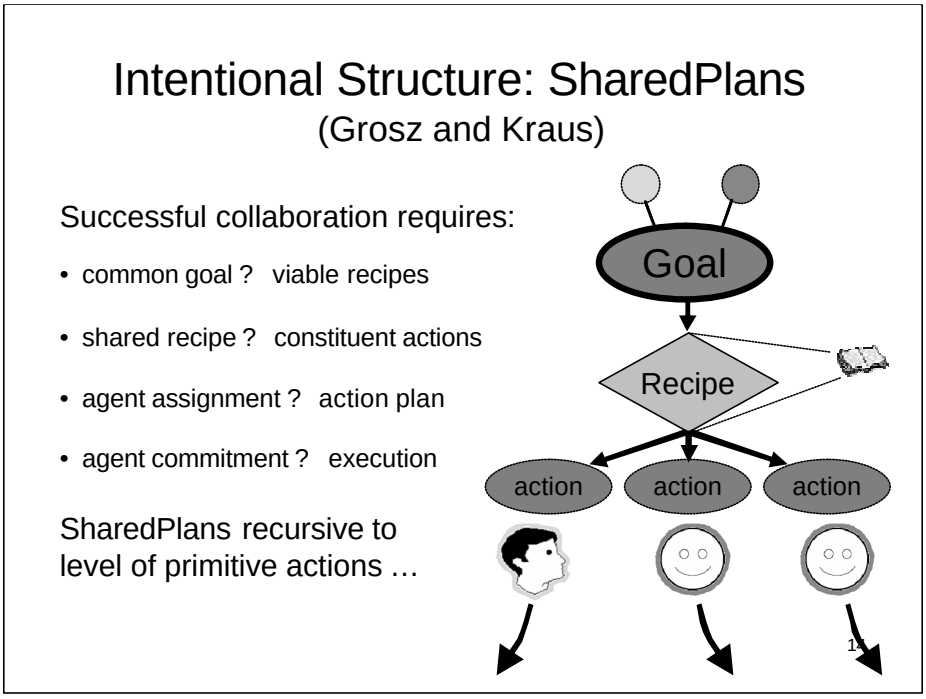
13

Grosz and Sidner propose three essential elements for collaborative discourse:

Intentional Structure. Agents must build a model of the goals of each other agent and how they may be decomposed into sub-goals. When goals or sub-goals intersect between multiple agents, agents should identify them as shared goals, so that their individual specialized capabilities can be used in collaboration.

Linguistic Structure. Agents must construct the intentional model by extracting relevant information from linguistic or observational dialogue. They must decompose dialog into distinct relevant segments of information.

Attentional Structure. Agents must interpret the relevance of dialog according to the flow of discourse: the same exchange in two different contexts may convey entirely different meanings with respect to the semantics of intention. They must maintain an immediate context in order to correctly interpret interactions.



Grosz and Kraus describe intentional structure by means of the SharedPlan formalism, which models the recursive decomposition of goals into sub-goals.

The figure illustrates two higher-level goals, one of the user (blue) and one of the automated agent (red), which share a constituent sub-goal (purple). This shared goal is further decomposed according to the following steps:

Identify common goal. Agents can then identify the possible recipes (viable sequences of constituent sub-goals) from a recipe library to achieve the shared goal.

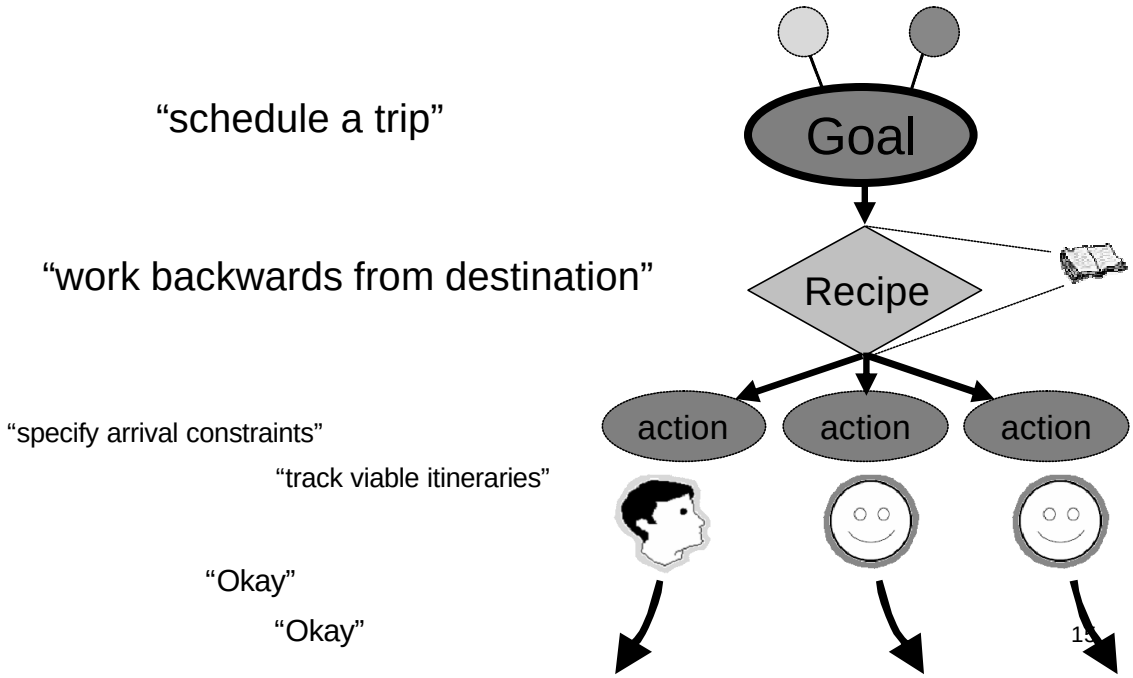
Select shared recipe. Agents agree upon a sequence of lower-level actions to be carried out between them to achieve the goal.

Assign agents to actions. Agents allocate individual efforts in carrying out different constituent actions according to their specialized capabilities.

Commit to collaboration. Agents agree to the shared agenda and commit to completing the sub-goals at subsequent levels through collaborative interaction.

SharedPlans exhibit a recursive structure: each sub-goal generated becomes a new goal presenting the opportunity for further collaboration. Goals are decomposed until they reach the level of so-called primitive actions, for which no recipes exist in the library.

Intentional Structure: SharedPlans
(Grosz and Kraus)



This illustrates the process of collaboration using the SharedPlan framework in the context of the travel scheduling example.

Here, the user and automated agent identify scheduling a trip as a shared goal.

They agree to construct the schedule by working backwards from the destination, one of the recipes known to the agent.

They partition the tasks so that the user (with knowledge of his schedule) specifies constraints on arrival and the agent (with knowledge of available flights) tracks the corresponding viable itineraries.

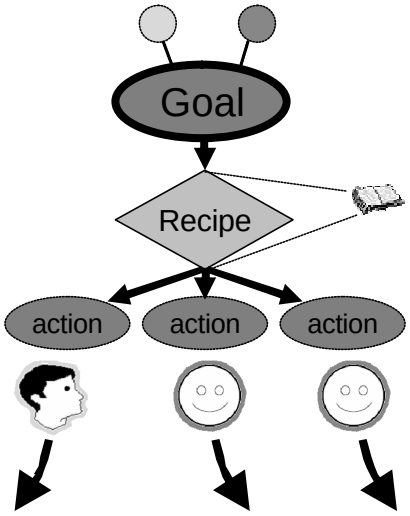
Finally, they agree to carry out the plan and continue the collaboration.

Linguistic Structure: Segmentation
(Grosz; Lochbaum)

Interactions identified by purpose:

- directly achieve current goal
- identify recipe to be used for goal
- achieve step in recipe for goal
- specify parameter of step/goal
- identify agent to perform step/goal

Hierarchy used to track content and context of discourse …



16

Grosz and Lochbaum describe the essential principle of dialog decomposition: communications and observed actions are grouped into segments corresponding to a specific purpose. Segments are grouped into a recursive hierarchy paralleling the intentional structure of the SharedPlan.

Relevant bits of information from these segments may be incorporated into the intentional structure in any order. Segments may be relevant in five basic ways, wherein they:

•declare the completion of the current goal;

•identify the recipe to be used for the current goal;

•declare the completion of a step in the recipe for the current goal;

•specify a parameter of a step in the recipe or the current goal; or

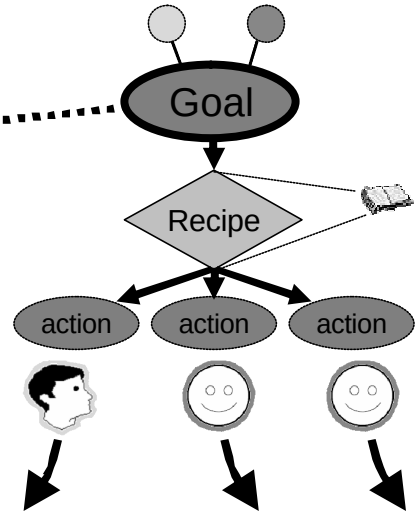
•identify the agent responsible for performing a step in the recipe or the current goal.

Segments not relevant to the current goal are known as interrupts, and generate new top-level goals in the intentional structure and the corresponding segment hierarchy.

Linguistic Structure: Segmentation
(Grosz; Lochbaum)

Interactions identified by purpose:

- directly achieve current goal
- identify recipe to be used for goal
- achieve step in recipe for goal
- specify parameter of step/goal
- identify agent to perform step/goal

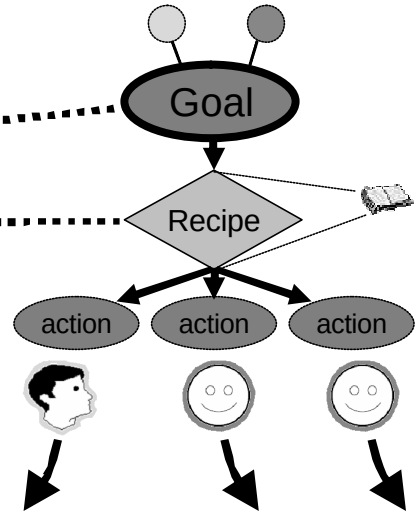


Hierarchy used to track content and context of discourse …

Linguistic Structure: Segmentation
(Grosz; Lochbaum)

Interactions identified by purpose:

- directly achieve current goal
- identify recipe to be used for goal
- achieve step in recipe for goal
- specify parameter of step/goal
- identify agent to perform step/goal



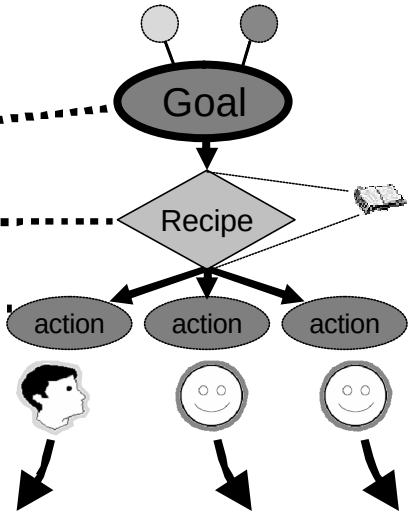
Hierarchy used to track content and context of discourse …

18

Linguistic Structure: Segmentation
(Grosz; Lochbaum)

Interactions identified by purpose:

- directly achieve current goal
- identify recipe to be used for goal
- achieve step in recipe for goal
- specify parameter of step/goal
- identify agent to perform step/goal



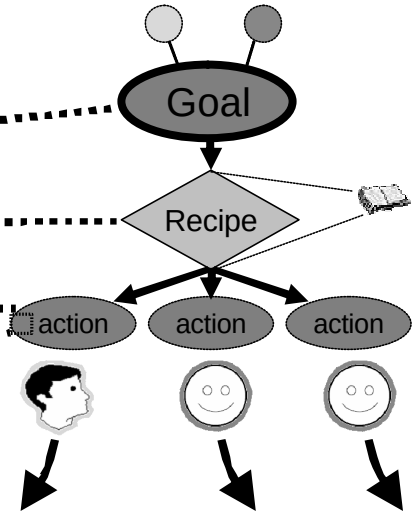
Hierarchy used to track content and context of discourse …

19

Linguistic Structure: Segmentation
(Grosz; Lochbaum)

Interactions identified by purpose:

- directly achieve current goal
- identify recipe to be used for goal
- achieve step in recipe for goal
- specify parameter of step/goal
- identify agent to perform step/goal



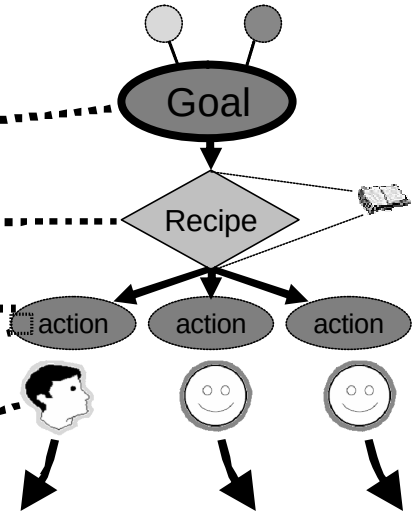
Hierarchy used to track content and context of discourse …

20

Linguistic Structure: Segmentation
(Grosz; Lochbaum)

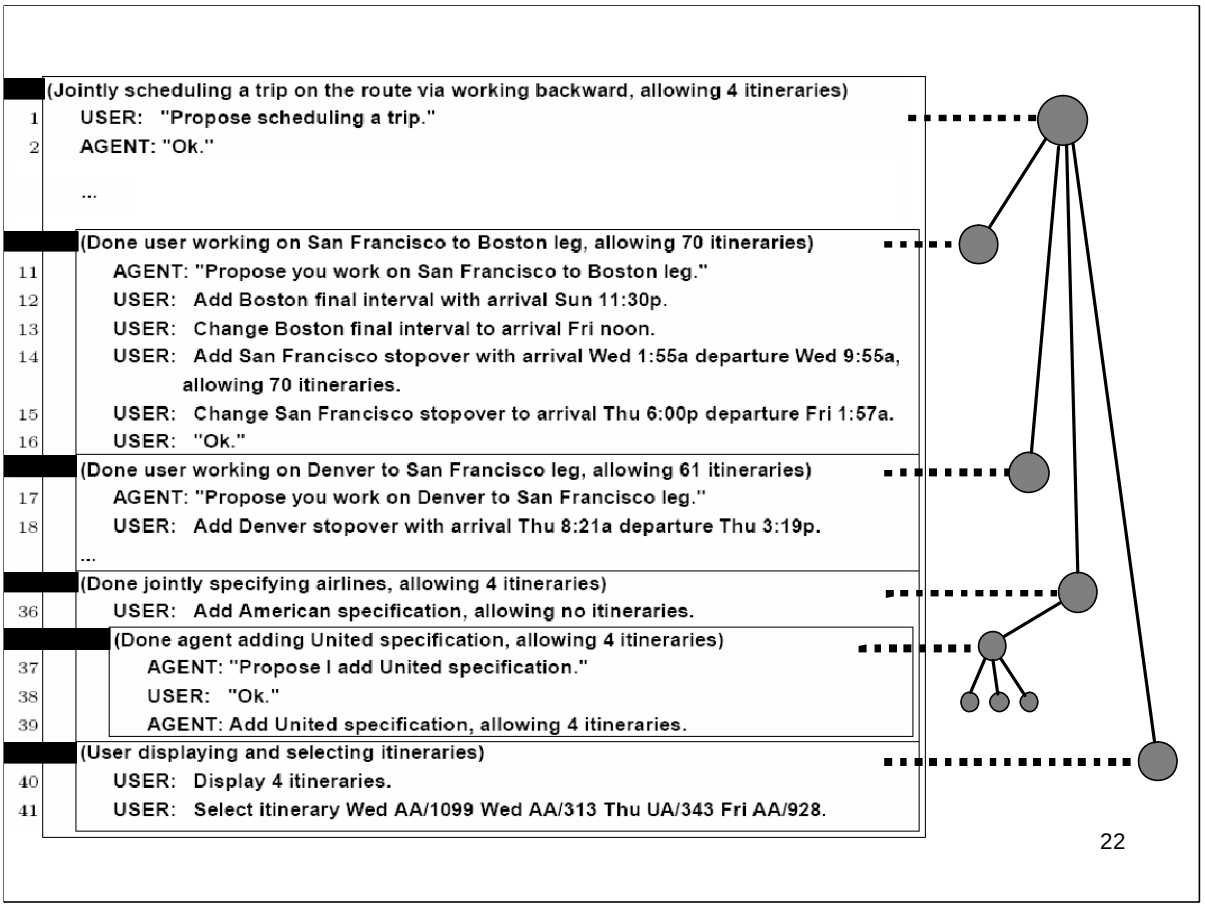
Interactions identified by purpose:

- directly achieve current goal
- identify recipe to be used for goal
- achieve step in recipe for goal
- specify parameter of step/goal
- identify agent to perform step/goal



Hierarchy used to track content
and context of discourse …

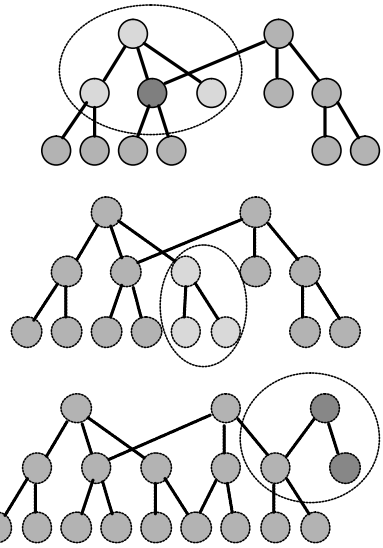
21



The figure shows the purpose-driven segmentation of some dialog between a user and the collaborative travel scheduling agent presented earlier. The first line of each segment (box) states the purpose with which the segment is identified.

The segments are interpreted with relevance to goals at different levels in the hierarchy: the top level segment references the overall goal of scheduling a trip; one of the next-level segments references the sub-goal of specifying an airline; and the lowest-level segment shown references the sub-sub-goal of selecting United Airlines.

Attentional Structure: Focus Stack
(Lochbaum)



Tracks current context of discourse:

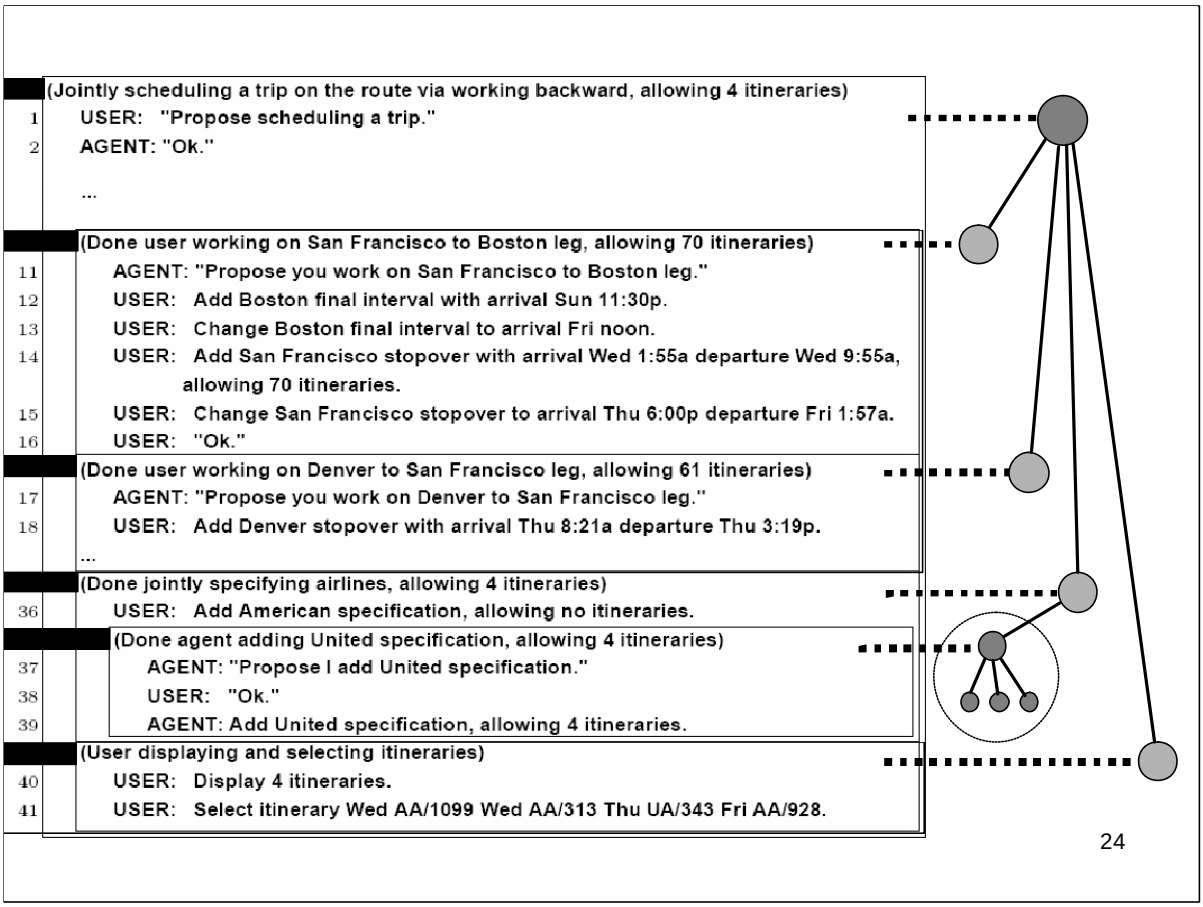
- maintains active path in segment hierarchy
- goal decomposition increases depth ; goal completion decreases depth
- goal abandonment (interrupt) generates new top-level focus

Restricted context mitigates worst-case exponential complexity of general plan recognition

23

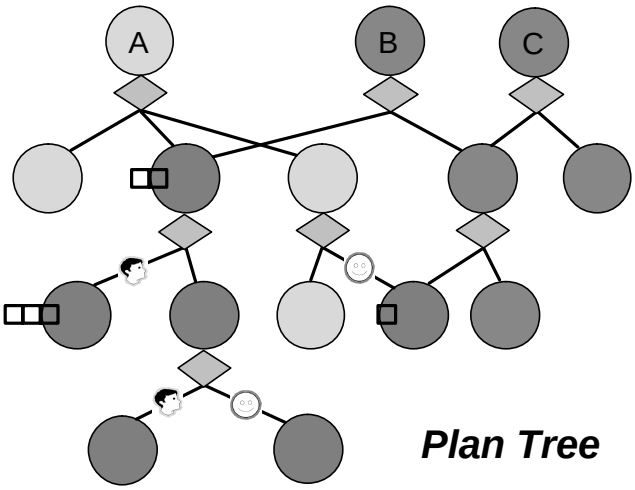
As seen in the previous example, interpreting discourse requires using the immediate context of the dialog to differentiate meaning. Even if meanings could be determined without context, this would require astronomical computation, since segments would require interpretation with respect to every possible element of the joint intention structure. The immediate context of the conversation makes interpretation feasible in both respects.

Attentional structure is maintained by the focus stack, a stored path in the segment hierarchy identifying the current context of the discourse. When goals are decomposed into sub-goals, the depth of this path increases: the newly generated sub-goals become the current focus (top). When sub-goals are completed, the focus jumps upward to the parent goals in progress, decreasing the depth of the stack (middle). When the current goal is abandoned via an interrupt, the focus jumps to the newly generated top-level goal, which may eventually become linked with the rest of the intentional structure (bottom).

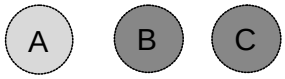


The figure illustrates the focus stack at a particular point in the travel scheduling dialog seen above. Here the relevant path in the hierarchy of goals and segments is highlighted: the focus stack informs the agent that current dialog references airline selection, which allows the agent to propose the choice of United Airlines based on its information about itineraries.

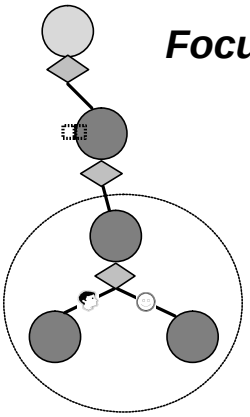
Discourse State Representation



Plan Tree



History List



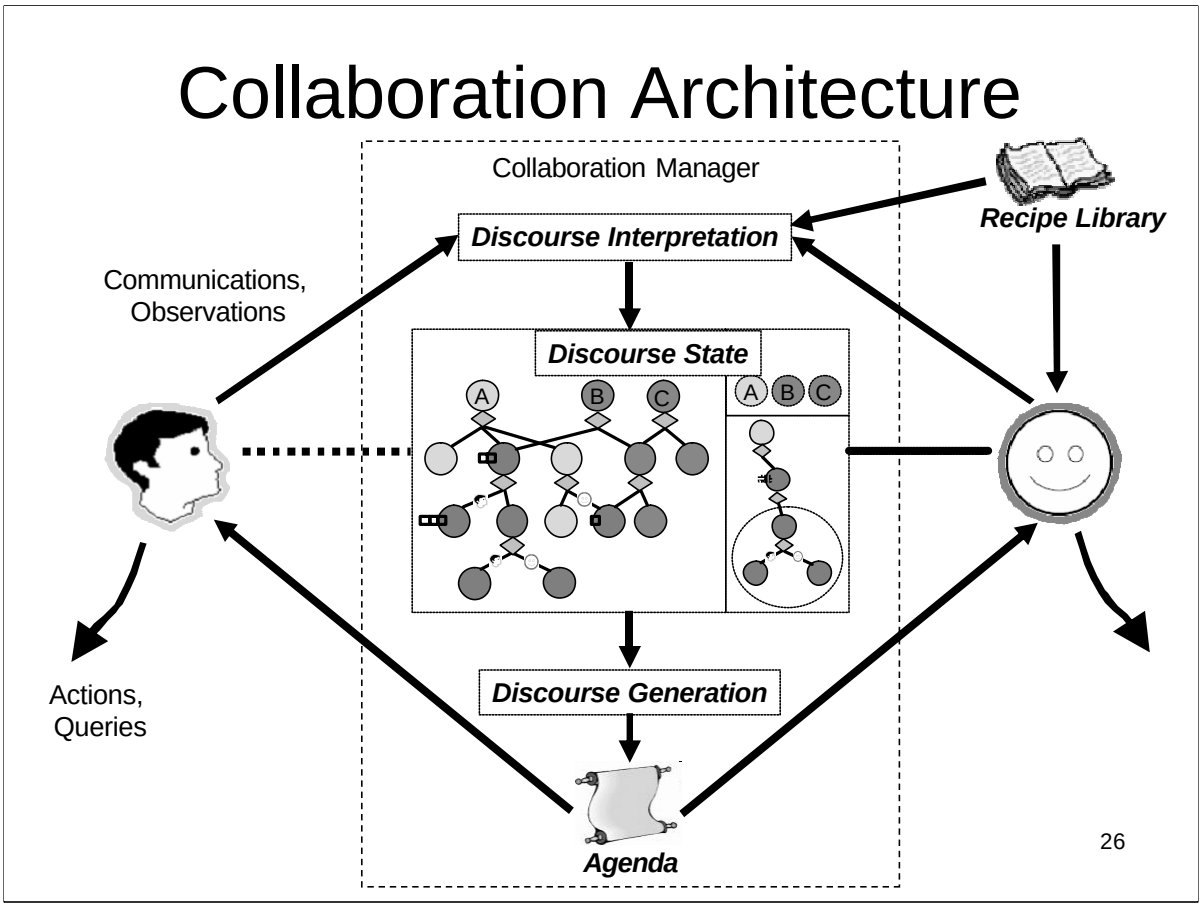
Focus Stack

History list maintains top-level focus segments for subsequent reference during discourse

25

We saw earlier that interrupts (segments irrelevant to the immediate context) can generate new top-level goals in the intentional structure, whose relationship with the rest of the structure may not be readily apparent. Because they start out as disconnected elements, the system requires one additional component: a history list to keep track of top-level nodes in the structure. This prevents loss of information gathered from prior collaboration by maintaining references to the entire collaboration history.

The plan tree (SharedPlan), focus stack, and history list comprise the discourse state representation maintained by the collaboration manager.



Collaborative agents like our later example, COLLAGEN, use a multi-agent collaboration architecture like the one illustrated above. Shown here for a human user and automated agent, the architecture generalizes naturally to many agents.

The discourse interpretation process accepts communications and observations from the agents along with recipes from a library (known to one or more agents) to construct the discourse state representation. The discourse generation (inverse) process generates an agenda of communications and actions (often in the form of decision points) for the agents with which to continue their collaboration. The agents themselves perform individual actions and query individual sources of information in accordance with their SharedPlans. Finally, while the automated agent can retrieve a complete view of the discourse state (by accessing the data structure), human users can obtain a partial view by examining the real-time segment hierarchy generated by the dialog stream, presented in a natural interface.

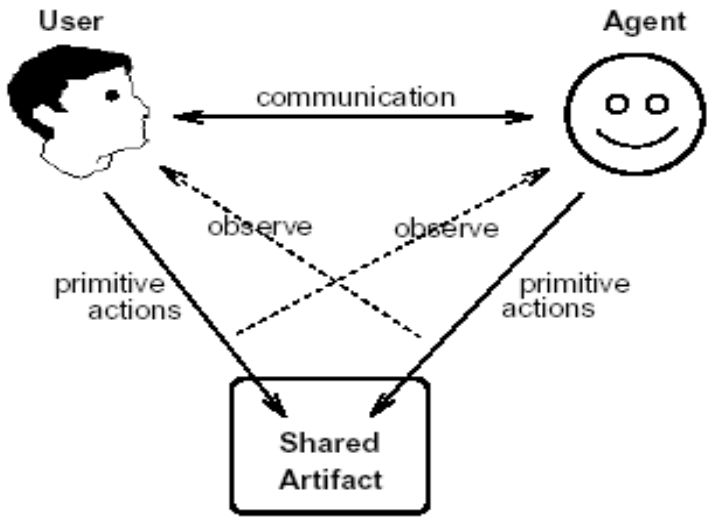
Collaborative Discourse: Recap

- Collaboration manager uses goal-based model to interpret and generate discourse, interleaving planning and execution
- Worst-case exponential plan recognition avoided by limited immediate focus
- Localization ensures small number of communication choices, immediately relevant and manageable for the user
- Future work may pursue higher-level representation of parallel tasks to avoid excessive interrupt behavior

Outline

- Travel Scheduling Example
- Collaborative Discourse
- **Plan Recognition**
- TRIPS
- COLLAGEN

Setting for Collaboration



29

Two participants can both communicate with each other and observe each other's actions on some shared artifact. Sometimes the communication can be reduced by more observations.

COLLAGEN E-mail Example

	Without plan recognition		*With plan recognition*
1	USER: "Let's work on email."	USER:	"Let's work on email."
2	AGENT: "You have 5 unread messages."	AGENT:	"You have 5 unread messages."
3	USER: "Let's read a message."		
4	USER: <Select message from Chuck>	USER:	<Select message from Chuck>
5	USER: "Let's read message in its own window."		
6	USER: <Open message>	USER:	<Open message>
7	USER: "Let's do something using message."		
8	USER: <Visit http://foo.bar.com>	USER:	<Visit http://foo.bar.com>
9	...long lunch...		...long lunch...
10	USER: "What was I doing?"	USER:	"What was I doing?"
11	AGENT: "Working on email.	AGENT:	"Working on email.
12	Reading message from Chuck.		Reading message from Chuck.
13	Reading message in its own window.		Reading message in its own window.
14	Done something using message.		Done something using message.
15	<Visit http://foo.bar.com>		<Visit http://foo.bar.com>
16	Expecting to react to message (opt)."		Expecting to react to message (opt)."
17	USER: "Let's react to message."		
18	USER: "Let's send email."		
19	USER: <Start new message>	USER:	<Start new message>
20	USER: "Let's fill in message."		
21	USER: <Address message to Dan>	USER:	<Address message to Dan>
22		AGENT:	"Why are you sending email to Dan?
23			(a) reacting to message from Chuck
24			(b) as a new goal "
25		USER:	"I am reacting to message from Chuck."
26	...additional editing actions...		...additional editing actions...
27	USER: "Done filling in message."	USER:	"Done filling in message."
28	AGENT: "Let's send, cancel or save message."	AGENT:	"Let's send, cancel or save message."
29	USER: "Please save message."	USER:	"Please save message."
30	AGENT: <Save message>	AGENT:	<Save message>

30

Here is an example where a collaborative agent assists a user in checking e-mail. On the left is a transcript of the interaction between the user and an agent that does not use plan recognition, and on the right is a transcript of the interaction between the user and an agent that does use plan recognition. As can be seen from the boxed portion of the interaction, plan recognition helps the agent to identify the user's intents without requiring the user to specify through verbal communication each primitive action that the user takes.

COLLAGEN E-mail Example

Without plan recognition		*With plan recognition*	
1 USER:	"Let's work on email."	USER:	"Let's work on email."
2 AGENT:	"You have 5 unread messages."	AGENT:	"You have 5 unread messages."
3 USER:	"Let's read a message."		
4 USER:	<Select message from Chuck>	USER:	<Select message from Chuck>
5 USER:	"Let's read message in its own window."		
6 USER:	<Open message>	USER:	<Open message>
7 USER:	"Let's do something using message."		
8 USER:	<Visit http://foo.bar.com>	USER:	<Visit http://foo.bar.com>
9	...long lunch...		...long lunch...
10 USER:	"What was I doing?"	USER:	"What was I doing?"
11 AGENT:	"Working on email.	AGENT:	"Working on email.
12	Reading message from Chuck.		Reading message from Chuck.
13	Reading message in its own window.		Reading message in its own window.
14	Done something using message.		Done something using message.
15	<Visit http://foo.bar.com>		<Visit http://foo.bar.com>
16	Expecting to react to message (opt)."		Expecting to react to message (opt)."
17 USER:	"Let's react to message."		
18 USER:	"Let's send email."		
19 USER:	<Start new message>	USER:	<Start new message>
20 USER:	"Let's fill in message."		
21 USER:	<Address message to Dan>	USER:	<Address message to Dan>
22		AGENT:	"Why are you sending email to Dan?
23			(a) reacting to message from Chuck
24			(b) as a new goal "
25		USER:	"I am reacting to message from Chuck."
26	...additional editing actions...		...additional editing actions...
27 USER:	"Done filling in message."	USER:	"Done filling in message."
28 AGENT:	"Let's send, cancel or save message."	AGENT:	"Let's send, cancel or save message."
29 USER:	"Please save message."	USER:	"Please save message."
30 AGENT:	<Save message>	AGENT:	<Save message>

31

This boxed portion shows a couple of things:

1. The agent's knowledge of the user's actions is identical in both cases, i.e. in the case without plan recognition where the user had to verbally specify all primitive actions, and in the case with plan recognition where the agent learns of the user's intent by observations of the user's actions.
2. In both cases the agent is able to regurgitate the history of the user's actions. This is very useful for the user after taking a break (eg. "long lunch") from the continuum of working.

COLLAGEN E-mail Example

Without plan recognition		*With plan recognition*	
1 USER:	"Let's work on email."	USER:	"Let's work on email."
2 AGENT:	"You have 5 unread messages."	AGENT:	"You have 5 unread messages."
3 USER:	"Let's read a message."		
4 USER:	<Select message from Chuck>	USER:	<Select message from Chuck>
5 USER:	"Let's read message in its own window."		
6 USER:	<Open message>	USER:	<Open message>
7 USER:	"Let's do something using message."		
8 USER:	<Visit http://foo.bar.com>	USER:	<Visit http://foo.bar.com>
9	...long lunch...		...long lunch...
10 USER:	"What was I doing?"	USER:	"What was I doing?"
11 AGENT:	"Working on email.	AGENT:	"Working on email.
12	Reading message from Chuck.		Reading message from Chuck.
13	Reading message in its own window.		Reading message in its own window.
14	Done something using message.		Done something using message.
15	<Visit http://foo.bar.com>		<Visit http://foo.bar.com>
16	Expecting to react to message (opt)."		Expecting to react to message (opt)."
17 USER:	"Let's react to message."		
18 USER:	"Let's send email."		
19 USER:	<Start new message>	USER:	<Start new message>
20 USER:	"Let's fill in message."		
21 USER:	<Address message to Dan>	USER:	<Address message to Dan>
22		AGENT:	"Why are you sending email to Dan?
23			(a) reacting to message from Chuck
24			(b) as a new goal "
25		USER:	"I am reacting to message from Chuck."
26	...additional editing actions...		...additional editing actions...
27 USER:	"Done filling in message."	USER:	"Done filling in message."
28 AGENT:	"Let's send, cancel or save message."	AGENT:	"Let's send, cancel or save message."
29 USER:	"Please save message."	USER:	"Please save message."
30 AGENT:	<Save message>	AGENT:	<Save message>

32

Agents that act with plan recognition learns the user's intent by observing the user's actions, but sometimes the user's actions may be ambiguous to the agent. For example, in this case, the user starts a new message, but the user may do this for a variety of reasons:

1. To start a new thread of messages,
2. To reply someone else's thread.

COLLAGEN E-mail Example

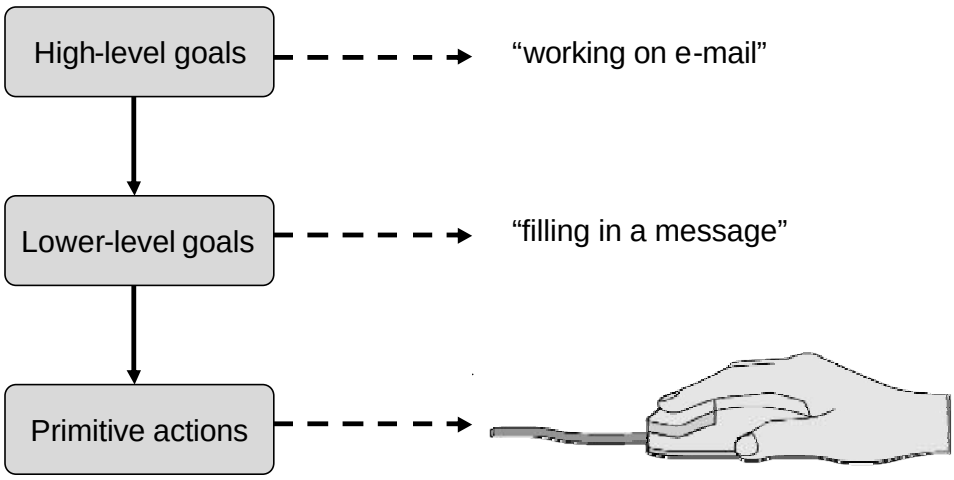
Without plan recognition		*With plan recognition*	
1 USER:	"Let's work on email."	USER:	"Let's work on email."
2 AGENT:	"You have 5 unread messages."	AGENT:	"You have 5 unread messages."
3 USER:	"Let's read a message."		
4 USER:	<Select message from Chuck>	USER:	<Select message from Chuck>
5 USER:	"Let's read message in its own window."		
6 USER:	<Open message>	USER:	<Open message>
7 USER:	"Let's do something using message."		
8 USER:	<Visit http://foo.bar.com>	USER:	<Visit http://foo.bar.com>
9	...long lunch...		...long lunch...
10 USER:	"What was I doing?"	USER:	"What was I doing?"
11 AGENT:	"Working on email.	AGENT:	"Working on email.
12	Reading message from Chuck.		Reading message from Chuck.
13	Reading message in its own window.		Reading message in its own window.
14	Done something using message.		Done something using message.
15	<Visit http://foo.bar.com>		<Visit http://foo.bar.com>
16	Expecting to react to message (opt)."		Expecting to react to message (opt)."
17 USER:	"Let's react to message."		
18 USER:	"Let's send email."		
19 USER:	<Start new message>	USER:	<Start new message>
20 USER:	"Let's fill in message."		
21 USER:	<Address message to Dan>	USER:	<Address message to Dan>
22		AGENT:	"Why are you sending email to Dan?
23			(a) reacting to message from Chuck
24			(b) as a new goal "
25		USER:	"I am reacting to message from Chuck."
26	...additional editing actions...		...additional editing actions...
27 USER:	"Done filling in message."	USER:	"Done filling in message."
28 AGENT:	"Let's send, cancel or save message."	AGENT:	"Let's send, cancel or save message."
29 USER:	"Please save message."	USER:	"Please save message."
30 AGENT:	<Save message>	AGENT:	<Save message>

33

Thus in these ambiguous circumstances, the agent must ask the user for clarification.

Notice that this interaction is unnecessary in the case without plan recognition because the user already specified his/her intent. However, plan recognition is more advantageous because generally, ambiguous actions are not frequent, and the amount of verbal communications that can be omitted by using plan recognition is greater than the amount of clarification questions necessary.

Task Structure Model



34

The task structure is a hierarchy of goals and actions, where high-level goals such as working on e-mail breaks down to lower-level goals such as preparing a message. At the bottom of the hierarchy are primitive actions such as clicking a mouse or typing in text.

Air Travel Example

1 USER: "Propose scheduling a trip."
2 AGENT: "Ok."

High-level goal

(Done user identifying route of scheduling a trip as Boston to Dallas to Denver to San Francisco to Boston, allowing 100+ itineraries)

3 USER: <Add Boston to the route.>
4 USER: <Add Dallas to the route, allowing 97 itineraries.>
5 USER: <Add Denver to the route, allowing 100+ itineraries.>
6 USER: <Add San Francisco to the route, allowing 100+ itineraries.>
7 USER: <Add Boston to the route, allowing 100+ itineraries.>
8 USER: "Ok."

(Done user proposing a trip on the route be scheduled via working backward)

Low-level goal

9 AGENT: "How should a trip on the route be scheduled?"
10 USER: "Propose scheduling a trip on the route via working backward."

(Done user working on San Francisco to Boston leg, allowing 70 itineraries)

11 AGENT: "Propose you work on San Francisco to Boston leg."
12 USER: <Add Boston final interval with arrival Sun 11:30p.>
13 USER: <Change Boston final interval to arrival Fri noon.>
14 USER: <Add San Francisco stopover with arrival Wed 1:55a departure Wed 9:55a, allowing 70 itineraries.>
15 USER: <Change San Francisco stopover to arrival Thu 6:00p departure Fri 1:57a.>
16 USER: "Ok."

...

35

Here we see the air travel scheduling example in the context of task structure.

The top-level goal is to schedule a trip from Boston to Dallas to Denver to San Francisco and back to Boston. The first box contains the actions that the user takes to identify the high-level goals. The second box contains the actions for accomplishing a lower-level goal such as planning the travel leg from San Francisco to Boston.

Air Travel Example

...

(Done user working on Denver to San Francisco leg, allowing 61 itineraries)
17 AGENT: "Propose you work on Denver to San Francisco leg."
18 USER: <Add Denver stopover with arrival Thu 8:21a departure Thu 3:19p.>
...
(Done jointly specifying airlines, allowing 4 itineraries)
36 USER: <Add American specification, allowing no itineraries.>
(Done agent adding United specification, allowing 4 itineraries)
37 AGENT: "Propose I add United specification."
38 USER: "Ok."
39 AGENT: <Add United specification, allowing 4 itineraries.>
(User displaying and selecting itineraries)
40 USER: <Display 4 itineraries.>
41 USER: <Select itinerary Wed AA/1099 Wed AA/313 Thu UA/343 Fri AA/928.>

36

Notice that the lower-level goals can be performed by either the user or in the case shown in the box, they can be performed by the agent as well.

With Plan Recognition

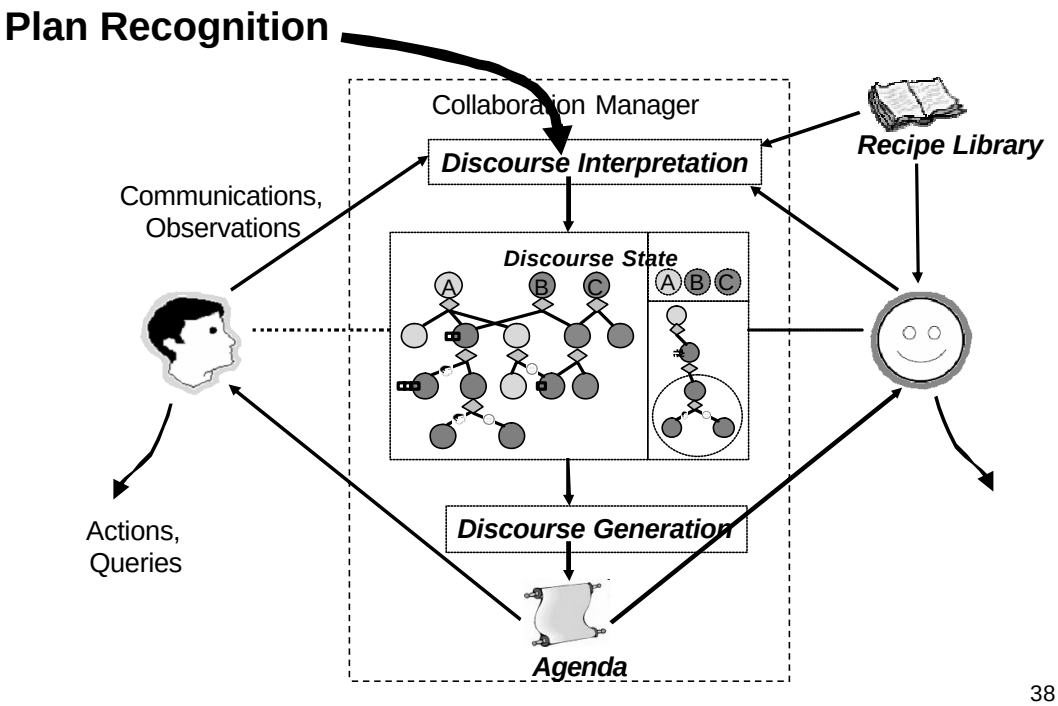
- User need not announce each goal
- Intentions conveyed through actions
- Verbal clarification resolves ambiguities

37

To summarize what we have seen thus far in the e-mail example and the air travel scheduling example:

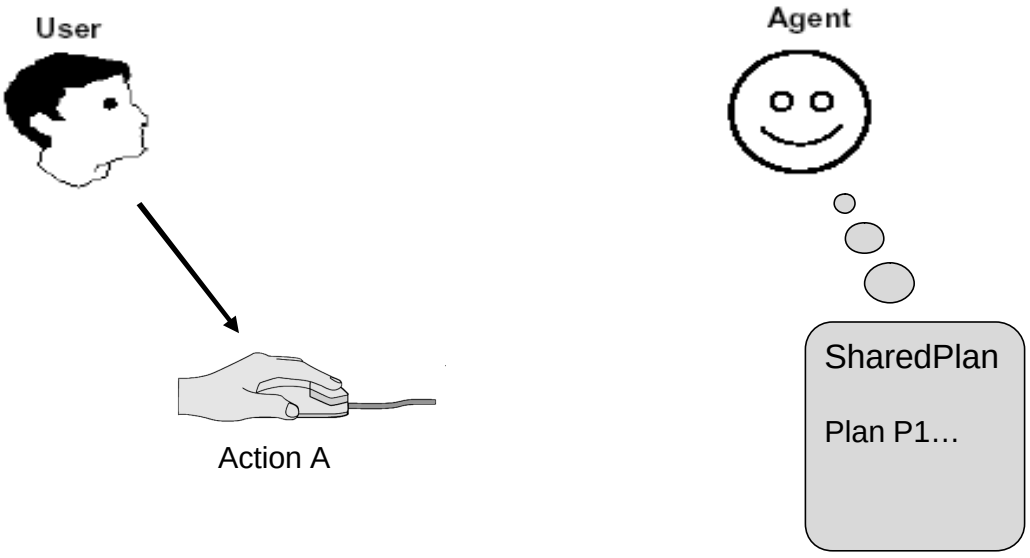
Plan recognition allows the user more flexibility in performing the actions that lead to goals by not requiring the user to verbally communicate to the agent every goal that the user is pursuing. The agent will derive the user's goals by observing the user's actions. If there are instances where the user's actions are unclear or may imply multiple possible user goals, then the agent can ask the user for verbal clarification.

Collaboration Architecture



Within the overall collaboration architecture, plan recognition is applied in Discourse Interpretation.

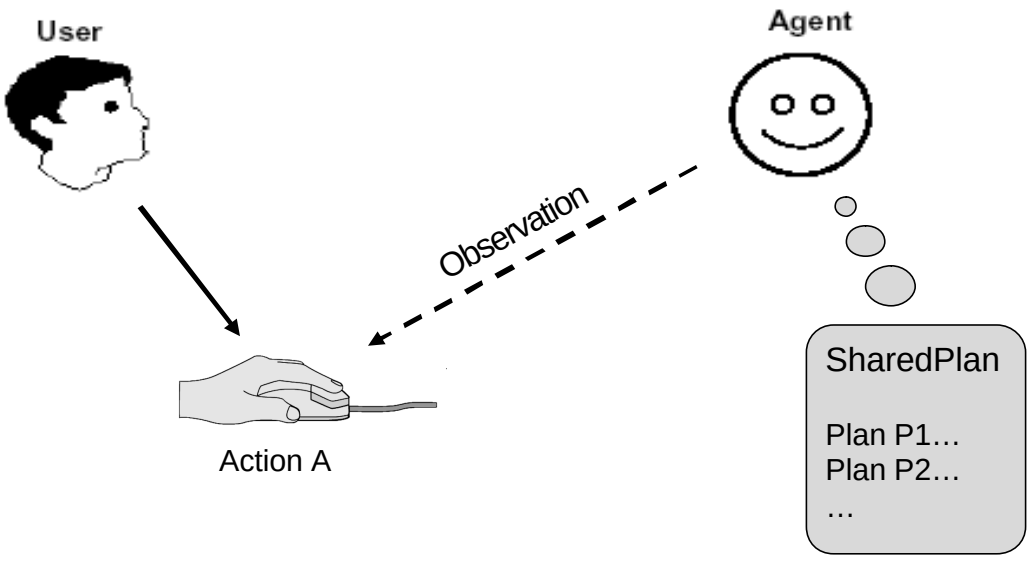
Plan Recognition Overview



39

During plan recognition, the user performs some action A. Based on previous observations, the agent maintains a SharedPlan in its knowledge space.

Plan Recognition Overview

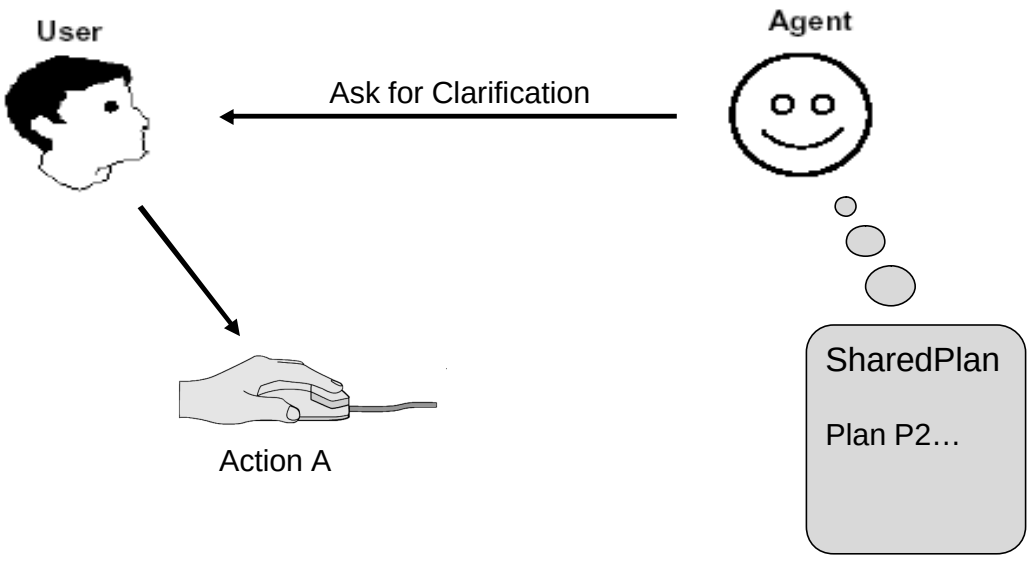


40

Suppose the agent observes the user perform the action A.

Plan recognition identifies the set of possible extensions to the agent's current SharedPlan which are consistent with its recipe knowledge and contains the user performing A.

Plan Recognition Overview



41

Multiple extensions require clarification to reduce the SharedPlan to a more manageable domain size.

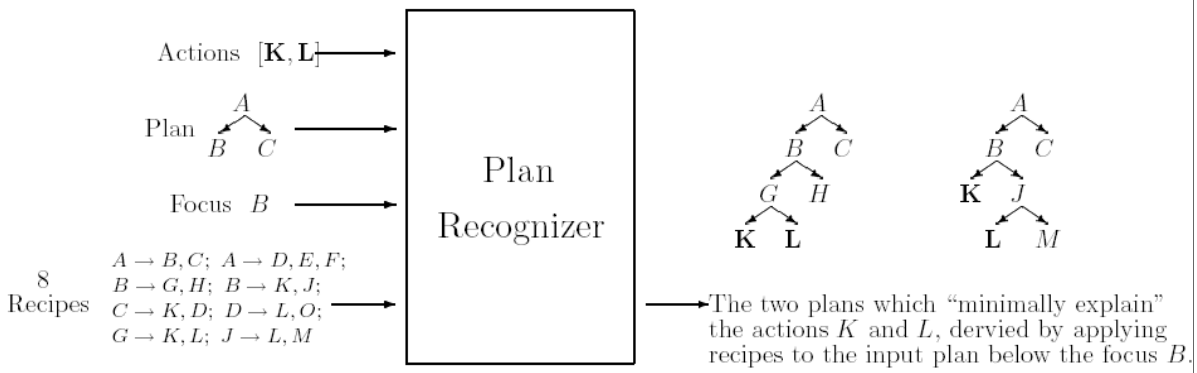
Properties of Collaborative Setting

- Focus of attention
- Incremental extension on a plan
- Clarification reduces recognizer interpretations

42

In a collaborative setting, the agent and user share a focus of attention on some specific action. The plans that are generated are dynamic in that they are incrementally extended when necessary. This differs from most existing forms of human-computer interaction, where the computer executes a predefined plan of actions, and if some step of the plan fails, then the planning must start over. Within dynamic plan generation, the agent must keep track of the possible goals that the user is trying to pursue at each step. To keep the list of possible goals small and tractable, the agent may ask for clarifications.

Plan Recognition Example



43

In this example, the user sees new input actions K and L from the user. The current plan has A -> B and C. The focus of attention is on action B. The recipe library holds 8 recipes that may be helpful in expanding the plan.

The plan recognizer performs an exhaustive search of all possible ways of extending the input plan to explain the input actions.

The plan recognizer takes the current plan, and expands it on the focus action B using all the necessary recipes. It will stop expanding once it reaches an explanation for a target (input) action. The two plans on the right goes from the current plan to the target actions via B using the least amount of actions.

Algorithm Setup

Plan $P =< A, E, C >$

- $A \equiv$ set of actions $\{A_1, \ldots, A_n\}$
- $E \equiv$ set of directed acyclic edges $\{E_1, \ldots, E_m\}$
  - The edge $A_i \rightarrow A_j$ indicates A_j is a step in achieving A_i
- $C \equiv$ set of constraints $\{C_1, \ldots, C_k\}$

$ACT \equiv$ set of all actions

- $PRIM \subseteq ACT$
- $TOP \subseteq ACT$

$RECIPE \equiv$ set of recipes, where each R_k is a function from a non-primative action A_i to plan $< A', E', C' >$

- A_i is in A'
- For every $A_j \neq A_i$ in A' there is an edge $A_i \rightarrow A_j$ in E'

44

To set up the algorithm, we need to define a plan as a tuple of a set of actions, edges and constraints. We also define the set of all actions which are divided into primitive and top level actions. We then define a set of recipes that map non-primitive actions to plans.

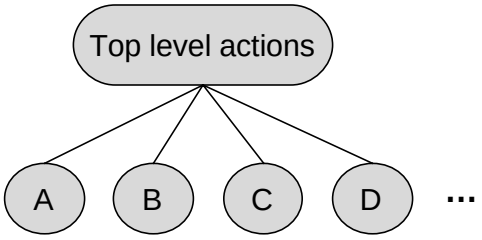
Curly letters represents set values; normal variables are single elements.

Plan Recognition Algorithm

(a) Plan recognition.

RECOGNIZE($[A_1, .., A_n], P, f, \mathcal{R}$) $\equiv$
$\mathcal{EXPL} \leftarrow \emptyset, \mathcal{Q} \leftarrow \emptyset$
if $P = \langle \emptyset, \emptyset, \emptyset \rangle$
  foreach $T_i \in \mathcal{TOP}$
    add $\langle [A_1, .., A_n], \langle \{T_i\}, \emptyset, \emptyset \rangle, T_i \rangle$ to $\mathcal{Q}$
else foreach $g_i \in$ FRINGE(P, f)
  add $\langle [A_1, .., A_n], P, g_i \rangle$ to $\mathcal{Q}$
until $\mathcal{Q} = \emptyset$
  remove $\langle [A'_1, .., A'_{n'}], P', act \rangle$ from $\mathcal{Q}$
  $P'' \leftarrow$ REPLACE(P', act, A'_1)
  if CONSISTENT?(P'')
    if $n' = 1$ add P'' to $\mathcal{EXPL}$
    else foreach $g_i \in$ FRINGE(P'', f)
      add $\langle [A'_2, .., A'_{n'}], P'', g_i \rangle$ to $\mathcal{Q}$
  if $act \notin \mathcal{PRIM}$
    foreach recipe $R_k \in \mathcal{R}$
      $P''' \leftarrow$ EXTEND(P', R_k, act)
      foreach s_j, where $act \rightarrow s_j \in P'''$
        add $\langle [A'_1, .., A'_{n'}], P''', s_j \rangle$ to $\mathcal{Q}$
return $\mathcal{EXPL}$

[A1,… An], P (0,0,0)



Q:
{*[A1,… An],* ({A}, 0, 0), A},
{*[A1,… An],* ({B}, 0, 0), B},
{*[A1,… An],* ({C}, 0, 0), C},
{*[A1,… An],* ({D}, 0, 0), D},
…

45

The plan recognition function is called with a list of input actions [A1, … An], some existing plan P, a focus action f, and a recipe library R.

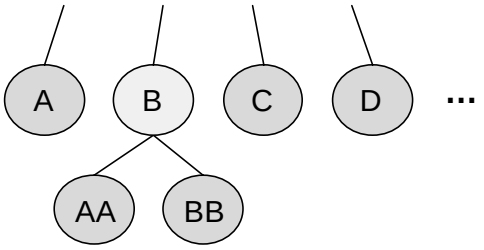
When plan recognition first starts off, there are no plans, so it will add all of the top level actions to the queue (Q).

Plan Recognition Algorithm

(a) Plan recognition.

RECOGNIZE($[A_1, .., A_n]$,P,f,$\mathcal{R}$) ≡
$\mathcal{EXPL} \leftarrow \emptyset, \mathcal{Q} \leftarrow \emptyset$
if $P = \langle \emptyset, \emptyset, \emptyset \rangle$
  foreach $T_i \in \mathcal{TOP}$
    add $\langle [A_1, .., A_n], \langle \{T_i\}, \emptyset, \emptyset \rangle, T_i \rangle$ to $\mathcal{Q}$
else foreach $g_i \in$ FRINGE(P, f)
  add $\langle [A_1, .., A_n], P, g_i \rangle$ to $\mathcal{Q}$
until $\mathcal{Q} = \emptyset$
  remove $\langle [A'_1, .., A'_{n'}], P', act \rangle$ from $\mathcal{Q}$
  $P'' \leftarrow$ REPLACE(P', act, A'_1)
  if CONSISTENT?(P'')
    if $n' = 1$ add P'' to $\mathcal{EXPL}$
    else foreach $g_i \in$ FRINGE(P'', f)
      add $\langle [A'_2, .., A'_{n'}], P'', g_i \rangle$ to $\mathcal{Q}$
  if $act \notin \mathcal{PRIM}$
    foreach recipe $R_k \in \mathcal{R}$
      $P''' \leftarrow$ EXTEND(P', R_k, act)
      foreach s_j, where $act \rightarrow s_j \in P'''$
        add $\langle [A'_1, .., A'_{n'}], P''', s_j \rangle$ to $\mathcal{Q}$
return $\mathcal{EXPL}$

[A1,… An], P (A,E,C), f = B



Q:

{*[A1,… An],* ({AA, B}, *E*, *C*), AA},

{*[A1,… An],* ({BB, B}, *E*, *C*), BB},

{*[A1,… An],* ({A}, 0, 0), A},

{*[A1,… An],* ({B}, 0, 0), B},

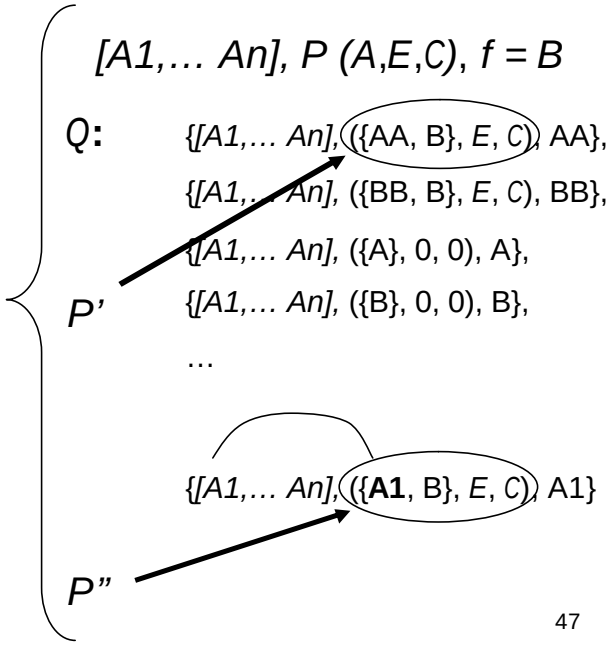
…

When performing plan recognition with some plans already in place, we will choose the focus action (in this case action B shown in yellow), and expand on the focus action, adding the leaves (AA and BB) to the queue as shown.

Plan Recognition Algorithm

(a) Plan recognition.

RECOGNIZE($[A_1, .., A_n], P, f, \mathcal{R}$) $\equiv$
$\mathcal{EXPL} \leftarrow \emptyset, \mathcal{Q} \leftarrow \emptyset$
if $P = \langle \emptyset, \emptyset, \emptyset \rangle$
  foreach $T_i \in \mathcal{TOP}$
    add $\langle [A_1, .., A_n], \langle \{T_i\}, \emptyset, \emptyset \rangle, T_i \rangle$ to $\mathcal{Q}$
else foreach $g_i \in$ FRINGE(P, f)
  add $\langle [A_1, .., A_n], P, g_i \rangle$ to $\mathcal{Q}$
until $\mathcal{Q} = \emptyset$
  remove $\langle [A'_1, .., A'_{n'}], P', act \rangle$ from $\mathcal{Q}$
  $P'' \leftarrow$ REPLACE(P', act, A'_1)
  if CONSISTENT?(P'')
    if $n' = 1$ add P'' to $\mathcal{EXPL}$
    else foreach $g_i \in$ FRINGE(P'', f)
      add $\langle [A'_2, .., A'_{n'}], P'', g_i \rangle$ to $\mathcal{Q}$
  if $act \notin \mathcal{PRIM}$
    foreach recipe $R_k \in \mathcal{R}$
      $P''' \leftarrow$ EXTEND(P', R_k, act)
      foreach s_j, where $act \rightarrow s_j \in P'''$
        add $\langle [A'_1, .., A'_{n'}], P''', s_j \rangle$ to $\mathcal{Q}$
return $\mathcal{EXPL}$



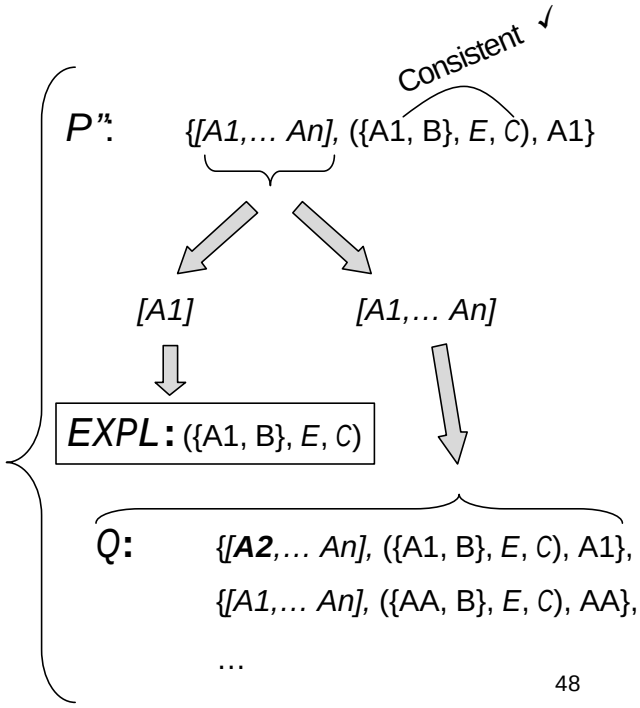
47

Items are taken off the queue, and the first item's plan is assigned to P'. We then replace the action in P' with the first action in the input action sequence.

Plan Recognition Algorithm

(a) Plan recognition.

RECOGNIZE($[A_1, .., A_n]$,P,f,$\mathcal{R}$) $\equiv$
$\mathcal{EXPL} \leftarrow \emptyset, \mathcal{Q} \leftarrow \emptyset$
if $P = \langle \emptyset, \emptyset, \emptyset \rangle$
  foreach $T_i \in \mathcal{TOP}$
    add $\langle [A_1, .., A_n], \langle \{T_i\}, \emptyset, \emptyset \rangle, T_i \rangle$ to $\mathcal{Q}$
else foreach $g_i \in$ FRINGE(P, f)
  add $\langle [A_1, .., A_n], P, g_i \rangle$ to $\mathcal{Q}$
until $\mathcal{Q} = \emptyset$
  remove $\langle [A'_1, .., A'_{n'}], P', act \rangle$ from $\mathcal{Q}$
  $P'' \leftarrow$ REPLACE(P', act, A'_1)
  if CONSISTENT?(P'')
    if $n' = 1$ add P'' to $\mathcal{EXPL}$
    else foreach $g_i \in$ FRINGE(P'', f)
      add $\langle [A'_2, .., A'_{n'}], P'', g_i \rangle$ to $\mathcal{Q}$
  if $act \notin \mathcal{PRIM}$
    foreach recipe $R_k \in \mathcal{R}$
      $P''' \leftarrow$ EXTEND(P', R_k, act)
      foreach s_j, where $act \rightarrow s_j \in P'''$
        add $\langle [A'_1, .., A'_{n'}], P''', s_j \rangle$ to $\mathcal{Q}$
return $\mathcal{EXPL}$



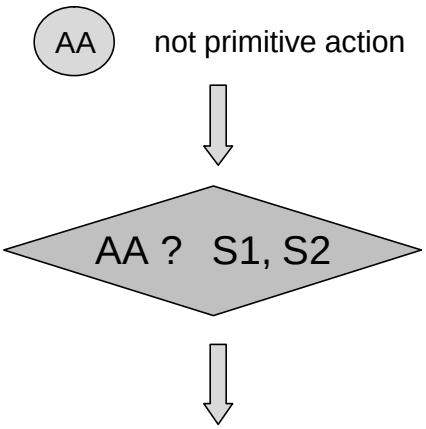
48

If the plan P" which contains A1 (from the input action sequence) is consistent, i.e. the actions follow the constraints, then we look at the length of the input action sequence. If there was only one action in the input sequence, then we know that it is consistent, so we can add it to the list of extended plans, or EXPL. If there are more actions in the input sequence, then we add to the queue with the first action removed from the list of input actions.

Plan Recognition Algorithm

(a) Plan recognition.

RECOGNIZE($[A_1, .., A_n]$,P,f,$\mathcal{R}$) $\equiv$
$\mathcal{EXPL} \leftarrow \emptyset, \mathcal{Q} \leftarrow \emptyset$
if $P = \langle \emptyset, \emptyset, \emptyset \rangle$
  foreach $T_i \in \mathcal{TOP}$
    add $\langle [A_1, .., A_n], \langle \{T_i\}, \emptyset, \emptyset \rangle, T_i \rangle$ to $\mathcal{Q}$
else foreach $g_i \in$ FRINGE(P, f)
  add $\langle [A_1, .., A_n], P, g_i \rangle$ to $\mathcal{Q}$
until $\mathcal{Q} = \emptyset$
  remove $\langle [A'_1, .., A'_{n'}], P', act \rangle$ from $\mathcal{Q}$
  $P'' \leftarrow$ REPLACE(P', act, A'_1)
  if CONSISTENT?(P'')
    if $n' = 1$ add P'' to $\mathcal{EXPL}$
    else foreach $g_i \in$ FRINGE(P'', f)
      add $\langle [A'_2, .., A'_{n'}], P'', g_i \rangle$ to $\mathcal{Q}$
  if $act \notin \mathcal{PRIM}$
    foreach recipe $R_k \in \mathcal{R}$
      $P''' \leftarrow$ EXTEND(P', R_k, act)
      foreach s_j, where $act \rightarrow s_j \in P'''$
        add $\langle [A'_1, .., A'_{n'}], P''', s_j \rangle$ to $\mathcal{Q}$
return $\mathcal{EXPL}$

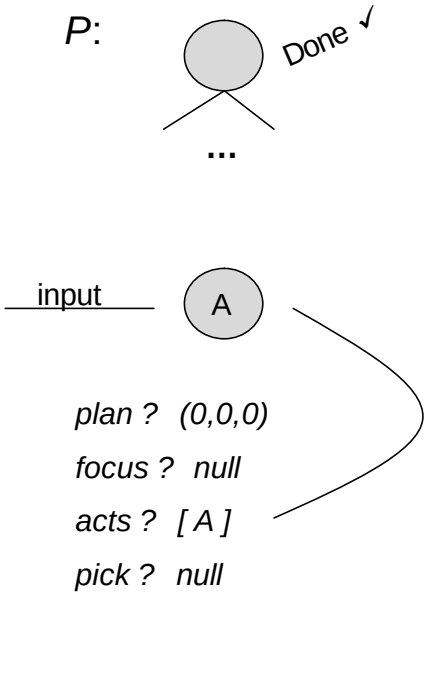


Q: {*[A1,… An],* ({**S1**, AA, B}, *E*, *C*), S1},
{*[A1,… An],* ({**S2**, AA, B}, *E*, *C*), S2},
…

49

If the current action of interest is not a primitive action, then it can be expanded using the recipe library. Here, we see that the recipe library says AA -> S1 and S2, so we add to the queue those extensions.

Focus and Clarification Algorithm



(b) Focus, ambiguity and clarification.

$plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$, $acts \leftarrow [\,]$
repeat
  wait for next input action A_i,
  if DONE?(root of $plan$)
    $plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$
  add A_i to $acts$
  pick $\leftarrow null$
  $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $\mathcal{EXPL} = \emptyset$
    set $focus$ to root of $plan$
    $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $|\mathcal{EXPL}| = 1$
    remove $pick$ from $\mathcal{EXPL}$
  else if $\mathcal{EXPL} = \emptyset$ or $|acts| > MaxWait$
    $pick \leftarrow$ CLARIFY$(\mathcal{EXPL})$
  if $pick \neq null$
    $plan \leftarrow pick$
    $focus \leftarrow$ UDATEFOCUS$(plan, A_i)$
    $acts \leftarrow [\,]$

50

For the focus, ambiguity and clarification algorithm, we use variables *plan* to represent the current plan, *focus* to represent the focus action which may be updated by a function called UpdateFocus, *acts* to represent the input actions, and *pick* to take on extended plans. Initially, we wait for the next input action and add it to *acts*. Everything else is set to null.

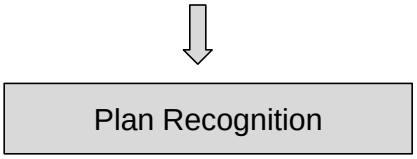
Focus and Clarification Algorithm

plan ? (0,0,0)

focus ? null

acts ? [A]

pick ? null



Plan Recognition

EXPL : ...

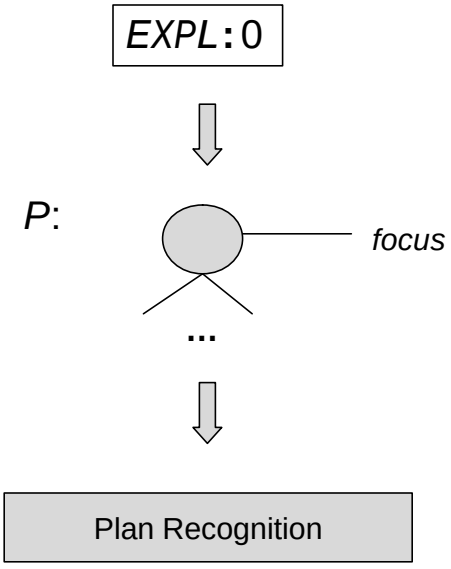
(b) Focus, ambiguity and clarification.

$plan \leftarrow \langle\emptyset,\emptyset,\emptyset\rangle$, $focus \leftarrow null$, $acts \leftarrow [\,]$
repeat
  wait for next input action A_i,
  if DONE?(root of $plan$)
    $plan \leftarrow \langle\emptyset,\emptyset,\emptyset\rangle$, $focus \leftarrow null$
  add A_i to $acts$
  pick $\leftarrow null$
  $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $\mathcal{EXPL} = \emptyset$
    set $focus$ to root of $plan$
    $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $|\mathcal{EXPL}| = 1$
    remove $pick$ from $\mathcal{EXPL}$
  else if $\mathcal{EXPL} = \emptyset$ or $|acts| > MaxWait$
    $pick \leftarrow$ CLARIFY$(\mathcal{EXPL})$
  if $pick \neq null$
    $plan \leftarrow pick$
    $focus \leftarrow$ UDATEFOCUS$(plan, A_i)$
    $acts \leftarrow [\,]$

51

We first perform plan recognition on this initial action and get some EXPL in return, which may or may not be empty.

Focus and Clarification Algorithm



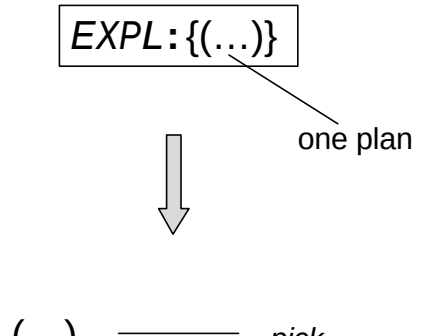
(b) Focus, ambiguity and clarification.

$plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$, $acts \leftarrow [\,]$
repeat
  wait for next input action A_i,
  if DONE?(root of $plan$)
    $plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$
  add A_i to $acts$
  pick $\leftarrow null$
  $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $\mathcal{EXPL} = \emptyset$
    set $focus$ to root of $plan$
    $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $|\mathcal{EXPL}| = 1$
    remove $pick$ from $\mathcal{EXPL}$
  else if $\mathcal{EXPL} = \emptyset$ or $|acts| > MaxWait$
    $pick \leftarrow$ CLARIFY$(\mathcal{EXPL})$
  if $pick \neq null$
    $plan \leftarrow pick$
    $focus \leftarrow$ UDATEFOCUS$(plan, A_i)$
    $acts \leftarrow [\,]$

52

If EXPL is empty, or in other words there was no extended plan returned, then we set the focus to the root of the plan and perform plan recognition again.

Focus and Clarification Algorithm



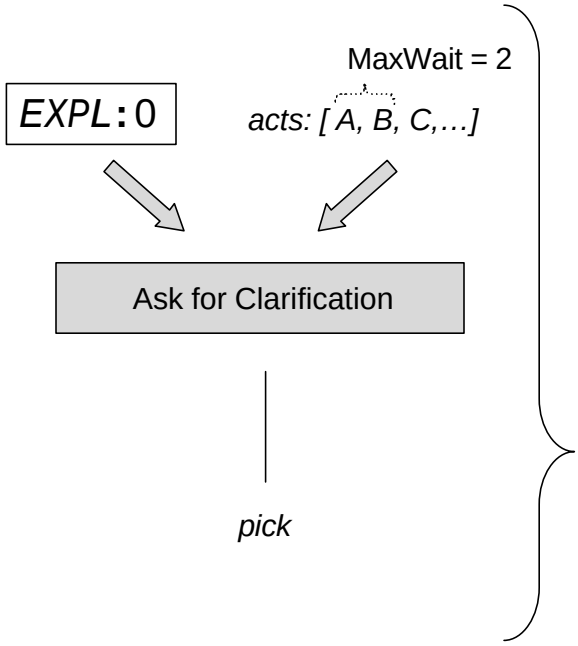
(b) Focus, ambiguity and clarification.

$plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$, $acts \leftarrow [\,]$
repeat
  wait for next input action A_i,
  if DONE?(root of $plan$)
    $plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$
  add A_i to $acts$
  pick $\leftarrow null$
  $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $\mathcal{EXPL} = \emptyset$
    set $focus$ to root of $plan$
    $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $|\mathcal{EXPL}| = 1$
    remove *pick* from $\mathcal{EXPL}$
  else if $\mathcal{EXPL} = \emptyset$ or $|acts| > MaxWait$
    $pick \leftarrow$ CLARIFY$(\mathcal{EXPL})$
  if $pick \neq null$
    $plan \leftarrow pick$
    $focus \leftarrow$ UDATEFOCUS$(plan, A_i)$
    $acts \leftarrow [\,]$

53

If the EXPL returned one plan, then set that plan to *pick*, removing it from EXPL.

Focus and Clarification Algorithm



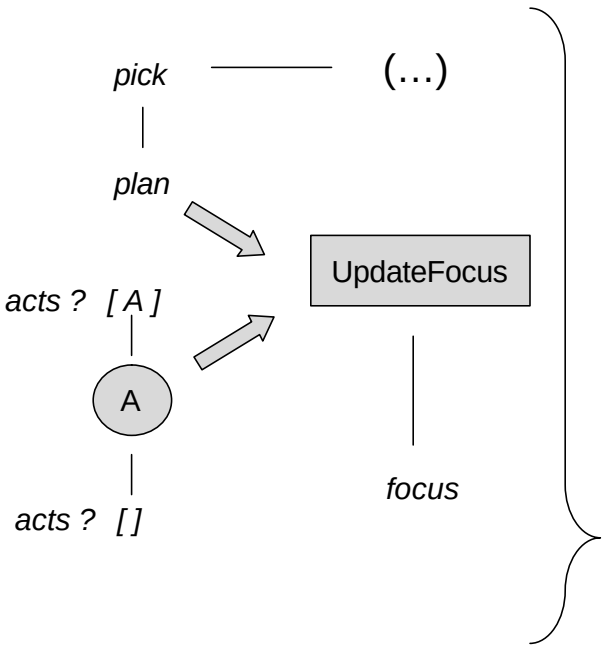
(b) Focus, ambiguity and clarification.

$plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$, $acts \leftarrow [\,]$
repeat
  wait for next input action A_i,
  if DONE?(root of $plan$)
    $plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$
  add A_i to $acts$
  pick $\leftarrow null$
  $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $\mathcal{EXPL} = \emptyset$
    set $focus$ to root of $plan$
    $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $|\mathcal{EXPL}| = 1$
    remove *pick* from $\mathcal{EXPL}$
  else if $\mathcal{EXPL} = \emptyset$ or $|acts| > MaxWait$
    $pick \leftarrow$ CLARIFY$(\mathcal{EXPL})$
  if $pick \neq null$
    $plan \leftarrow pick$
    $focus \leftarrow$ UDATEFOCUS$(plan, A_i)$
    $acts \leftarrow [\,]$

54

If EXPL returned nothing, or if the number of input actions exceeds the set value MaxWait (which is the maximum number of actions to observe before inquiring for clarification), then the algorithm calls a Clarify function, which will narrow down the possible plans to some single plan, which is set to *pick*.

Focus and Clarification Algorithm



(b) Focus, ambiguity and clarification.

$plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$, $acts \leftarrow [\,]$
repeat
  wait for next input action A_i,
  if DONE?(root of $plan$)
    $plan \leftarrow \langle \emptyset, \emptyset, \emptyset \rangle$, $focus \leftarrow null$
  add A_i to $acts$
  pick $\leftarrow null$
  $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $\mathcal{EXPL} = \emptyset$
    set $focus$ to root of $plan$
    $\mathcal{EXPL} \leftarrow$ RECOGNIZE$(acts, plan, focus, \mathcal{R})$
  if $|\mathcal{EXPL}| = 1$
    remove $pick$ from $\mathcal{EXPL}$
  else if $\mathcal{EXPL} = \emptyset$ or $|acts| > MaxWait$
    $pick \leftarrow$ CLARIFY$(\mathcal{EXPL})$
  if $pick \neq null$
    $plan \leftarrow pick$
    $focus \leftarrow$ UDATEFOCUS$(plan, A_i)$
    $acts \leftarrow [\,]$

55

If *pick* is actually set, then that becomes our current plan, which we can use to update the focus. The list of input actions is set back to empty and we are ready to repeat this whole process again… “wait for next input action”, etc.

Complexity

- Size of search space to explain one action: $F(R \times S)^L$
  - S: max number of steps in a recipe
  - R: max number of recipes applicable to an action
  - F: number of actions on the fringe of P
  - L: length of longest sequence of recipes the algorithm must consider
- Recursion for each action: worst-case complexity $O((F'(R \times S)^L)^N)$
  - N: number of input actions
- General case: $O((R^S)^d)$
  - d: depth of deepest possible plan

56

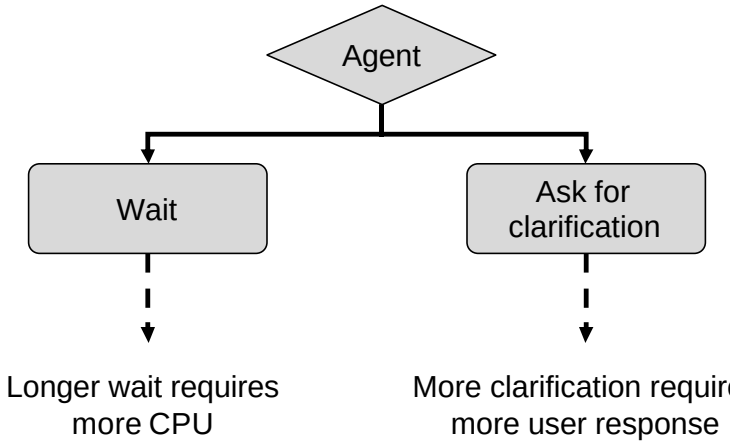
Complexity to explain one action is given by $F(R \times S)^L$. But since there are N input actions, the complexity needs to be raised to the power of N, namely $F'(R \times S)^L)^N$.

In the general case, the complexity is on the order of $(R^S)^d$ where d is the depth of the deepest possible plan.

If number of input actions is small (N is small), collaborative plan recognition problem is significantly more tractable than general plan recognition problem. Small number of input actions is guaranteed through clarification.

Focus, Ambiguity and Clarification

What to do when recognizer returns multiple explanations?



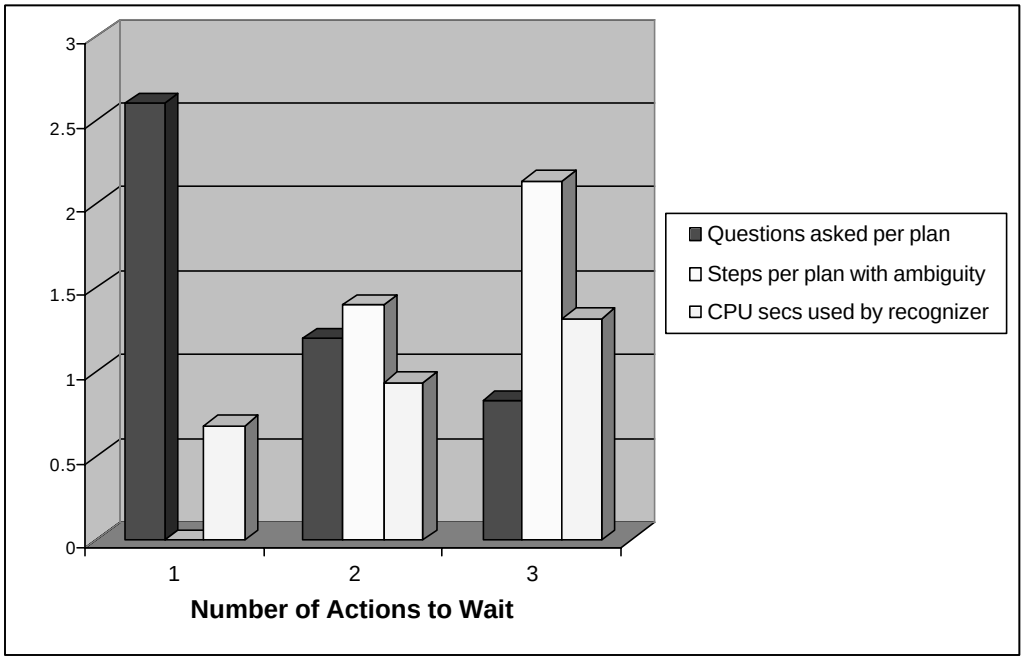
57

When plan recognition returns more than one explanation for an action, the agent can either wait for more observations on user actions, or it can ask for clarification right away.

When the agent waits for more observations, it saves the user the inconvenience of constantly communicating with the agent. However, the more actions that the agent waits, the larger the domain space, and as we saw in the previous slide, there is more complexity in solving the problem.

Asking for clarification keeps the complexity down, but it requires more user response. So the key is to keep a good balance between these two.

Average Results for 100 E-mail Scenarios

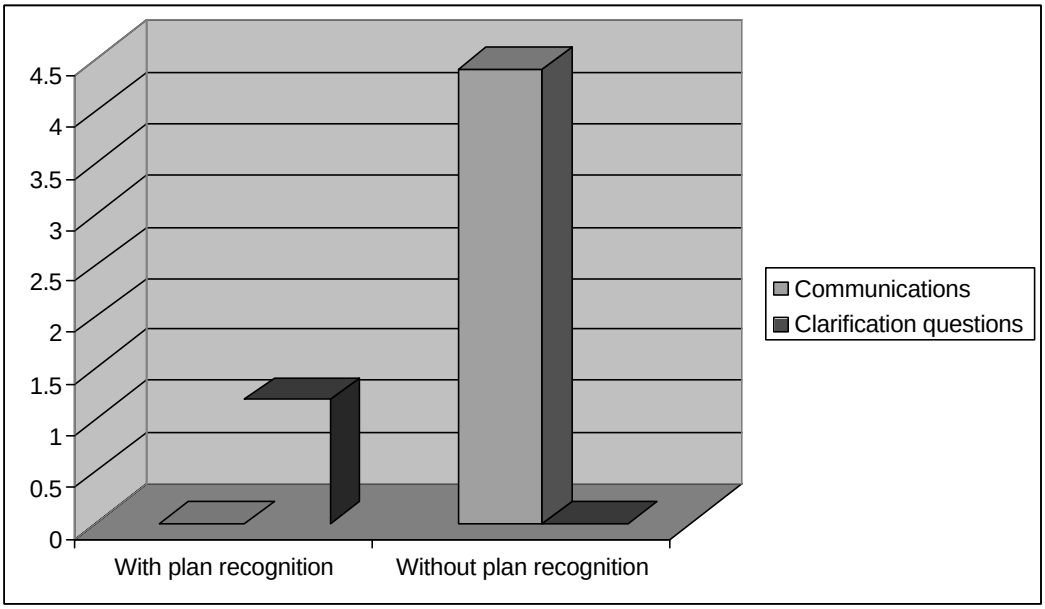


58

Here is shown some results from the e-mail problem. These are all average values obtained from averaging 100 cases.

On the x-axis is the number of actions that the agent waited to observe before asking for clarification. As we can see, the red bars indicating the number of questions asked per plan decreased as the agent waited for more observations. The number of steps per plan with ambiguity increased with more waiting, and the CPU time that was needed increased with more waiting.

Comparisons With and Without Plan Recognition



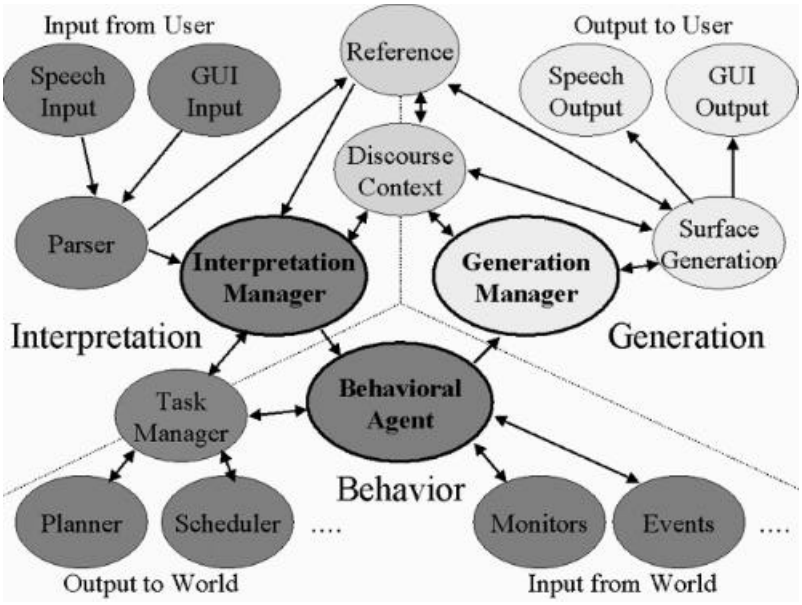
59

We can see in this graph that with plan recognition, the agent had to ask on average 1.2 clarification questions, but without plan recognition, the user had on average needed to make 4.4 communications to the agent. Both of these accomplished the same tasks, but it is obvious that plan recognition reduces the overall agent-user communications required to complete a task.

Outline

- Travel Scheduling Example
- Collaborative Discourse
- Plan Recognition
- **TRIPS**
- COLLAGEN

TRIPS Architecture



61

We showed a demo of the collaborative agent TRIPS in the beginning of this lecture. Here is a high-level architectural diagram of TRIPS. Notice the complexity of this agent: the technology behind speech input and output modules on the top to the actual planner and scheduler and event monitors on the bottom are not covered in this lecture.

The discourse context module contains the algorithm concerning the collaborative discourse theory, and the task manager has plan recognition abilities. Both of them work with the higher level module Interpretation Manager towards the understanding of the user inputs. Discourse context is also used to generate agent responses through the Generation Manager.

TRIPS Demo 2

This is another demo of the same TRIPS agent applied to another scenario concerning Pacifica refugee evacuation. Again, the goal of the exercise presented is self-explanatory. Below is a list of running commentaries concerning specific elements of the demo. Plan recognition and collaborative discourse are used throughout this exercise. We will only be pointing out some of the obvious occurrences. The number in the beginning of each bullet is the estimated time in the demo the commentary should take place followed by a brief description of what happened and how this relates to the theory we mentioned.

•1:12 – TRIPS displays a map of Pacifica at the user's request. This is the level tasks that regular systems without collaborative discourse and plan recognition should be able to perform. However, this action also serves the purpose of narrowing the recipe library to recipes related to Pacifica.

•1:29 – When the user asked for the location of the trucks, TRIPS gave a verbal response along with a visual indication of the truck's location on the map. This shows that our collaboration theories are not limited to verbal communication alone. This also demonstrates the usefulness of a setting where the user can not only hear but also observe actions of the agent and vise versa. A large amount of time can be saved because the agent and user does not need to fully explain all their actions.

•1:41 – The user asks the agent to use "the truck," the agent replies that it does not understand the request. (The user actually says "a truck" but "the truck" appears on the screen. This is either a mistake the user made in typing, or a mistake in the speech recognition system. In either case, the plan recognition part of the agent received "the truck" as an input) This is an example of clarification when the agent receives an ambiguous request. Since there are multiple trucks none of which officially labeled "the truck," the agent got confused. Instead of taking a guess which would have increased the ambiguity, the agent asked for a clarification to refine the search space.

•1:56 – Once both parties agreed to use a truck to get the people from Calypso to Delta, the agent highlighted that route on the map and displayed a schedule bar, this time without explicit request from the user. This further demonstrates the effectiveness of visual communication as well as the agent's ability to give its own input to this problem solving process, this time in a non verbal format.

•2:12 – This time the user asked a hypothetical questions "what if we went along the coast instead." In this case, the agent starts a separate sub-segment in parallel with the current path given and designates this as the current active segment. (Notice that a new route appears on the map, and a new schedule window also opens on top of the old one.) When the user says "forget it," the agent have to close this segment and return to the last active segment. (The new route is removed and the new schedule window closes, thus placing the former schedule window back on the screen.) This demonstrate the agent's ability to track multiple scenarios at once.

•2:57 – In this case the user first asked to use another truck to get people from Exodus to Delta, and then he changed his mind and asked the user to use "a helicopter instead." There are 2 significant points associated with this action. First, it demonstrates the agent's ability to retrace steps back to an earlier point in the planning process. Also, when the user used the word "instead," the agent must realize that the user is referring to the same route and only changing the means of transportation. This is easily accomplished with the segmentation layers of collaborative discourse.

•3:25 – Again when the user says "go on to Abyss," he did not have to specify truck one to Abyss. The agent realizes this because that is the segment currently in focus.

•3:47 –The rest of the demo works under the condition that the Delta bridge is out. The agent identifies the parts of the plan that are no longer executable, and a re-planning process begins. Since the rest of the demo covers much of the same concepts already discussed, we will stop the demo here in the interest of time.

One last thing to note about this demo is that the user never stated the top level goal of refugee evacuation. However, the agent can still make inferences and perform tasks based on the sub-goals such as moving people from one location to another.

This demo can be found at the following url:
http://www.cs.rochester.edu/research/cisd/projects/trips/movies/TRIPS-98_v4.0/.

Outline

- Travel Scheduling Example
- Collaborative Discourse
- Plan Recognition
- TRIPS
- **COLLAGEN**

63

COLLAGEN is another collaborative agent currently under development. This is the same agent used to evaluated the email scenario covered in the plan recognition section above. However, below is the same COLLAGEN agent applied to a completely different platform

Mel the Robotic Penguin

64

This is Mel, the robotic penguin. For most of this lecture, we focused on examples where the collaborative agent is linked to a static computer and communication only takes the form of speech or on-screen displays. However, we mentioned in the beginning of this lecture, that collaborative agents are modular and application independent. And we believe that Mel is a good demonstration of this fact. We want to emphasize that the COLLAGEN system used in Mel is the same agent used in the email application we discussed earlier. From email client to robotic penguin, one can see truly how mobile a collaborative agent can be.

Communicating with Mel

- Mel can seek out a person to interact with for the demonstration.
- Mel also recognizes and responds to head gestures (such as nods) to indicate interest in Mel and environment.
- People find the Mel's gestures more natural than an unmoving conversational partner, and that they direct their attention more to Mel than the unmoving conversational partner.

65

Here are some facts about Mel that distinguishes him from the other platforms used.

Up to this point, although we've seen collaborative agents work under various scenarios, all of them can be categorized as some from of planning and scheduling. However, this demo shows Mel performing an educational tutorial, completely unrelated to planning. This further demonstrates the versatility and modularity of collaborative agents.

Again, below is a list of running commentaries concerning specific elements of this demo. The number in the beginning of each bullet is the estimated time in the demo the commentary should take place followed by a brief description of what happened and how this relates to the theory we mentioned.

•0:48 – Mel uses gestures instead of words to indicated the location of the iGlassware system.

•1:16 – The user poured water into the cup and Mel immediately noted that the task has been accomplished without any verbal confirmation from the user. This is a demonstration of plan recognition similar to the email example given above. A large amount of time can be saved when the agent recognizes actions through observation. In this case, Mel observed that the glass is full and thus inferred that the user is ready to move on.

•1:42 – Again, through observation, Mel noticed that the glass is now empty.

•1:51 – Mel asks whether the user would like to continue, the user replays "OK," Mel then uses a nod instead of a verbal response to indicate that he understood the user and will continue.

• 2:12 – The user says "OK," but Mel could not make out the words. So he asks for clarification to avoid confusion.

•2:44 – Finally, Mel uses both words and gestures to indicate the end of the conversation.

The important thing to note is that Mel is the one taking the lead in this demo, and he requires very little input from the user. However, he is constantly using words, gestures, and questions to keep the user interested. He also re-confirms the users interest by directly asking.

A CD containing this demo will be provided.

Summary

- Resourceful, interactive collaboration agents can be tremendously useful
- Many techniques are required to construct these agents including collaborative discourse and plan recognition
- TRIPS and COLLAGEN are two such agents that uses these techniques
- These agents are largely platform independent and can be applied to various different applications.

67

Further Work

- Improve the flexibility and robustness of the algorithms to deal with incompleteness of the agent's recipe library and handle parallel interleaved goals
- Support negotiation between the user and agent
- Build agents that operate remotely (e.g., on the Internet), which will require more discussion between the agent and user about past and future actions.

Acknowledgements

- The video demo of COLLAGEN was provided by Dr. Charles Rich at Mitsubishi Electric Research Laboratories (MERL).

- The TRIPS demo can be found on the TRIPS website, url: http://www.cs.rochester.edu/research/cisd/projects/trips/movies/TRIPS_CPoF/.

- *Star Trek IV: The Voyage Home* is copyright of Paramount Pictures.

69

References

- Rich, C.; Sidner, C.L., "COLLAGEN: A Collaboration Manager for Software Interface Agents", *An International Journal: User Modeling and User-Adapted Interaction*, Vol. 8, Issue 3/4, pps 315-350, 1998

- N. Lesh, C. Rich, Charles and C. Sidner. "Using Plan Recognition in Human-Computer Collaboration." in Proceedings of the Seventh Int. Conf. on User Modelling, Banff, Canada, July 1999.

70

Questions?