Learning by Asking Questions and Learning by Aligning Stories: How a Story-Grounded Problem Solver can Acquire Knowledge

Zhutian Yang\textsuperscript{1} and Patrick Henry Winston\textsuperscript{2}

\textsuperscript{1} Adaptive Computing Laboratory, National University of Singapore
\textsuperscript{2} Computer Science and Artificial Intelligence Laboratory, MIT
and Center for Brains, Minds, and Machines, MIT

We describe how a problem solver, grounded in the Genesis Story Understanding System, acquires knowledge by asking questions and by aligning successful problem-solving stories.

To illustrate learning by asking questions, we demonstrate how the Genesis problem solver learns the steps to mix a blocks-world martini from another problem solver and how it learns to make a real-world fruit salad from a human.

To illustrate learning by aligning stories, we demonstrate how our Genesis problem solver learns to replace a phone battery from two 80-word stories that have much irrelevant detail and nothing expressed in exactly the same way.

We conclude that the Genesis problem solver learns much like humans do. It asks questions and exploits precedents. It learns something specific from each experience. It tells itself its own story as it solves problems, exhibiting a kind of self-aware behavior.

Keywords: computational models of human intelligence; story understanding; self-aware problem solving; learning by asking questions; learning by aligning stories.
Learning by Asking Questions and Learning by Aligning Stories: How a Story-Grounded Problem Solver can Acquire Knowledge

Zhutian Yang⁵ and Patrick Henry Winston²

¹Adaptive Computing Laboratory, National University of Singapore
²Computer Science and Artificial Intelligence Laboratory, MIT
and Center for Brains, Minds, and Machines, MIT

December 17, 2018

Abstract
We describe how a problem solver, grounded in the Genesis Story Understanding System, acquires knowledge by asking questions and by aligning successful problem-solving stories.

To illustrate learning by asking questions, we demonstrate how the Genesis problem solver learns the steps to mix a blocks-world martini from another problem solver and how it learns to make a real-world fruit salad from a human.

To illustrate learning by aligning stories, we demonstrate how our Genesis problem solver learns to replace a phone battery from two 80-word stories that have much irrelevant detail and nothing expressed in exactly the same way.

We conclude that the Genesis problem solver learns much like humans do. It asks questions and exploits precedents. It learns something specific from each experience. It tells itself its own story as it solves problems, exhibiting a kind of self-aware behavior.

Keywords: computational models of human intelligence; story understanding; self-aware problem solving; learning by asking questions; learning by aligning stories

Vision

How can a problem solver ask questions when it needs help?
How can a problem solver extract actionable knowledge from human stories?

We argue that the right answers to these questions can be found in treating problem solving as a special case of recipe following and in treating recipe following as a special case of story understanding (Winston and Holmes, 2018). There are two key reasons.

First, in our Genesis story understanding system, all knowledge is expressed in English, so all problem-solving knowledge is likewise expressed in English. This commitment to English makes the entire system highly transparent and enables easy instruction without programming.
Second, because problem solving amounts to developing an internal story, Genesis can bring all of its story understanding capabilities to bear on that internal story, including, for example, common sense reasoning, concept discovery, and concept-driven summary.

Accordingly, our approach to asking and discovery extends the Genesis story-grounded problem solver developed by Winston (2018). The Genesis problem solver, much influenced by the work of Pat Langley and his students (Langley et al., 2014; Bai et al., 2015; Pearce et al., 2015; Langley et al., 2016), answers complex questions about stories, such as \textit{Did Lu murder Shan because America glorifies violence}, thereby modeling culturally specific norms uncovered by social scientists (Morris and Peng, 1994).

In our robot-oriented work reported here, we extended the Genesis problem solver in three major steps:

- First, we added the knowledge required to solve robot problems.

We provided the Genesis problem solver with basic knowledge about how to move over, place at, and rotate. Using that basic knowledge, our Genesis problem solver knows how to mix a martini, driving either a blocks-world simulator or a real robot as shown in figures 1 and 2:

![Figure 1: A blocks-world simulator mixing a martini. Each block is identified with an ingredient.](image1)

![Figure 2: A real robot mixing a martini. Courtesy of Adam Kraft.](image2)

The blocks-world simulator was, of course, inspired by Winograd’s SHRDLU program (1970). We use the blocks-world simulator during test and development, noting that the commands issued to it are the same as those that we have our Genesis problem solver issue to a real robot.

- Second, we added a mechanism enabling our Genesis problem solver to ask questions when stuck and to provide suggestions when queried.

Knowledge can flow in both directions, between two problem solvers or between a problem solver and a human, as shown in figure 3. For example, if a problem solver does not know how to pour from one container into another, it asks another problem solver to suggest the missing knowledge. If a robot’s vision system does not see the object to be picked up, it asks a human where the object is stored.

Knowledge flowing from expert humans to our problem solver, in English, facilitates rapid knowledge acquisition. Knowledge flowing from a problem solver to a novice is important in situations where sophisticated perception and action is needed but there is no robot available with human-level perceptual or motor abilities.

- Third, we added a mechanism that examines story ensembles in which each story describes a successful how-it-was-done scenario. That mechanism extracts essential steps, leaving behind the incidentals.
The key idea is to align different-looking steps in two stories by noting the goals they achieve. For example, when reading stories about replacing a phone battery, Genesis aligns *Alice removed the cover of the phone* with *Bob took the cover off the phone* because both steps achieve the goal of exposing the phone’s parts according to common sense abduction rules. Given these two steps align, we can assume the goal they achieve to be important. In contrast, *Bob poured himself a cap of coffee* is considered incidental if the goal it achieves appears only in Bob’s story. This idea resonates with the near-miss group idea (Winston and Rao, 1990) as a technique for acquiring knowledge.

**Genesis solves problems**

In this section, we explain how the Genesis Story Understanding System solves problems by using recipe-like descriptions of what to do. We also demonstrate how we have adapted the version previously reported (Winston, 2018) to handle robot problem solving.

In other problem solvers, the emphasis is on problem spaces and search. In Genesis, the emphasis is on how to react to questions and problems with an assortment of short, recipe-like microstories that connect questions and problems to a small set of simple, just-do-it programs.

The short, recipe-like microstories fall naturally into several categories. Figure 4 shows how categories fit together. Figure 5 explains the categories.

**The Genesis problem solver bottoms out in just-do-it programs**

Note that our problem solver bottoms out in what we call just-do-it programs that do computations without reasoning. Consider, for example, what happens when someone asks how you drink from a glass. You answer: I reach out, surround the glass with my fingers, close my fingers, lift, move the glass to my mouth, tilt.... Then, if asked, how do you close your fingers, all you can answer is: I just do it. You can perform experiments that characterize the speed and force of your grasping, but you cannot explain it.

In our prior work on story-oriented problem solving, just-do-it programs determine if, for example, a particular element is in a story and whether two elements in a story are connected by causal relations. In our robot-oriented work, about to be described in detail, some perceptual just-do-it programs determine if, for example, one object supports another, while some motor just-do-it
The Genesis problem solver explains what it is doing

Each problem, approach, checker, and method reports on what is happening, so you can always see exactly what the problem solver is doing. Here, for example, is a running account of what the problem solver does when asked to put a gin container above a glass.

0 Problem: Put gin above glass.
   1 Intention: Put gin above glass.
   2 Condition: Verify that nothing is on glass.
   3 Method: Verify that nothing is on glass.
   3 Just did: Verify that nothing is on glass.
   2 Condition satisfied: Verify that nothing is on glass.
   2 Condition: Verify that nothing is on gin.
   3 Method: Verify that nothing is on gin.
   3 Just did: Verify that nothing is on gin.
   2 Condition satisfied: Verify that nothing is on gin.
   2 Method: Move gin above glass.
   2 Just did: Move gin above glass.
   1 Intention succeeded: Put gin above glass.
0 Problem solved: Put gin above glass.

The Genesis Problem Solver Learns by Asking Questions

In this section, we describe the mechanisms that enable the Genesis problem solver to ask questions when it lacks needed knowledge. First, we show how a novice version of the Genesis problem solver asks a how-do-I question, inviting an expert version of the Genesis problem solver to suggest this missing knowledge. Then, we describe how the Genesis problem solver asks a human questions as it learns about and executes recipes for cooking.
### Category | Definition | Example microstories
---|---|---
**Insight** | *Insights* connect problems to intentions and optional results. Intentions indicate what is to be done without specifying how. | If the problem is "Put xx on yy". Intention: Put xx on yy.

**Approach** | An *approach* specifies how an intention may be addressed. An intention may consist of one or more conditions and a method that is to be performed if all the conditions are satisfied. | If the intention is "Put xx above yy". Condition: Verify that nothing is on yy. Condition: Verify that nothing is on xx. Method: Move xx above yy.

| | An *Approach* may also consist of *steps*, each of which specifies a problem to be solved. A series of steps enables high-level task planning for robot manipulation problems. | If the intention is "Mix martini". Step: Identify elements. Step: Pour gin into glass. Step: Pour vermouth into glass. Step: Place lemon in glass.

**Checker** | *Checkers* determine if conditions are satisfied, and if not, create subproblems aimed at satisfying the unsatisfied conditions. Checkers enable the Genesis problem solver to recurse. | If the condition is "Verify that nothing is on xx". Check: Verify that nothing is on xx. Instantiate: yy is supported by xx. Solve: Get rid of yy.

**Executor** | *Executors* connect methods to just-do-it programs for perception and action. | If the method is "Move xx to yy". Execute: Call "jdPutOn" with xx with yy.

---

**Genesis asks how to mix a blocks-world martini**

Suppose a novice version of the Genesis problem solver knows the basics of blocks manipulation but does not know how to mix a martini. It has a default insight:

If the problem is "Mix martini".

Intention: Mix martini.

The end.

But alas, the novice has no known approach with **Mix Martini** as the intention. Accordingly, it broadcasts a request, in English, for approaches with the **Mix Martini** insight.

0 Problem: Mix martini.

1 How do I mix martini?

An expert version of the Genesis problem solver, one that does know how to mix a martini then suggests, in English:
1 Expert suggests steps:

1 Step: Identify elements.
1 Step: Pour gin into glass.
1 Step: Pour vermouth into glass.
1 Step: Place lemon in glass.

The Identify-elements step merely associates gin, vermouth, lemon, and glass with various objects that the simulator knows about, bringing the novice to the next blocker: it does not know how to pour, so it asks again, and again gets help from an expert, with the request and the suggestion, again, in English:

2 Problem: Pour gin into glass.

3 How do I pour gin into glass?

3 Expert suggests steps:
3 Step: Put gin above glass.
3 Step: Rotate gin.
3 Step: Move gin to the table.

Note that knowledge about mixing a martini consists of an approach containing steps, two of which involve an approach that is also expressed as a series of steps. See Appendix A for a video demonstration. See Appendix B for the complete story told by the novice problem solver. See Appendix C for the knowledge used in mixing a martini.

Genesis asks how to prepare a fruit salad

Here, in a more complex example, the Genesis problem solver learns to make a fruit salad by interviewing a human expert; then, it instructs a real robot to make one.

Learning starts when a human prompts Genesis via a question, request, or proposal, such as the following:

(Question) How do you make a salad? How can I make a salad? Do you know how to make a salad?
(Request) Could you please make a salad? Please make a salad. Make a salad.
(Proposal) Let’s make a salad!

All three types of prompts, when translated into Genesis’s inner language of role frames (Winston and Holmes, 2018), contain the same embedded *make* expression, as shown in figure 6.

When stuck, Genesis asks the human expert to teach it the steps. It generates the request by constructing an inner-language expression that it then translates into an English sentence: *Can you make a fruit salad.* Yang, the human, responds with a two-step approach:

Yang: Let’s make a fruit salad!
Genesis: Can you teach me how to make a fruit salad?
Yang: You toss the fruits into the bowl. Then, you load the bowl with nuts.

From the response, Genesis generates an insight, a high-level approach, and insight/approach pairs corresponding to each of the lower-level steps in the higher-level approach:

If the problem is "Make a fruit salad".
   Intention: Make a fruit salad.
   The end.
If the intention is "Make a fruit salad".
Step: Toss the fruits into the bowl.
Step: Load the bowl with nuts.
The end.

// ---------- Step 1
If the problem is "Toss the fruits into the bowl".
Intention: Toss the fruits into the bowl.
The end.

If the intention is "Toss the fruits into the bowl".
Step: The fruits are in the bowl.
The end.

// ---------- Step 2
If the problem is "Load the bowl with nuts".
Intention: Load the bowl with nuts.
The end.

If the intention is "Load the bowl with nuts".
Method: Assume success.
The end.

Note that the intention Toss the fruits into the bowl implies the problem The fruits are in the bowl, which Genesis already knows how to solve. Genesis generates a new approach that ties together the intention and the lower-level step.
Because Genesis does not yet know how to deal with *Load the bowl with nuts*, Genesis connects the approach to the temporary Step *Assume success*. Later, it asks for an explanation as in the following:

Genesis: How do I load the bowl with nuts?
Yang: You pour the nuts from the black and white bottle into the bowl.

Then, Genesis updates the approach for Step 2:

- If the intention is "Load the bowl with nuts".
  - Step: Move nuts from the black and white bottle to the bowl.

Finally, to learn as much as it can, Genesis asks if there are any variations of the method:

Genesis: Do I always make a fruit salad in this way?
Yang: If I am not on a diet, you can also add salad dressing to the bowl.

Given the diet answer, Genesis creates a new approach with a condition and updates the existing approach with an opposing condition:

// Existing approach
If the intention is "Make a fruit salad".
Verify: I am on a diet.
Step: Toss the fruits into the bowl.
Step: Load the bowl with nuts.
The end.

// New approach
If the intention is "Make a fruit salad".
Verify: I am not on a diet.
Step: Toss the fruits into the bowl.
Step: Load the bowl with nuts.
Step: Add salad dressing to the bowl.
The end.

Genesis continues updating the generated micro-stories and asking *Is that all* until the expert has no more alternative methods to teach and answers: Yes.

Genesis: Is that all?
Yang: Yes.
Genesis: I learned how to make a fruit salad.

Finally, the human expert can request a real-time robot demonstration:

Yang: Show me.
Genesis: Let me show you how I make a fruit salad.

To execute the actions, we used a Kinova MICO arm. To execute perceptual checking, we used a vision system developed by Shridhar Mohit and David Hsu (2018), which locates objects based on their intrinsic properties, such as color or spatial relationships with other objects, as in the bottle on the left.

The Genesis problem solver first determines which approach to apply by asking a question based on a condition previously learned.

Genesis: Are you on a diet?
Yang: No.

Then, the Genesis problem solver gets stuck because it sees cups and a bowl on the table but does not see fruits and dressings. The conditions for direct manipulation are unsatisfied in the following approaches for the intention *Put xx in yy*. 
If the intention is "Put xx in yy".
Verify: xx is not an instance of seed.
Verify: xx is not an instance of flavorer.
Verify: xx can be observed.
Method: Move xx to yy. The end.
The end.

If the intention is "Put xx in yy".
Verify: xx cannot be observed on the table.
Instantiate: xx are distributed in zz in kk in qq.
Solve: xx moves from zz to yy.
Solve: xx moves from kk to yy.
Solve: xx moves from qq to yy.
The end.

If the intention is "Move xx from yy to zz".
Step: yy is above zz.
Step: yy is tilted.
Step: yy is on the table.
The end.

So the Genesis problem solver asks where the objects are stored:

Genesis: Where are the fruits?
Yang: The apples are in the green cup,
the watermelons are in the red cup,
and the blueberries are in the blue cup.

When the human answers with three containers, Genesis assigns the three containers to the three variables zz, kk, and qq. Then, it solves each of the new problems by pouring the fruits from the cups into the bowl in the suggested order.

Then, Genesis also asks where the salad dressing is stored because it is a kind of flavorer and according to the definition found in WordNet (Fellbaum, 1998), as shown in figure 8. We have predefined the common sense that flavorers cannot be manipulated directly.
Figure 8: Our Genesis problem solver checks WordNet to determine if the target object must be stored in a container. The left panel shows the WordNet threads for salad dressing, which is a kind of flavorer, so the novice will ask the human expert where the salad dressing is. The right panel shows the WordNet threads for fruits, which are not any kind of flavorer, seed, or other class that must be stored in containers.

Genesis: Where is the salad dressing?
Yang: It is in the bottle on the left.

Instantiating the unknown container with the bottle on the left, Genesis can then solve the final step for making a fruit salad with the following approach:

If the intention is "Put xx in yy".
Verify: xx is an instance of seed.
Instantiate: xx is stored in zz.
Solve: Move xx from zz to yy.
The end.

This simple example demonstrates how the Genesis problem solver asks a human questions in two scenarios. First, it asks the human to teach it recipes and their variations when it does not know how to solve a problem. It uses a human’s answers to enrich its problem-solving knowledge. Second, the Genesis problem solver asks where an object is stored when it cannot be observed or if it belongs to a category that cannot be directly manipulated. It extracts the problem-solving information from a human’s answers and executes recipes with initially incomplete information.

Summary

The Genesis problem solver, equipped with knowledge about robot manipulation, knows how to, or learns how to, prepare drinks and salads. The fruit-salad example is representative of several recipes that our real robot has learned and executed. Others include preparing breakfast cereals and seasoning pasta. The problem solver can instruct a blocks-world simulator, a real robot simulator, or an actual robot. When stuck, the Genesis problem solver asks a more knowledgeable program or a human for help. Because all problem-solving knowledge is expressed in an inner language, readily
translated to and from English, the Genesis problem solver can learn by asking questions and tell its story in English. Sample videos, in Appendix A, illustrate how the Genesis problem solver uses a blocks-world simulator to mix a martini with help from a smarter version of itself, and how the Genesis problem solver uses a real robot to prepare a fruit salad and other dishes.

**Genesis learns from story ensembles**

In this section, we explain how we have gone beyond learning by asking to learning by aligning pairs of stories.

Consider the following story about replacing a phone battery:

**Start story titled “Alice’s story.”** Alice put her pants in the washing machine. Alice forgot to take her phone from her pants. The phone became wet. It does not work. Alice quickly removed the cover from the phone and she collected the old battery from the phone. Alice purchased the replacement battery online. The replacement battery arrived and she inserted it into the phone. She put the cover on the phone. She recharged her phone for four hours. Bravo! The phone worked again. The end.

What can be learned from the story about replacing a phone battery? Not that you have to put your pants in a washing machine. Not even that the phone became wet. Irrelevant details swamp the key elements of phone battery replacement. How can a program decide which details to include in a phone battery replacement recipe?

The problem should be easier if there is a second story, such as this one:

**Start story titled “Bob’s story.”** A client’s phone did not work. The client asked Bob to change the battery of the client’s phone. Bob made coffee and drank it. Bob got a replacement battery from the stockroom. Bob slid down the cover from the client’s phone. Bob takes out the old battery from the client’s phone and put the replacement battery in the phone. Lastly, he placed the cover on the client’s phone. The client’s phone started working again. The client’s problem is solved. The client was happy. The end.

One idea would be to keep only those parts that correspond, but alas, no parts correspond exactly. You could argue that there are paraphrases and paraphrases indicate correspondence, but how can you determine which elements are paraphrases?

Our approach deploys the full power of our Genesis story understanding system.

- Genesis analyzes the stories producing elaboration graphs using rules and concept patterns.
- Genesis notes where various actions imply common sub-goals by abduction.
- Genesis forms a general recipe from the common sub-goals in the two stories.
- Genesis forms two phone-specific recipes by listing the actions that produced the common sub-goals.

**Genesis uses abduction rules to identify common sub-goals**

Here are the abduction rules to determine the sub-goals indicated by various actions:

- xx and yy are persons.
- If xx drinks coffee, then xx must want to become alert.
We note that many of the words used in the wanting specifications evoke the container metaphor of George Lakoff and Mark Johnson (1980). This is perhaps not surprising in part-replacement scenarios.

In its analysis, Genesis also uses various common sense deduction and explanation rules (marked with *may*) that enrich understanding and generate elements that somehow must be ignored:

- If xx forgot to take yy from zz, then zz contains yy.
- If a phone becomes wet, then the phone may not work.
- If xx’s problem evidently is solved, then xx may be happy.

Now, when Genesis reads Alice’s story, Genesis produces the elaboration graph shown in figure 9.

![Alice’s story](image_url)

Figure 9: Alice’s story. Abducted sub-goals appear in purple. A deduction appears in yellow. An explanation rule, signaled by an orange line, connects getting wet with not working.

When Genesis reads Bob’s story, Genesis produces the elaboration graph shown in figure 10. The abductions enable Genesis to discover instances of the *Action-evoking goal* concept pattern:

Start description of "Action-evoking goal".

- xx’s wanting vv leads to aa.

The end.

At this point, the abducted sub-goals in two *Action-evoking goal* patterns do not match:
Bob’s story

Alice wants to clean her pants.
Bob wants to become alert.

Four do match:

Alice wants to expose her phone’s parts.
Bob wants to expose the client’s phone’s parts.

Alice wants to separate the old battery from her phone.
Bob wants to separate the client’s phone’s battery from the client’s phone.

Alice wants to incorporate the replacement battery into her phone.
Bob wants to incorporate the replacement battery into the client’s phone.

Alice wants to attach the cover to her phone.
Bob wants to attach the cover to the client’s phone.

Genesis uses common sub-goals to create recipes

From the matches, Genesis creates a recipe consisting of four goals from the objects of the wanting expressions.
Abstract battery replacement recipe:
Step: Expose the phone's parts.
Step: Separate the old battery from the phone.
Step: Incorporate the replacement battery into the phone.
Step: Attach the cover to the phone.
The end.

Looking at the other end of the matching abductions, the consequences, Genesis also creates two recipes, each of which works for a particular phone. Note that Genesis removes entity owners from the inner-language expressions before translating them to English:

Specific battery replacement recipe learned from Alice's story:
Step: Remove the cover from the phone.
Step: Collect the old battery from the phone.
Step: Insert the replacement battery into the phone.
Step: Put the cover on the phone.
The end.

Specific battery replacement recipe learned from Bob's story:
Step: Slide the cover from the phone.
Step: Take out the old battery from the phone.
Step: Put the replacement battery in the phone.
Step: Place the cover on the phone.
The end.

Summary

Two stories told about repair each may involve a great deal of irrelevant detritus. To extract a common repair recipe, Genesis must determine which story elements are fundamental and which are incidental. Genesis's overall strategy is to look for common elements in two stories. First, Genesis identifies sub-goals in each story by abduction from specified actions. Then, Genesis aligns the sub-goals to construct recipes based on those sub-goals. One is an abstract recipe listing the sub-goals; two others are specific recipes, based on using common sub-goals to identify the relevant parts of the two contributing stories.

Contributions

- We argued that problem solving is a special case of recipe following and that recipe following is a special case of story understanding.
- We argued that grounding a problem solver in the Genesis story understanding system has major benefits: First, all knowledge is expressed in English, facilitating understanding, explanation, and learning. Second, the problem-solving activity tells its own story, enabling self reflection and unleashing all of Genesis's story processing capability on that story.
- We explained how the Genesis problem solver can acquire problem-solving knowledge by asking a more knowledgeable version of itself or a human expert for help. We demonstrated learning by asking questions by having the Genesis problem solver to mix a blocks-world martini and by having the Genesis problem solver to make a fruit salad.
• We explained how the Genesis problem solver can discover problem-solving recipes by aligning problem-solving stories, finding implied sub-goals, and then constructing steps from common sub-goals. We demonstrated learning by aligning stories by having the Genesis problem solver create abstract and specific recipes for replacing a phone battery from two different-looking stories told by humans.

Acknowledgements

The work described in the paper was supported, in part, by the Air Force Office of Scientific Research, contract number FA9550-17-1-0081.

Yang’s robotic demonstrations were done at and in collaboration with the Adaptive Computing Laboratory at the National University of Singapore. The battery replacement task was suggested by Dynamic Object Language Labs, Inc.

References


Appendix A: Video examples

At time of DSpace deposit, video available via links attached to pictures below:

Figure 11: Mixing a martini with a blocks-world robot.

Figure 12: Mixing a martini with a real robot. Courtesy of Adam Kraft.

Figure 13: Preparing a fruit salad. The robot instructed by Genesis is a Kinova Micro Robotic Arm. The voice recognition and synthesis systems are by Amazon Web Services.

Figure 14: Additional examples of food preparation.

Figure 15: Simulated robot changing phone battery. Courtesy of Cagri Zaman.
Appendix B: Martini problem solving report

When asked to mix a martini, the novice program succeeds with help from an expert.

0 Problem: Mix martini.

1 How do I mix martini?

1 Expert suggests steps:
1 Step: Identify elements.
1 Step: Pour gin into glass.
1 Step: Pour vermouth into glass.
1 Step: Place lemon in glass.

1 Intention: Mix martini.
2 Problem: Identify elements.
3 Intention: Identify elements.
4 Problem: B5 is known as the glass.
5 Intention: B5 is known as the glass.
6 Method: B5 is known as the glass.
6 Just did: B5 is known as the glass.
5 Approach succeeded: B5 is known as the glass.
4 Problem solved: B5 is known as the glass.
4 Problem: Pour gin into glass.
4 Intention: Pour gin into glass.
5 Intention: B1 is known as the vermouth.
6 Method: B1 is known as the vermouth.
6 Just did: B1 is known as the vermouth.
5 Approach succeeded: B1 is known as the vermouth.
4 Problem solved: B1 is known as the vermouth.
4 Problem: B7 is known as the lemon.
5 Intention: B7 is known as the lemon.
6 Method: B7 is known as the lemon.
6 Just did: B7 is known as the lemon.
5 Approach succeeded: B7 is known as the lemon.
4 Problem solved: B7 is known as the lemon.
3 Approach succeeded: Identify elements.
2 Problem solved: Identify elements.
2 Problem: Pour gin into glass.
3 How do I pour gin into glass?

3 Expert suggests steps:
3 Step: Put gin above glass.
3 Step: Rotate gin.
3 Step: Move gin to the table.

3 Intention: Pour gin into glass.
4 Problem: Put gin above glass.
5 Intention: Put gin above glass.
6 Condition: Verify that nothing is on glass.
7 Method: Verify that nothing is on glass.
7 Just did: Verify that nothing is on glass.
6 Condition satisfied: Verify that nothing is on glass.
6 Condition: Verify that nothing is on gin.
7 Method: Verify that nothing is on gin.
7 Just did: Verify that nothing is on gin.
6 Condition satisfied: Verify that nothing is on gin.
6 Method: Move gin above glass.
6 Just did: Move gin above glass.
5 Approach succeeded: Put gin above glass.
4 Problem solved: Put gin above glass.
4 Problem: Rotate gin.
5 Intention: Rotate gin.
6 Method: Rotate gin.
6 Just did: Rotate gin.
5 Approach succeeded: Rotate gin.
4 Problem solved: Rotate gin.
4 Problem: Move gin to the table.
5 Intention: Move gin to the table.
6 Method: Move gin to the table.
6 Just did: Move gin to the table.
5 Approach succeeded: Move gin to the table.
4 Problem solved: Move gin to the table.
3 Approach succeeded: Pour gin into glass.
3 Helper exploited advice.
2 Problem solved: Pour gin into glass.
2 Problem: Pour vermouth into glass.
3 Intention: Pour vermouth into glass.
4 Problem: Put vermouth above glass.
5 Intention: Put vermouth above glass.
6 Condition: Verify that nothing is on glass.
7 Method: Verify that nothing is on glass.
7 Just did: Verify that nothing is on glass.
6 Condition satisfied: Verify that nothing is on glass.
6 Condition: Verify that nothing is on vermouth.
7 Method: Verify that nothing is on vermouth.
7 Just did: Verify that nothing is on vermouth.
6 Condition satisfied: Verify that nothing is on vermouth.
6 Method: Move vermouth above glass.
6 Just did: Move vermouth above glass.
5 Approach succeeded: Put vermouth above glass.
4 Problem solved: Put vermouth above glass.
4 Problem: Rotate vermouth.
5 Intention: Rotate vermouth.
6 Method: Rotate vermouth.
6 Just did: Rotate vermouth.
5 Approach succeeded: Rotate vermouth.
4 Problem solved: Rotate vermouth.
4 Problem: Move vermouth to the table.
5 Intention: Move vermouth to the table.
6 Method: Move vermouth to the table.
6 Just did: Move vermouth to the table.
5 Approach succeeded: Move vermouth to the table.
4 Problem solved: Move vermouth to the table.
3 Approach succeeded: Pour vermouth into glass.
2 Problem solved: Pour vermouth into glass.
2 Problem: Place lemon in glass.
3 Intention: Put lemon on glass.
  4 Condition: Space is for lemon on glass.
  5 Method: Verify that space is for lemon on glass.
  5 Just did: Verify that space is for lemon on glass.
  4 Condition satisfied: Space is for lemon on glass.
  4 Condition: Verify that nothing is on lemon.
  5 Method: Verify that nothing is on lemon.
  5 Just did: Verify that nothing is on lemon.
  4 Condition satisfied: Verify that nothing is on lemon.
  4 Method: Move lemon to glass.
  4 Just did: Move lemon to glass.
  3 Approach succeeded: Put lemon on glass.
  2 Problem solved: Place lemon in glass.
1 Approach succeeded: Mix martini.
1 Helper exploited advice.
0 Problem solved: Mix martini.
Appendix C: Problem solving knowledge

The knowledge used in mixing a martini, in order of use, is:

    // Insight
    If the problem is "Mix martini".
    Intention: Mix martini.
    The end.

    // Approach
    If the intention is "Mix martini".
    Step: Identify elements.
    Step: Pour gin into glass.
    Step: Pour vermouth into glass.
    Step: Place lemon in glass.
    The end.

    // Insight
    If the problem is "Identify elements".
    Intention: Identify elements.
    The end.

    // Approach
    If the intention is "Identify elements".
    Step: B5 is known as the glass.
    Step: B2 is known as the gin.
    Step: B1 is known as the vermouth.
    Step: B7 is known as the lemon.
    The end.

    // Insight
    If the problem is "XX is known as YY".
    Intention: XX is known as YY.
    The end.

    // Approach
    If the intention is "XX is known as YY".
    Method: XX is known as YY.
    The end.

    // Executor
    If the method is "XX is known as YY".
    Execute: Call jdremember with XX with YY.
    The end.

    // Insight
    If the problem is "Pour XX into YY".
    Intention: Pour XX into YY.
    The end.
// Approach
If the intention is "Pour XX into YY".
Step: Put XX above YY.
Step: Rotate XX.
Step: Move XX to the table.
The end.

// Insight
If the problem is "Put XX above YY".
Intention: Put XX above YY.
The end.

// Approach
If the intention is "Put XX above YY".
Condition: Verify that nothing is on YY.
Condition: Verify that nothing is on XX.
Method: Move XX above YY.
The end.

// Checker
If the condition is "Verify that nothing is on XX".
Check: Verify that nothing is on XX.
Instantiate: YY is supported by XX.
Solve: Get rid of YY.
The end.

// Executor
If the method is "Verify that nothing is on XX".
Execute: Call jdhascleartop with XX.
The end.
// Executor
If the method is "Move XX above YY".
Execute: Call jdputover with XX with YY.
The end.

// Insight
If the problem is "Rotate XX".
Intention: Rotate XX.
The end.

// Approach
If the intention is "Rotate XX".
Method: Rotate XX.
The end.

// Executor
If the method is "Move XX above YY".
Execute: Call jdtilt with XX.
Verify: Did "I rotate XX".
The end.

// Insight
If the problem is "Move XX to YY".
Intention: Move XX to YY.
The end.

// Approach
If the intention is "Move XX to YY".
Method: Move XX to YY.
The end.

// Executor
If the method is "Move XX to YY".
Execute: Call jdputon with XX with YY.
The end.

// Insight
If the problem is "Place XX in YY".
Intention: Put XX on YY.
The end.

// Approach
If the intention is "Put XX on YY".
Condition: Space is for XX on YY.
Condition: Verify that nothing is on XX.
Method: Move XX to YY.
The end.

// Checker
If the condition is "Space is for XX on YY".
Check: Verify that space is for XX on YY.
Instantiate: ZZ is supported by YY.
Solve: Get rid of ZZ.
The end.
// Executor
If the method is "Verify that space is for XX on YY".
Execute: Call jdfindspace with XX with YY.
The end.