# Towards an Architecture for Learning with Instruction

Shiwali Mohan and John E. Laird

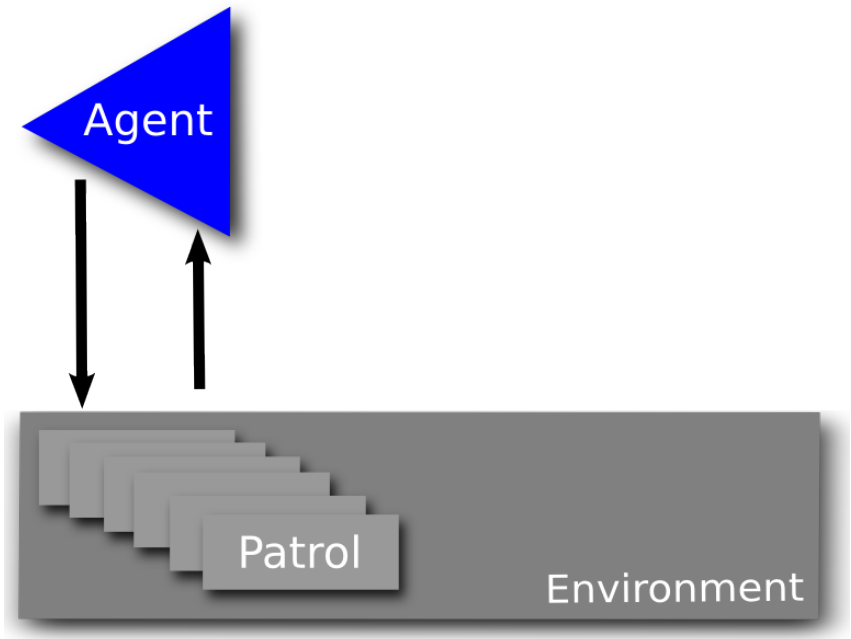
Computer Science and Engineering
University of Michigan

June 17th, 2011

# Outline

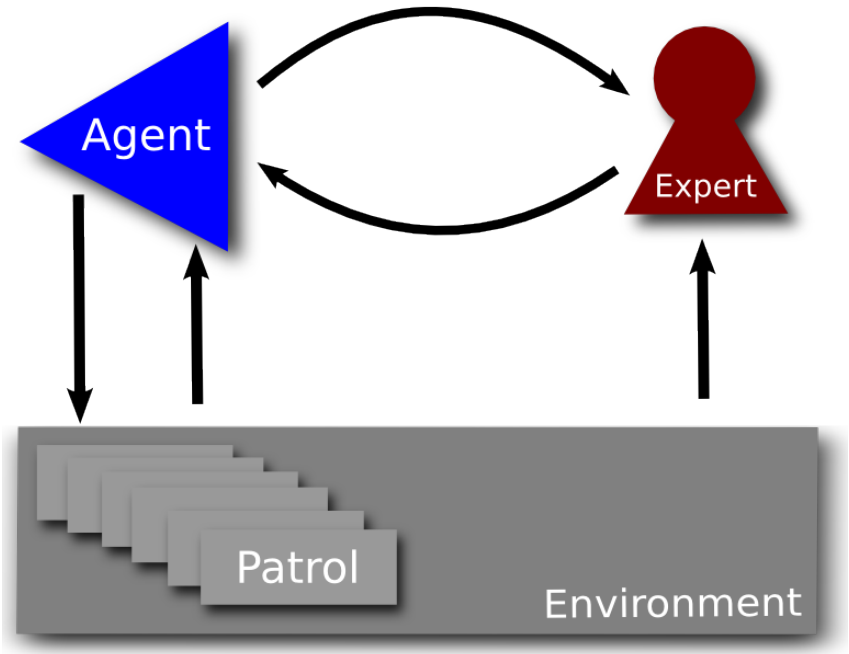
# Motivation

- Long living, general, intelligent agents
  - demonstrate a wide range of behavior
  - on a variety of task
  - knowledge must be added throughout the lifetime of the agent



# Motivation

- Long living, general, intelligent agents
  - demonstrate a wide range of behavior
  - on a variety of task
  - knowledge must be added throughout the lifetime of the agent
- Instruction
  - agent can be *taught* to learn new procedures
  - knowledge/information that can be trusted
  - situated, interactive instruction produces strong human learning[1]
  - semantic, procedural, problem solving etc




[1] Bloom, B. S. (1986). The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher*, 13(6):4–16

# Related Work

- Robo-Soar[1]
  - search guidance in case of an *operator tie*

[1] Laird, J. E., Yager, E. S., Hucka, M., and Tuck, C. M. (1991). Robo-Soar: An Integration of External Interaction, Planning, and Learning using Soar. *Robotics and Autonomous Systems*, 8(1-2):113–129

# Related Work

- Robo-Soar[1]
  - search guidance in case of an *operator tie*
- Instructo-Soar[2]
  - Identified properties of tutorial instruction, agent requirements
  - Situated explanation
  - learning through induction and inference, generalize from specific examples

[1] Laird, J. E., Yager, E. S., Hucka, M., and Tuck, C. M. (1991). Robo-Soar: An Integration of External Interaction, Planning, and Learning using Soar. *Robotics and Autonomous Systems*, 8(1-2):113–129

[2] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

# Related Work

- Robo-Soar[1]
  - search guidance in case of an *operator tie*
- Instructo-Soar[2]
  - Identified properties of tutorial instruction, agent requirements
  - Situated explanation
  - learning through induction and inference, generalize from specific examples
- Training Personal Robots Using Natural Language Instruction[3]
  - focus on natural language comprehension and *mapping*

[1] Laird, J. E., Yager, E. S., Hucka, M., and Tuck, C. M. (1991). Robo-Soar: An Integration of External Interaction, Planning, and Learning using Soar. *Robotics and Autonomous Systems*, 8(1-2):113–129

[2] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

[3] Lauria, S., Bugmann, G., Kyriacou, T., Bos, J., and Klein, A. (2001). Training Personal Robots Using Natural Language Instruction. *Intelligent Systems, IEEE*, 16(5):38–45

# Related Work

- Robo-Soar[1]
  - search guidance in case of an *operator tie*
- Instructo-Soar[2]
  - Identified properties of tutorial instruction, agent requirements
  - Situated explanation
  - learning through induction and inference, generalize from specific examples
- Training Personal Robots Using Natural Language Instruction[3]
  - focus on natural language comprehension and *mapping*
- Task Learning by Instruction: Benefits and Challenges for Intelligent Interactive Systems[4]
  - integration of various learning and reasoning components

---

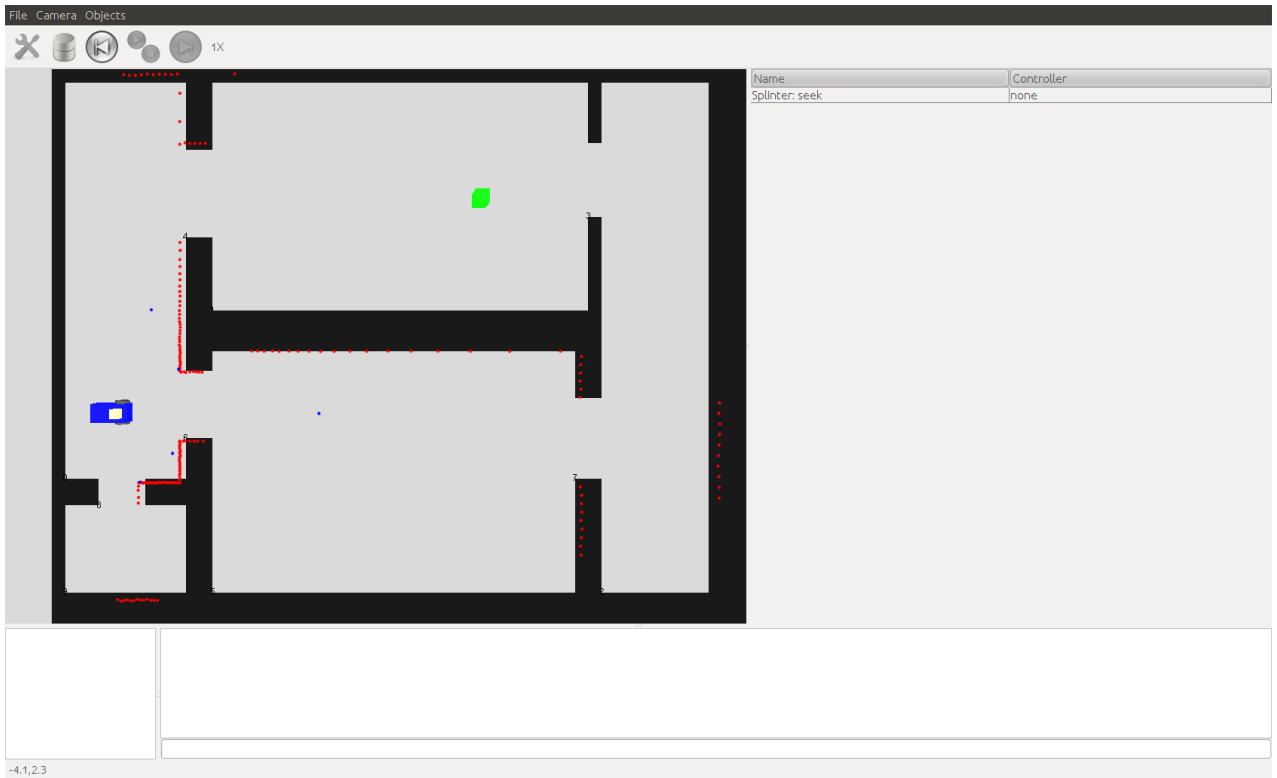
[1] Laird, J. E., Yager, E. S., Hucka, M., and Tuck, C. M. (1991). Robo-Soar: An Integration of External Interaction, Planning, and Learning using Soar. *Robotics and Autonomous Systems*, 8(1-2):113–129

[2] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

[3] Lauria, S., Bugmann, G., Kyriacou, T., Bos, J., and Klein, A. (2001). Training Personal Robots Using Natural Language Instruction. *Intelligent Systems, IEEE*, 16(5):38–45

[4] Blythe, J., Tandon, P., and Tillu, M. (2007). Task Learning by Instruction: Benefits and Challenges for Intelligent Interactive Systems. *American Association for Artificial Intelligence*

# Domain




**Given** a set of primitive actions:

```
go-to <area-id>, go-to-room <room-id>,
open-door <door-id>, close-door <door-id>,
turn-lights-on, turn-lights-off,
pick-up <object-id>, put-down <object-id>,
```

**Learn** *abstract* actions:

```
patrol [list-of-rooms]
turn-all-lights-off
get-object <object-id>
```

# Goals

- Identify and characterize different kinds of instructions in a complex environment
  - different kinds of knowledge, different modes of instruction

# Goals

- Identify and characterize different kinds of instructions in a complex environment
  - different kinds of knowledge, different modes of instruction
- Identify and characterize requirements on the agent design
  - general set of rules required to support instruction

# Goals

- Identify and characterize different kinds of instructions in a complex environment
  - different kinds of knowledge, different modes of instruction
- Identify and characterize requirements on the agent design
  - general set of rules required to support instruction
- Explore the utility of different long-term memories in a complex learning task

# Goals

- Identify and characterize different kinds of instructions in a complex environment
  - different kinds of knowledge, different modes of instruction
- Identify and characterize requirements on the agent design
  - general set of rules required to support instruction
- Explore the utility of different long-term memories in a complex learning task
- Develop a comprehensive model of learning that uses different reasoning and learning modules
  - inference, chunking, inductive reasoning, explanation-based generalization

# Goals

- Identify and characterize different kinds of instructions in a complex environment
  - different kinds of knowledge, different modes of instruction
- Identify and characterize requirements on the agent design
  - general set of rules required to support instruction
- Explore the utility of different long-term memories in a complex learning task
- Develop a comprehensive model of learning that uses different reasoning and learning modules
  - inference, chunking, inductive reasoning, explanation-based generalization
- Develop evaluation metrics

# Characterizing Instructions: Taxonomy

- Scope
  - Tutorial: instruct as the agent is acting [5]
  - Instruction Manual: instruct before the agent begins acting

[5] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

# Characterizing Instructions: Taxonomy

- Scope
  - Tutorial: instruct as the agent is acting [5]
  - Instruction Manual: instruct before the agent begins acting
- Application
  - When commanded: *patrol rooms A B and C*
  - Automated: *when non-friendly person is observed, report back*

[5] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

# Characterizing Instructions: Taxonomy

- Scope
  - Tutorial: instruct as the agent is acting [5]
  - Instruction Manual: instruct before the agent begins acting
- Application
  - When commanded: *patrol rooms A B and C*
  - Automated: *when non-friendly person is observed, report back*
- Context
  - Situated: *go-to-room A, turn-light-on*
  - Hypothetical: *imagine room A, the light-switch is on north wall*

[5] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

# Characterizing Instructions: Taxonomy

- Scope
  - Tutorial: instruct as the agent is acting [5]
  - Instruction Manual: instruct before the agent begins acting
- Application
  - When commanded: *patrol rooms A B and C*
  - Automated: *when non-friendly person is observed, report back*
- Context
  - Situated: *go-to-room A, turn-light-on*
  - Hypothetical: *imagine room A, the light-switch is on north wall*
- Communication
  - Agent Initiated: agent does not know how to proceed
  - Instructor Initiated: modify a learned procedure

[5] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

# Characterizing Instructions: Taxonomy

- Scope
  - Tutorial: instruct as the agent is acting [5]
  - Instruction Manual: instruct before the agent begins acting
- Application
  - When commanded: *patrol rooms A B and C*
  - Automated: *when non-friendly person is observed, report back*
- Context
  - Situated: *go-to-room A, turn-light-on*
  - Hypothetical: *imagine room A, the light-switch is on north wall*
- Communication
  - Agent Initiated: agent does not know how to proceed
  - Instructor Initiated: modify a learned procedure
- Task Structuring
  - Composite: learn *turn-all-lights-off*
  - Incremental: learn *patrol*, modify *patrol* to include *turning off lights*

[5] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

# Instruction Taxonomy

- Learning
  - Rote: memorize sequence of instructions
  - Generalized: learn general procedures

# Instruction Taxonomy

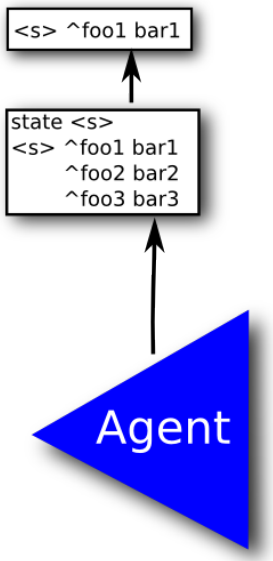
- Learning
  - Rote: memorize sequence of instructions
  - Generalized: learn general procedures
- Reasoning
  - Simple
  - Inference

# Instruction Taxonomy

- Learning
  - Rote: memorize sequence of instructions
  - Generalized: learn general procedures
- Reasoning
  - Simple
  - Inference
- Knowledge
  - Proposal and goal conditions: *turn-lights-off* when *in a room* and *light is on*
  - Control: *reporting non-friendly persons is more important than picking up objects*
  - Semantic: *objects with color=blue are dangerous*
  - Meta: arguments are sequential, random

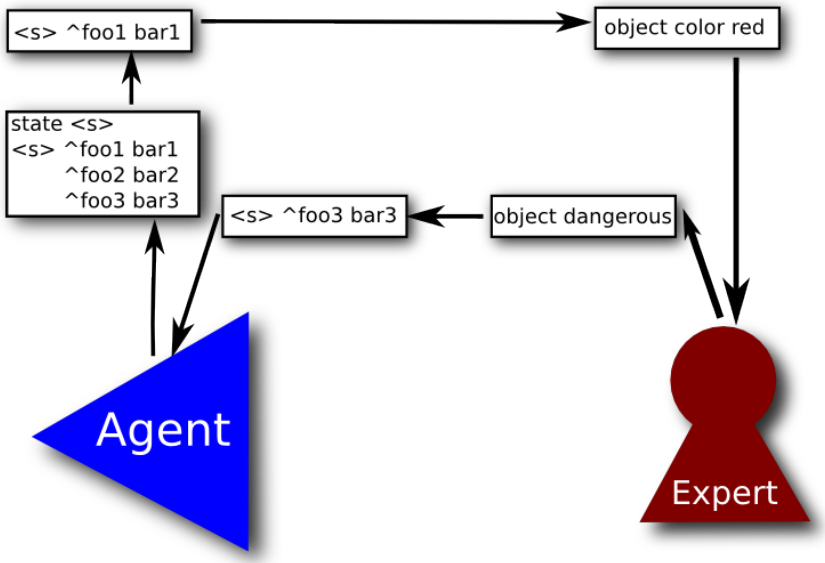
# Requirements on Agent Design

- The *Communication* Problem
  - The *Content* Problem
    - What information has to be communicated?
    - Model of the instructor
    - Assumption of shared environment, similar observations



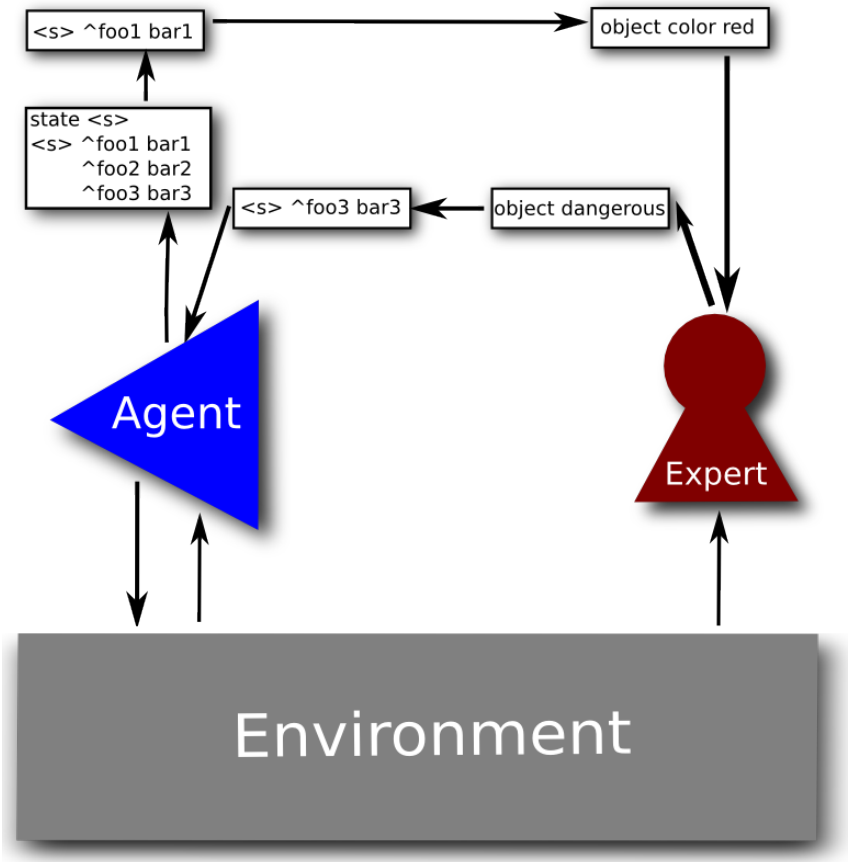

# Requirements on Agent Design

- The *Communication* Problem
  - The *Content* Problem
    - What information has to be communicated?
    - Model of the instructor
    - Assumption of shared environment, similar observations
  - The *Mapping* Problem
    - Common protocal between human and agent
    - Comprehending instructions
    - Generating discourse



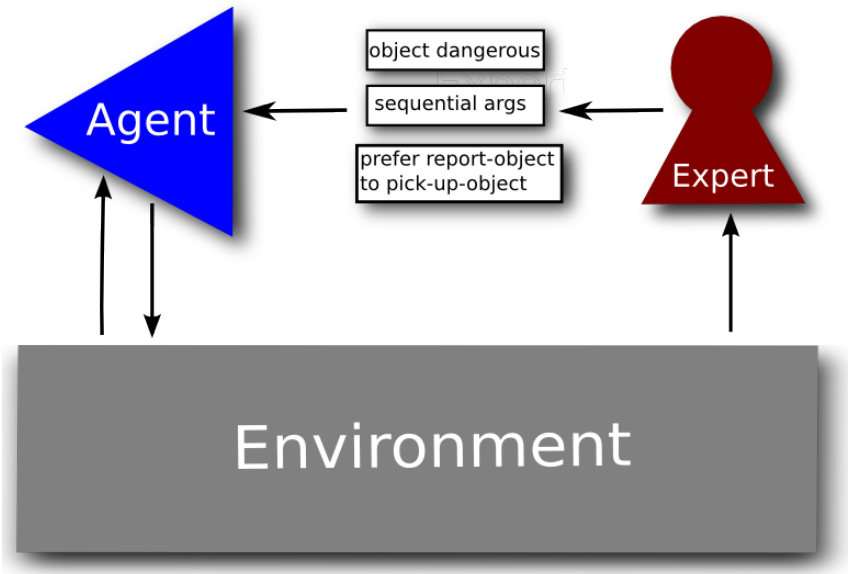
# Requirements on Agent Design

- The *Communication* Problem
  - The *Content* Problem
    - What information has to be communicated?
    - Model of the instructor
    - Assumption of shared environment, similar observations
  - The *Mapping* Problem
    - Common protocal between human and agent
    - Comprehending instructions
    - Generating discourse
  - The *Interaction* Problem
    - Maintaining a dialog
    - Integrating communication, learning and acting



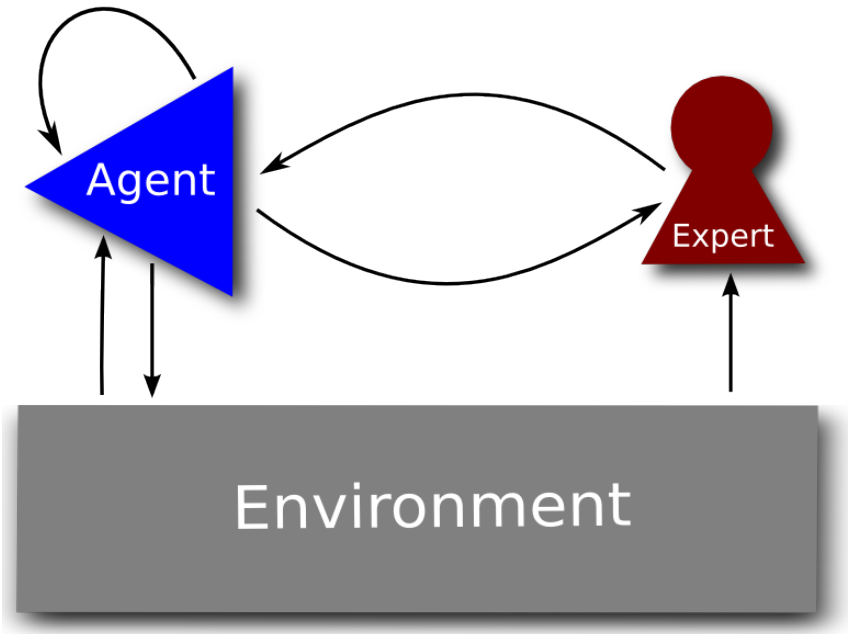
# Requirements on Agent Design

- The *Learning* Problem
  - The *Knowledge* Problem
    - Instruction may carry any type of knowledge (semantic, meta, control)
    - Applying knowledge to ongoing task in current context
    - Using one kind of knowledge to learn a different kind (semantic → procedural)



# Requirements on Agent Design

- The *Learning* Problem
  - The *Knowledge* Problem
    - Instruction may carry any type of knowledge (semantic, meta, control)
    - Applying knowledge to ongoing task in current context
    - Using one kind of knowledge to learn a different kind (semantic → procedural)
  - The *Transfer* Problem
    - Learning generally applicable knowledge
    - Transfer to appropriate conditions in future



# Learning from Instruction with Soar: An Example

- Learns abstract action, *patrol*

```
patrol room 1 room 2 room 3
go-to-room room 1, go-to-room room 2,
go-to-room room 3, go-to-room room 1
```

# Learning from Instruction with Soar: An Example

- Learns abstract action, *patrol*

```
patrol room 1 room 2 room 3
go-to-room room 1, go-to-room room 2,
go-to-room room 3, go-to-room room 1
```

- Instruction type

| Scope | Application | Communication | Context |
|---|---|---|---|
| tutorial | automated | agent-initiated | situated |

| Learning | Reasoning | Knowledge | Task Structuring |
|---|---|---|---|
| rote | - | - | - |

# Learning from Instruction with Soar: An Example

- Learns abstract action, *patrol*

```
patrol room 1 room 2 room 3
go-to-room room 1, go-to-room room 2,
go-to-room room 3, go-to-room room 1
```

- Instruction type

| Scope | Application | Communication | Context |
|---|---|---|---|
| tutorial | automated | agent-initiated | situated |

| Learning | Reasoning | Knowledge | Task Structuring |
|---|---|---|---|
| rote | - | - | - |

- Agent requirements
  - The *Mapping* Problem: hand-coded symbols
  - The *Content* Problem: pushed to the human
  - **The Interaction Problem**: agent-initiated communication
  - **The Knowledge Problem**: uses instructions to perform and learn *patrol*
  - The *Transfer* Problem: to be investigated later

# Learning from Instruction with Soar: An Example

- Learns abstract action, *patrol*

```
patrol room 1 room 2 room 3
go-to-room room 1, go-to-room room 2,
go-to-room room 3, go-to-room room 1
```

- Instruction type

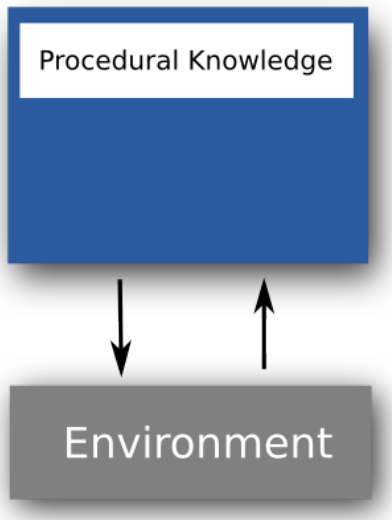
| Scope | Application | Communication | Context |
|---|---|---|---|
| tutorial | automated | agent-initiated | situated |

| Learning | Reasoning | Knowledge | Task Structuring |
|---|---|---|---|
| rote | - | - | - |

- Agent requirements
  - The *Mapping* Problem: hand-coded symbols
  - The *Content* Problem: pushed to the human
  - **The Interaction Problem**: agent-initiated communication
  - **The Knowledge Problem**: uses instructions to perform and learn *patrol*
  - The *Transfer* Problem: to be investigated later
- Soar architecture components
  - Procedural, semantic, working memory

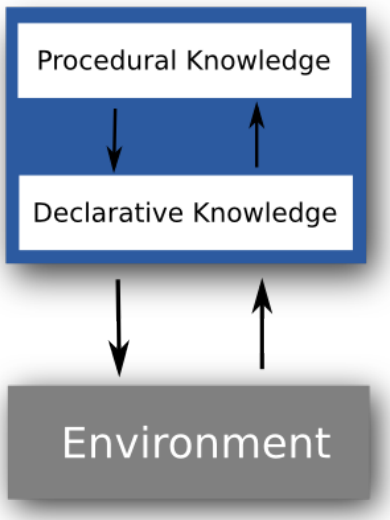
# Knowledge Levels

- Procedural Knowledge
  - Immediate application
  - *state-no-change*: implies lack of procedural knowledge for current state



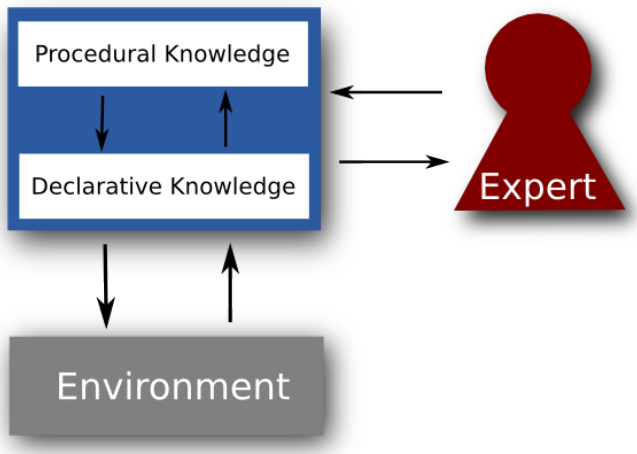
# Knowledge Levels

- Procedural Knowledge
  - Immediate application
  - *state-no-change*: implies lack of procedural knowledge for current state
- Semantic/Episodic Memory
  - Deliberate lookup
  - *retrieve/query failure*



# Knowledge Levels

- Procedural Knowledge
  - Immediate application
  - *state-no-change*: implies lack of procedural knowledge for current state
- Semantic/Episodic Memory
  - Deliberate lookup
  - *retrieve/query failure*
- Human Instructor
  - Deliberate questions
  - Incomplete/incorrect/specific knowledge
    - requires inference
    - generalization
    - *situated explanation* (Huffman and Laird, 1995)[1]




[1] Huffman, S. and Laird, J. (1995). Flexibly Instructable Agents. *Journal of Artificial Intelligence Research*, 3:271–324

# The Instruction Cycle

- *Detect* lack of knowledge to proceed further [knowledge]
  - *state-no-change* creates a *lookup-smem* subgoal
  - *retrieval* attempt may lead to a success/failure
  - if *success*, apply the retrieved command in *superstate*
  - if *failure*, store relevant information at the topstate [interaction]

# The Instruction Cycle

- *Detect* lack of knowledge to proceed further [knowledge]
  - *state-no-change* creates a *lookup-smem* subgoal
  - *retrieval* attempt may lead to a success/failure
  - if *success*, apply the retrieved command in *superstate*
  - if *failure*, store relevant information at the topstate [interaction]
- *Query*:
  - Create *query* using information stored in the *Detect* phase [content]
  - Generate discourse [mapping]

# The Instruction Cycle

- *Detect* lack of knowledge to proceed further [knowledge]
  - *state-no-change* creates a *lookup-smem* subgoal
  - *retrieval* attempt may lead to a success/failure
  - if *success*, apply the retrieved command in *superstate*
  - if *failure*, store relevant information at the topstate [interaction]
- *Query*:
  - Create *query* using information stored in the *Detect* phase [content]
  - Generate discourse [mapping]
- *Wait* for response from the instructor [interaction]

# The Instruction Cycle

- *Detect* lack of knowledge to proceed further [knowledge]
  - *state-no-change* creates a *lookup-smem* subgoal
  - *retrieval* attempt may lead to a success/failure
  - if *success*, apply the retrieved command in *superstate*
  - if *failure*, store relevant information at the topstate [interaction]
- *Query*:
  - Create *query* using information stored in the *Detect* phase [content]
  - Generate discourse [mapping]
- *Wait* for response from the instructor [interaction]
- *Parse* [mapping]

# The Instruction Cycle

- *Detect* lack of knowledge to proceed further [knowledge]
  - *state-no-change* creates a *lookup-smem* subgoal
  - *retrieval* attempt may lead to a success/failure
  - if *success*, apply the retrieved command in *superstate*
  - if *failure*, store relevant information at the topstate [interaction]
- *Query*:
  - Create *query* using information stored in the *Detect* phase [content]
  - Generate discourse [mapping]
- *Wait* for response from the instructor [interaction]
- *Parse* [mapping]
- *Assimilate*:
  - create appropriate *structures*
  - store in semantic memory using information stored in *Detect* phase [knowledge]

# The Instruction Cycle

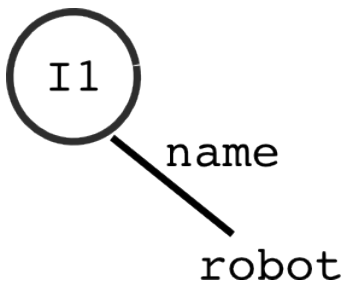
- *Detect* lack of knowledge to proceed further [knowledge]
  - *state-no-change* creates a *lookup-smem* subgoal
  - *retrieval* attempt may lead to a success/failure
  - if *success*, apply the retrieved command in *superstate*
  - if *failure*, store relevant information at the topstate [interaction]
- *Query*:
  - Create *query* using information stored in the *Detect* phase [content]
  - Generate discourse [mapping]
- *Wait* for response from the instructor [interaction]
- *Parse* [mapping]
- *Assimilate*:
  - create appropriate *structures*
  - store in semantic memory using information stored in *Detect* phase [knowledge]
- *Apply*:
  - recreate *state stack* [interaction]
  - apply the new piece of information [knowledge]

# The Instruction Cycle

- *Detect* lack of knowledge to proceed further [knowledge]
  - *state-no-change* creates a *lookup-smem* subgoal
  - *retrieval* attempt may lead to a success/failure
  - if *success*, apply the retrieved command in *superstate*
  - if *failure*, store relevant information at the topstate [interaction]
- *Query*:
  - Create *query* using information stored in the *Detect* phase [content]
  - Generate discourse [mapping]
- *Wait* for response from the instructor [interaction]
- *Parse* [mapping]
- *Assimilate*:
  - create appropriate *structures*
  - store in semantic memory using information stored in *Detect* phase [knowledge]
- *Apply*:
  - recreate *state stack* [interaction]
  - apply the new piece of information [knowledge]
- *Explain*: ? [transfer]

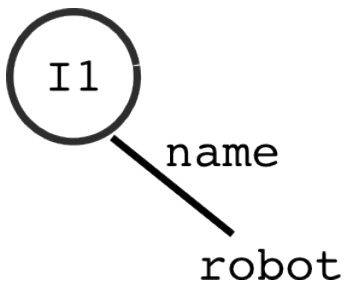
# Example Execution

- `S1 (robot): ?`
  `state-no-change`



# Example Execution

- `S1 (robot): ?`
  `state-no-change`
- `S2 (lookup-memory): retrieve-next`
  `failure`
  `S2 (lookup-memory): store-information`

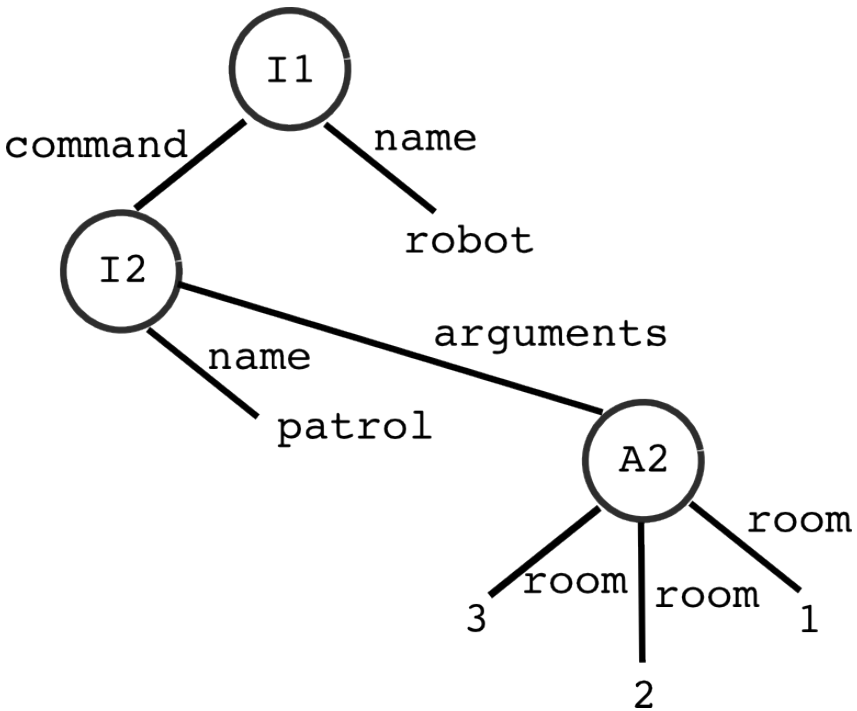


# Example Execution

- S1 (robot): ?
  state-no-change
- S2 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

  agent: "robot"
  instructor: "patrol room 1 room 2 room 3"

  S1 (robot): store-instruction

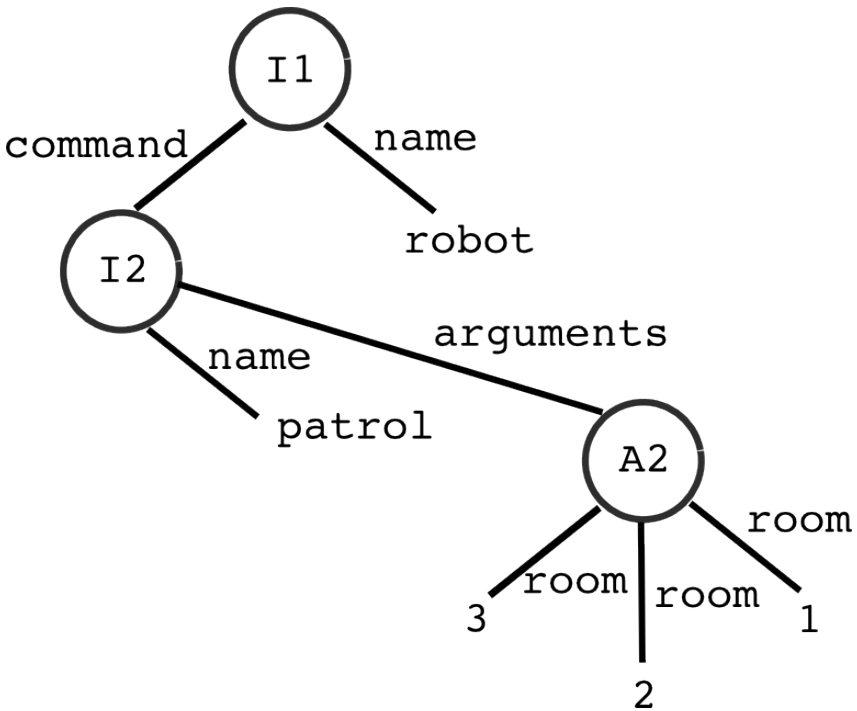


# Example Execution

- S1 (robot): ?
  state-no-change
- S2 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

  agent: "robot"
  instructor: "patrol room 1 room 2 room 3"

  S1 (robot): store-instruction
- S1 (topstate): ?
  state-no-change

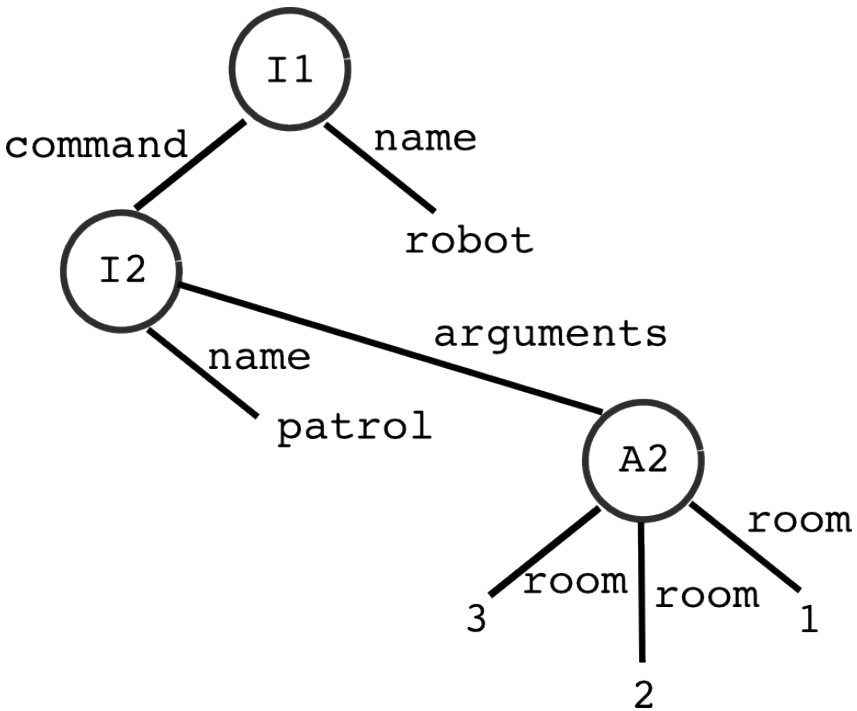


# Example Execution

- S1 (robot): ?
  state-no-change
- S2 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

  agent: "robot"
  instructor: "patrol room 1 room 2 room 3"

  S1 (robot): store-instruction
- S1 (topstate): ?
  state-no-change
- S2 (lookup-memory): retrieve-next
  success
  S2 (lookup-memory): propose patrol in S1

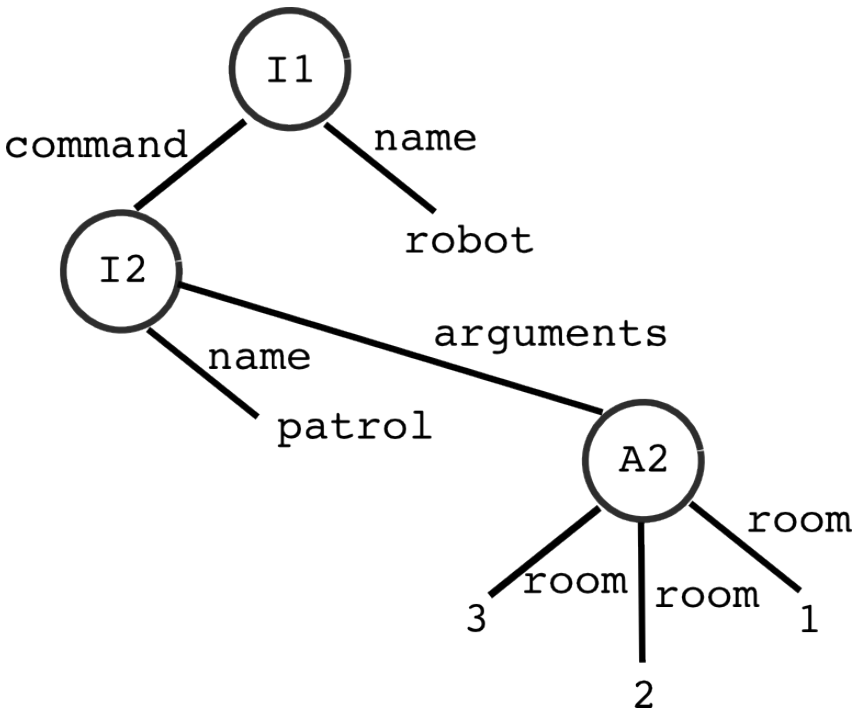


# Example Execution

- S1 (robot): ?
  state-no-change
- S2 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

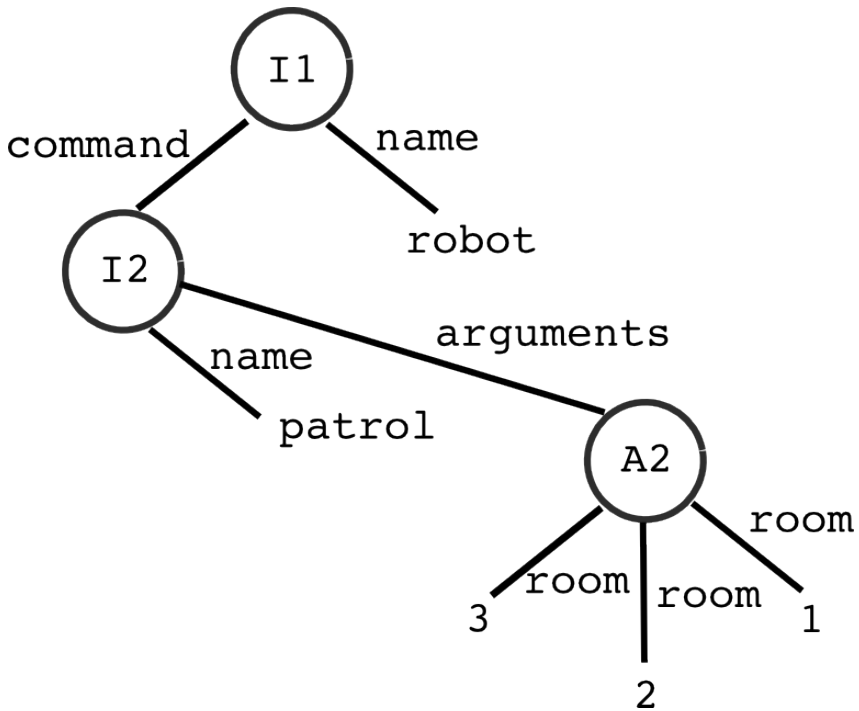
  agent: "robot"
  instructor: "patrol room 1 room 2 room 3"

  S1 (robot): store-instruction
- S1 (topstate): ?
  state-no-change
- S2 (lookup-memory): retrieve-next
  success
  S2 (lookup-memory): propose patrol in S1
- S1 (robot): patrol
  operator-no-change



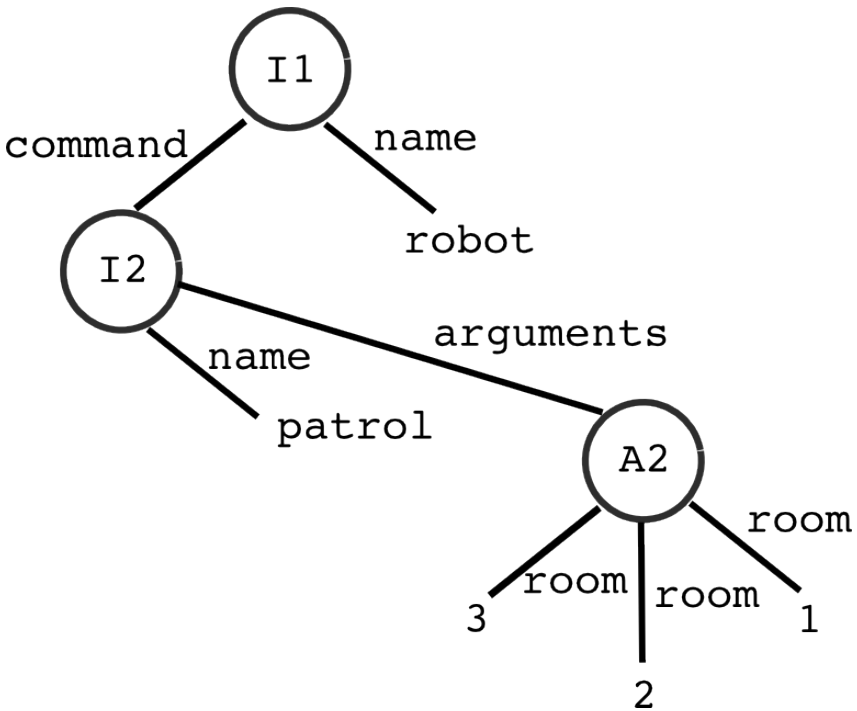
# Example Execution

- `S2 (patrol): ?`
  `state-no-change`



# Example Execution

- `S2 (patrol): ?`
  `state-no-change`
- `S3 (lookup-memory): retrieve-next`
  `failure`
  `S2 (lookup-memory): store-information`

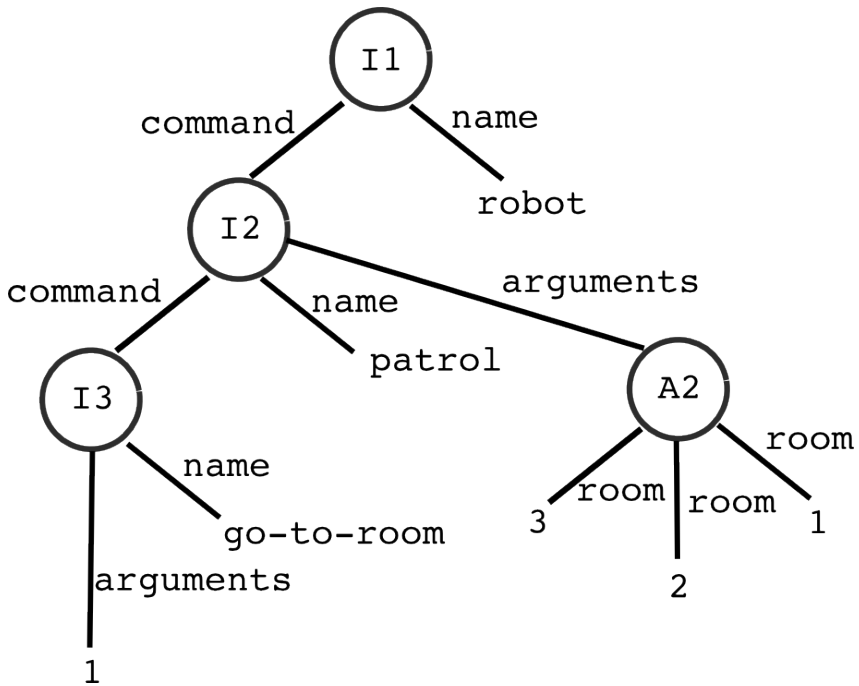


# Example Execution

- S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

  agent: "patrol"
  instructor: "go-to-room 1"

  S1 (robot): store-instruction

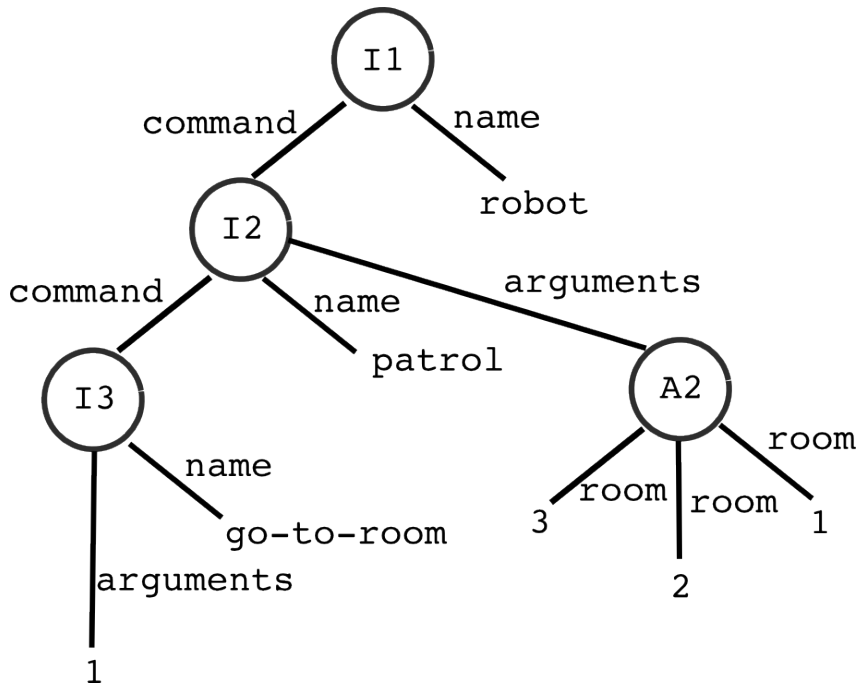


# Example Execution

- S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

  agent: "patrol"
  instructor: "go-to-room 1"

  S1 (robot): store-instruction
- S1 (topstate): ?
  state-no-change

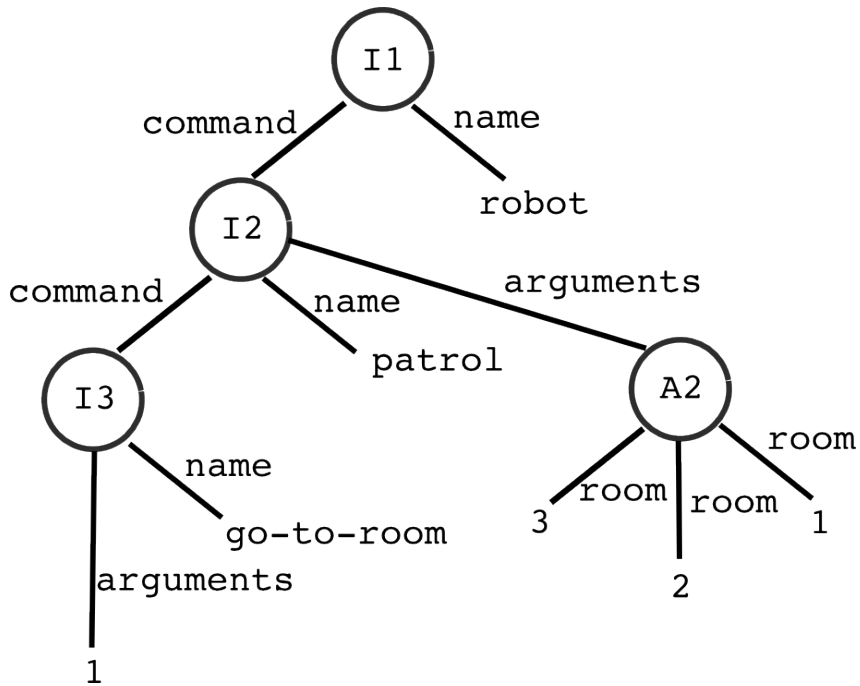


# Example Execution

- S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

  agent: "patrol"
  instructor: "go-to-room 1"

  S1 (robot): store-instruction
- S1 (topstate): ?
  state-no-change
- S2 (lookup-memory): retrieve-next
  success
  S2 (lookup-memory): propose patrol in S1

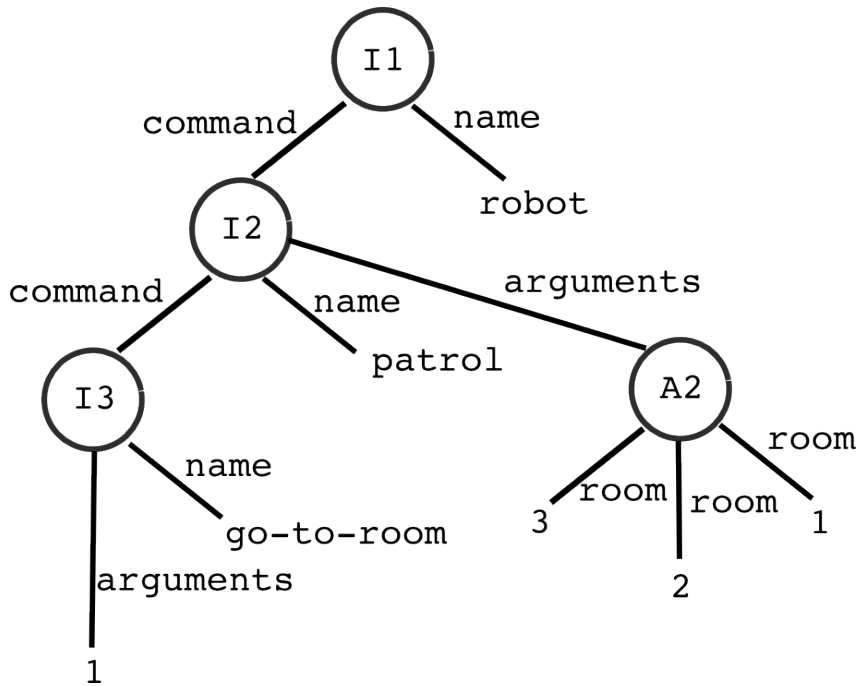


# Example Execution

- S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

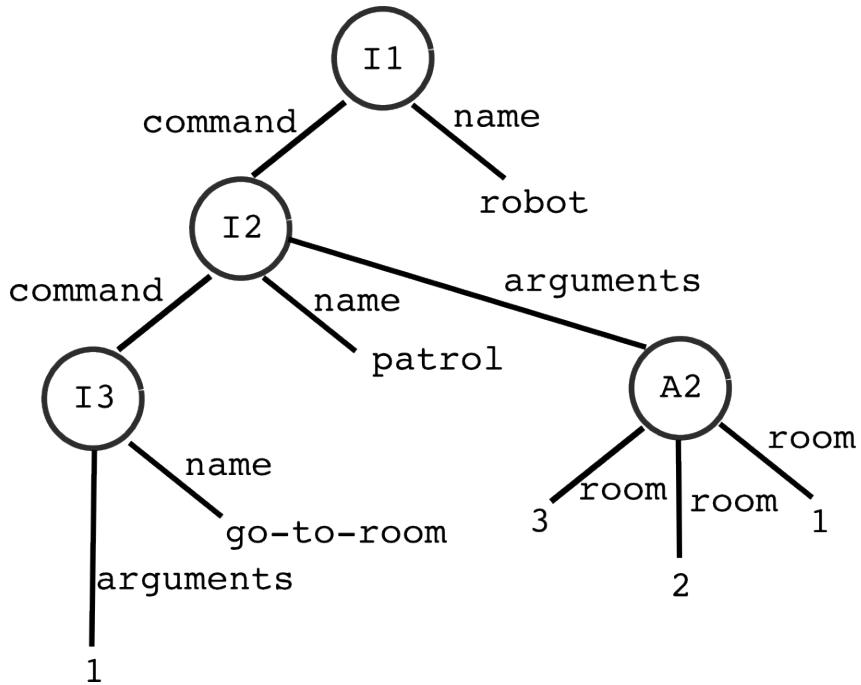
  agent: "patrol"
  instructor: "go-to-room 1"

  S1 (robot): store-instruction
- S1 (topstate): ?
  state-no-change
- S2 (lookup-memory): retrieve-next
  success
  S2 (lookup-memory): propose patrol in S1
- S1 (robot): patrol
  operator-no-change



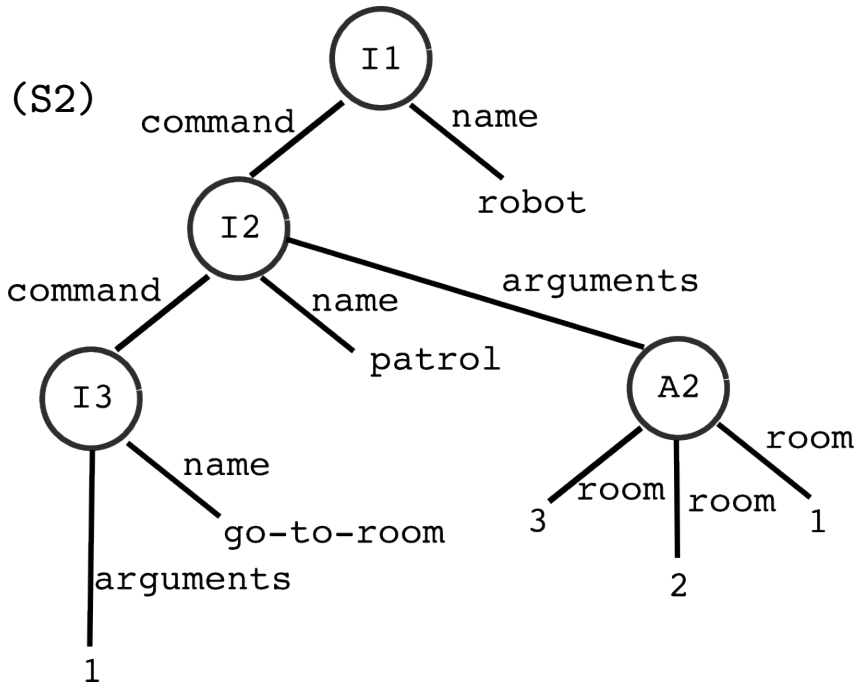
# Example Execution

- `S2 (patrol): ? state-no-change`



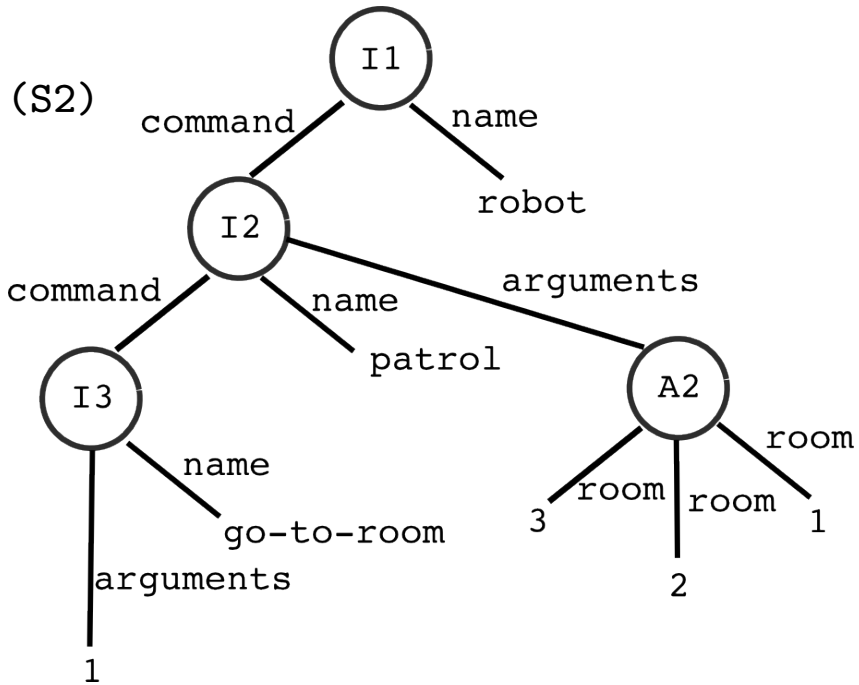
# Example Execution

- `S2 (patrol): ?`
  `state-no-change`
- `S3 (lookup-memory): retrieve-next`
  `success`
  `S2 (lookup-memory): propose go-to-room 1 (S2)`



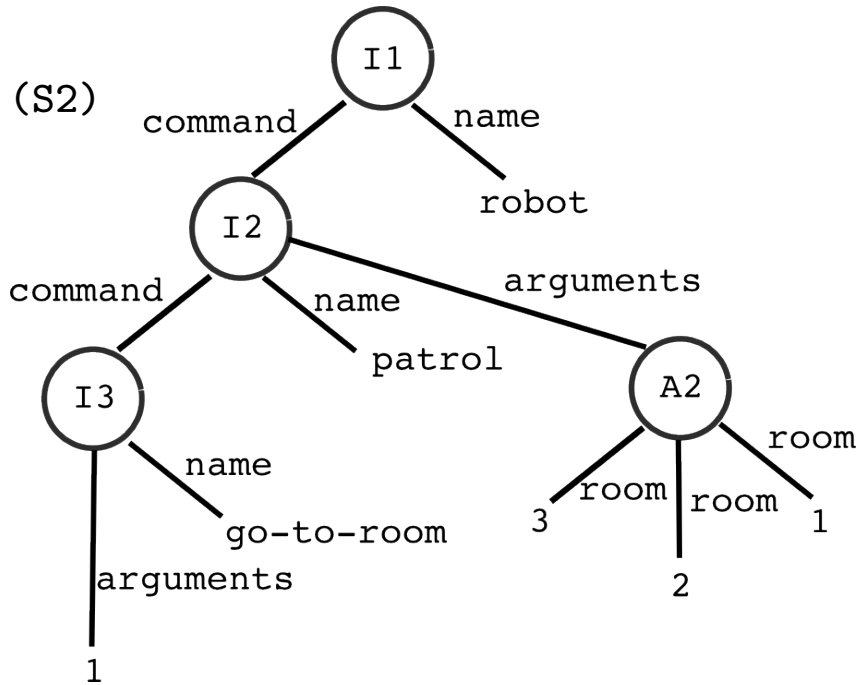
# Example Execution

- S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  success
  S2 (lookup-memory): propose go-to-room 1 (S2)
- S2 (patrol): go-to-room 1
  S2 (patrol): ?
  state-no-change



# Example Execution

- S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  success
  S2 (lookup-memory): propose go-to-room 1 (S2)
- S2 (patrol): go-to-room 1
  S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information

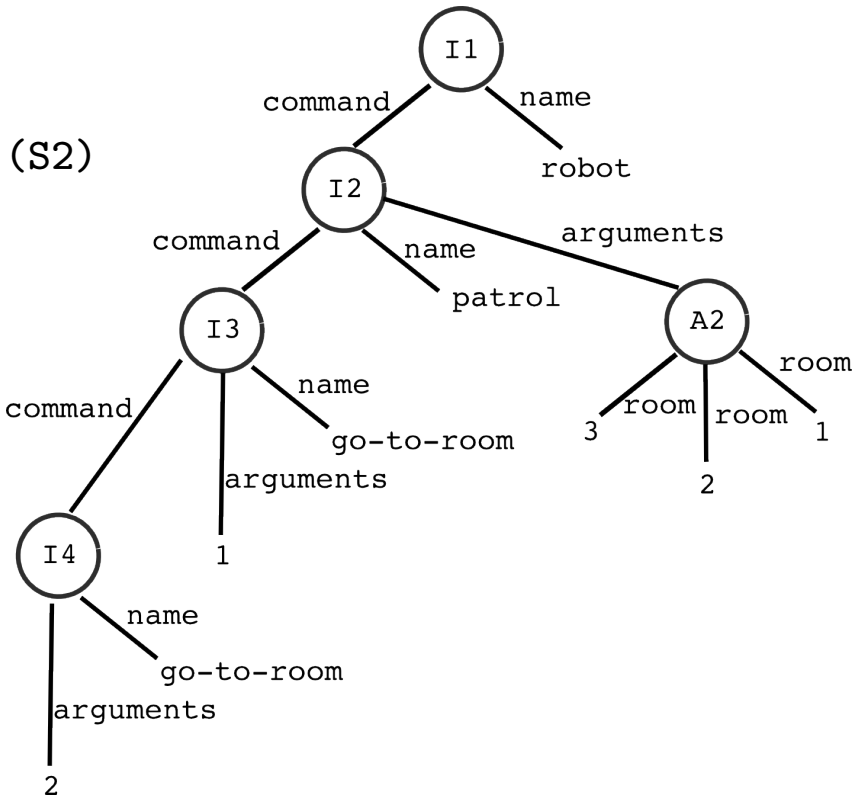


# Example Execution

- S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  success
  S2 (lookup-memory): propose go-to-room 1 (S2)
- S2 (patrol): go-to-room 1
  S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

  agent: "go-to-room 1"
  instructor: "go-to-room 2"

  S1 (robot): store-instruction

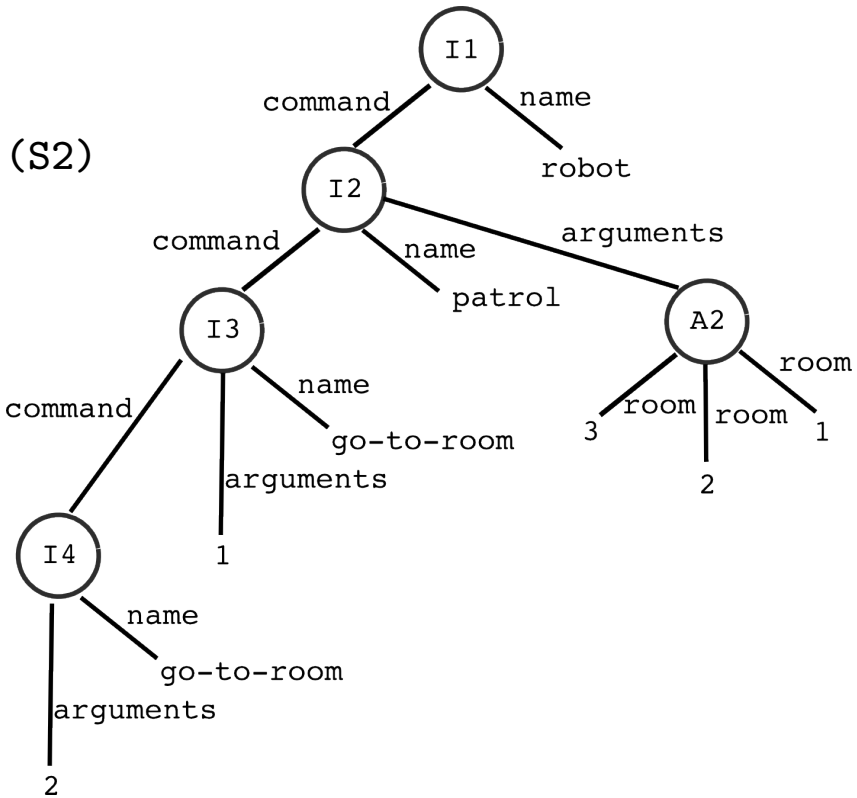


# Example Execution

- S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  success
  S2 (lookup-memory): propose go-to-room 1 (S2)
- S2 (patrol): go-to-room 1
  S2 (patrol): ?
  state-no-change
- S3 (lookup-memory): retrieve-next
  failure
  S2 (lookup-memory): store-information
- S1 (robot): ask-instructor

  agent: "go-to-room 1"
  instructor: "go-to-room 2"

  S1 (robot): store-instruction
- ...



# Conclusions

- Limitations
  - very limited in scope
  - learns by rote
  - agent initiated communication only

# Conclusions

- Limitations
  - very limited in scope
  - learns by rote
  - agent initiated communication only
- Future Work
  - Understanding and solving the *transfer* problem for this instruction set
  - Using episodic memory
  - More detailed investigation of various kinds of instruction

# Nuggets and Coal

- Nuggets
  - Have a proof of concept of how instruction can be used for acting
  - Learns ‘patrol’ and can learn other commands in a similar fashion
- Coal
  - Cannot learn general conditions yet
  - Not a good idea of what the goal of ‘patrol’ is