Multi-Agent Systems

Albert-Ludwigs-Universität Freiburg

Bernhard Nebel, Felix Lindner, and Thorsten Engesser

Winter Term 2018/19

GOAL Agents: Cognitive State

- GOAL agents are cognitive agents that maintain a cognitive state, which consists of:
  - Percepts: Received from the environment via the entity that the agent controls.
  - Messages: Received from other agents (speech acts).
  - Knowledge: Domain information and conceptual definitions that do not change over time.
  - Beliefs: Facts about the environment which may change over time.
  - Goals: (Partial) specifications of states the agent wants to bring about.

GOAL Agents: Core Abilities

- GOAL agents have three core abilities:
  - Event processing: Allows an agent to update its beliefs and goals based on percepts and messages.
  - Representing knowledge: Allows agents to represent and reason with its knowledge, beliefs, and goals.
  - Decision-making: Allows agents to select an action to perform next based on current beliefs and goals.
    - Action specifications: Inform agents about when an action can be performed and what its effects are.

Running Example Domain

- To exemplify the core concepts of GOAL, we will employ the Blocks World domain:
  - Blocks World consists of N numbered blocks $b1 \ldots bN$ and *table*.
  - Blocks can be stacked over oneanother by using $move(X,Y)$ actions. E.g., $move(b1,b2)$ would move block $b1$ on top of $b2$ if both blocks are clear.
  - A configuration in the Blocks World can be indentified with a set of facts of the form $on(X,Y)$. For each block X, the agent perceives one fact $on(X,Y)$.
  - Laws of Blocks World: At most one block is directly on top of another, a block cannot be directly on top of more than two other blocks etc.
  - Problem: Given some initial configuration of the blocks, the agent's task is to perform a sequence of stacking actions such that a configuration is achieved which satisfies the goal.

Initializing BlocksWorld Environment

- **use** “blocksworld-1.2.0.jar” **as environment with** start = [2, 3, 0, 5, 0, 7, 0].
- The environment is implemented as a Java program.
- The start parameter reads: b1 is on top of b2, b2 is on top of b3, b3 is on the table etc.

Initializing the Agent

```
define stackAgent as agent {
        use stackBehavior as main.
}
```

Connecting Agent to Environment

```
launchpolicy {
        when name=gripper launch stackAgent.
}
```

Actually Creating Cognitive States in GOAL

- Cognitive State
  - Informational State
    - Knowledge base: **use** ⟨prologfile⟩ **as knowledge.**
    - Belief base: **use** ⟨prologfile⟩ **as belief.**
  - Motivational State
    - Goal base: **use** ⟨prologfile⟩ **as goal.**

Domain Knowledge

- We define the predicate block/1 to make explicit that everything that is on top of something is a block:
  - block(X) :- on(X, _).
  - (A block which is not on top of another block is on the table. Hence, every block will qualify as a block according to this rule.)
- We define the predicate clear/1 to identify those entities which have no other block on top of them, i.e., those on top of which other blocks can be stacked. Any block can always be put on the table, therefore, the table also is clear according to this reading.
  - clear(table).
  - clear(X) :- block(X), not(on(_, X)).
- As on/2 is only used in bodies, we have to specify on/2 as dynamic:
  - :-dynamic on/2.

Floundering

- We cannot write this: clear(X) :- not(on(_, X)), block(X).
- What goes wrong: Negation is applied to non-ground atom (viz., variable X is not yet instantiated). However, not-operator does not bind any variables.
- A Prolog program that applies to a non-ground literal is said to flounder.

Closed-World Assumption

- Negation-as-Failure yields that everything not stated is assumed to be false. E.g., the absence of information about any blocks on, say, block 1 yields the inference that block 1 is clear.
- Cotrast this with entailment you know from e.g. propositional or modal logics.

Conceptual Knowledge

- We define the concept of a tower as a (possibly singleton) sequence of blocks stacked over another.
  - tower([X]) :- on(X, table).
  - tower([X, Y | T) :- on(X, Y), tower([Y | T]).
- I.e.: Any block directly on the table is a tower. If something is a tower, then the something which results from stacking a block on top of that tower is again a tower.
- Note that if [X | T] is a tower, this does not mean that X is clear (other blocks may extend the tower to yet a bigger one).

Starting the GOAL Agent

- Starting the GOAL agent opens a visualization of the environment.
- The stackAgent receives percepts on(b1, b2), ...
- The stackAgent lacks beliefs and goals. Therefore, nothing happens.

Building Beliefs from Percepts

- Add event module to agent definition, e.g., like this:
  use stackEvents **as event.**
- Add a belief base to event module. Add the knowledge base as knowledge base.
- Write into event module:

forall bel(on(X, Y)), **not**(**percept**(on(X, Y))) **do delete**(on(X, Y)).
forall percept(on(X, Y)), **not**(**bel**(on(X, Y))) **do insert**(on(X, Y)).

Adding Goals

- First add an init module which gets executed once after the agent started.
- Declare knowledge base.
- Write: **if true then adopt**(on(b1, b5), on(b2, table), on(b3, table), on(b4, b3), on(b5, b2), on(b6, b4), on(b7, table)).

Goals in GOAL

- Goals must be ground.
- Conjunctive goals: What's the difference?
  - *Goal*$_1$: on(2, 1). on(3, 2).
  - *Goal*$_2$: on(2, 1), on(3, 2).
- Achievement goal
  - **a-goal**(qry) = **goal**(qry), **not**(**bel**(qry))
- Sub-Goals Achieved:
  - **goal-a**(qry) = **goal**(qry), **bel**(qry)
- Blind commitment
- Explicit drop-action, adopt action

Decision Making

- Add goal prolog file.
- Declare goal file in main module.
- Implement simple strategy:

if a-goal(tower([X, Y | T])), **bel**(tower([Y|T])) **then** move(X, Y).
if a-goal(tower([X | T])) **then** move(X, table).

Action Specifications

- Add an action specification for action move(X,Y).
- Declare knowledge base.
- Define action's pre- and post-conditions. Like this:

  define move(X, Y) **with**

    pre{clear(X), clear(Y), on(X, Z)}

    post{not(on(X, Z)), on(X, Y)}
- Add actionspec to main module.
- Notes
  - Different meaning of **not**
  - Relation to add- and delete-lists in STRIPS
  - Instantaneous vs. Durative Actions — when to specify post-conditions and when better to use just **true**

Literature

Hindriks, K. V., Programming Cognitive Agents in GOAL, Technical Manual, 2017, https://goalapl.atlassian.net/wiki/.