Agent Oriented Programming

key concepts and principles related to programming agents as mental entities.

### ****Introduction to AOP****

AOP is based on ascribing mental qualities (such as beliefs, desires, and intentions) to programs and agents.

**ascribing** mental qualities to programs or agents means assigning or attributing abstract mental states (like beliefs or goals) to these computational entities.

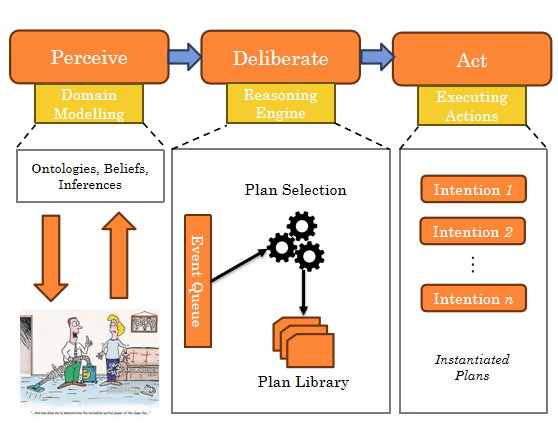
### 2. ****Why Ascribe Mental Qualities?****

* It is easier to reason about a program's goals and beliefs rather than analyzing its technical state.
* Mental models, such as belief structures, closely align with the design intent of the program, aiding debugging and enhancing comprehension.

### 3. ****AgentSpeak(L) (Rao, 1995)****

* AgentSpeak(L) is a language designed for BDI (Belief-Desire-Intention) agents, which process events based on their beliefs and predefined plans.

An AgentSpeak(L) agent consists of:

* a set of beliefs: predicate logic statements that define the state of the environment.
* a set of plans: context-sensitive recipes that map events to a sequence of steps in a context.
* an event queue: ordered list of events that model external (environment) or internal (reasoning) events and which are linked to the adoption (+) or retraction (-) of state.
* Agents process events sequentially from the event queue, selecting a plan that has adopted an intention.
* 

### ****Perceive-Deliberate-Act Cycle****

This is the core cycle that drives the agent's behavior. It's a continuous loop of **perceiving**, **deliberating**, and **acting**, which allows the agent to react to changes in the environment and make decisions.

**Perceive (Sense the Environment)**

* 1. In the perceive phase, the agent **monitors its environment** for changes or new information. The environment could include external factors (such as sensor readings, messages, or actions of other agents) or internal factors (such as changes to its own beliefs or goals).

**Deliberate (Reason and Decide)**

* 1. After perceiving the event changes in the environment, the agent enters the deliberate phase. Here, it **processes the information** it gathered during the perceive phase and updates its beliefs(aka domain model and event queue).
  2. it searches through its plan library to find a plan that matches the event and the current context (beliefs and goals). If a plan is found, the agent selects it as the best course of action to follow..

**Act (Execute Actions)**

* 1. In the act phase, the agent **executes actions** based on the selected plan (therefore **the selected plan becomes an intentio**n) from the deliberate phase. These actions may involve interacting with the environment (e.g., turning on a light) or changing its internal state (e.g., updating its beliefs).

This **Perceive-Deliberate-Act cycle** repeats continuously. As the environment changes, the agent responds by perceiving the new state, deliberating on the appropriate response, and acting accordingly.

**Plans** are a **predefined sequence of steps (actions)** to be executed by the agent and Plans can be made of:

* Private Actions: things that can be directly executed by the agent.
* Achievement Goals: decision points that are defined in terms of things the agent would like to happen; can be future state (intention-to-be) or future activities (intention-to-do) => Denoted with an exclamation mark (!)
* Test Goals: decision points that are defined in terms of the current state. =>Denoted with a question mark (?)
* Test and Achievement goals generate internal events that are added to the event queue.

**Structure**: Each plan consists of:

* **Triggering events**: External (e.g., change in environment) or internal (e.g., a goal is adopted).
* **Context**: The conditions under which the plan can be executed (based on the agent’s beliefs).
* **Actions**: The sequence of steps or tasks the agent should perform to handle the event.

The general syntax for a plan rule is:

+ event : context <- action; action .

* +event: A **triggering event**. This event is something the agent perceives, either from the environment or internally. The + symbol indicates the adoption of a new belief or event.
* **context condition:** it specifies that this plan should only be executed if the condition is met togheter with the triggering event
* <-: This is the "if-then" separator, which means that when the triggering event occurs, the agent should carry out the actions on the right-hand side.
* actions: These are the **steps** the agent should take when the plan is triggered.

**Intentions** are the specific **plans the agent has committed to executing**.

Once a plan is selected during the Deliberate phase, it becomes an Intention, and the agent commits to executing it **step by step** in the Act phase.

A step can be:

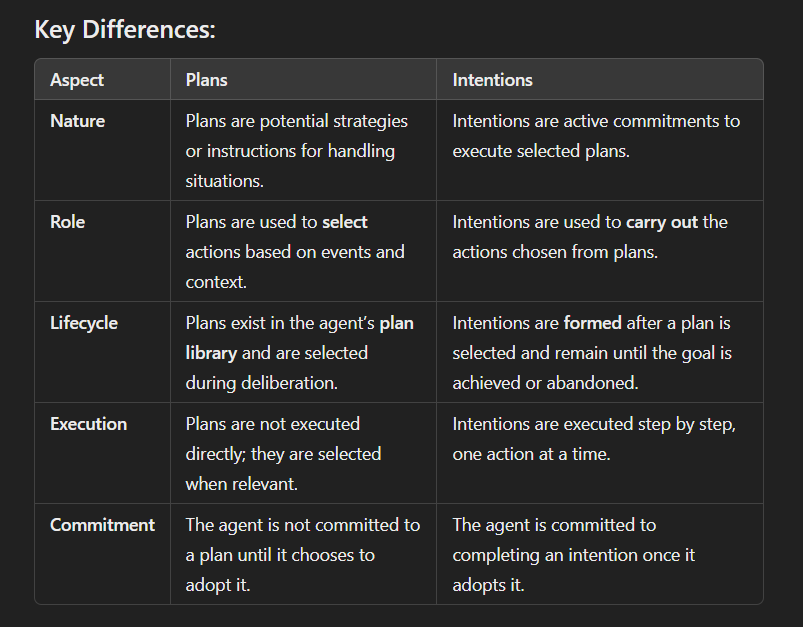
* query or change the beliefs
* perform an action on the external world
* suspend the execution until a certain condition is met
* submit new goals.

Intentions are executed **sequentially** over time

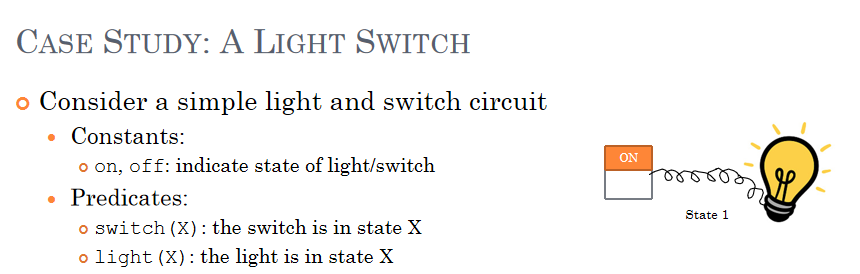
The operations performed by a step may generate new events, which, in turn, may start new intentions.

An intention succeeds when all its steps have been completed. It fails when the execution of an associated action reports errors and the agent may adopt a new intention or retry.

Difference between Intention and Plans in AgentSpeak Perceive-Deliberate-Act cycle System:



### ****Case Study: Light Switch****



* This example demonstrates how an agent can control a simple light switch by updating its beliefs based on changes in the environment (i.e., the state of the switch).
* The agent maintains rules such as:
  + If the switch is turned on, and the light is off, it updates the light to be on.
  + Similarly, for switching off.

In other words we can say that there are 2 desirable (acceptable) states:

State 1: switch(on), light(on)

State 2: switch(off), light(off)

Therefore we can say that the agent objective is to ensure that the light is in the correct state given the state of the switch.

Perceive-Deliberate-Act Cycle:

The System initial state is State 2

· **Perceive**: The agent notices a change in the environment, such as the switch being turned on.

· **Deliberate**: The agent searches for a relevant plan, finds one that addresses the event (the switch being turned on), and selects it. This plan specifies actions like "turn the light on" if the light is currently off.

· **Act**: The selected plan becomes an **intention**. The agent commits to turning the light on and begins executing the steps outlined in the intention, updating its beliefs as it acts.

So in AgentSpeak syntax:

Initlial State: switch(off), light(off);

Plan: +switch(on): light(off) <- -light(off);+light(on).

+switch(off) : light(on) <- -light(on);+light(off).

Practical Reasoning with AgentSpeak

This presentation emphasizes the concepts of **deliberation** and **means-end reasoning**, which are central to practical reasoning and agent decision-making.

#### 2. ****AgentSpeak(L) and BDI****:

FROM

BDI (old)

* Beliefs: the current state of the environment
* Desires: the agent ideal future state of the environment
* Intentions: subset of the desires that the agent commits to

To

BDI with AgentSpeak

* **Beliefs**: the current state
* **Desires**: represented as goals (assumed to be mutually consistent)
* **Intentions**: represented as plans that have been adopted to achieve goals

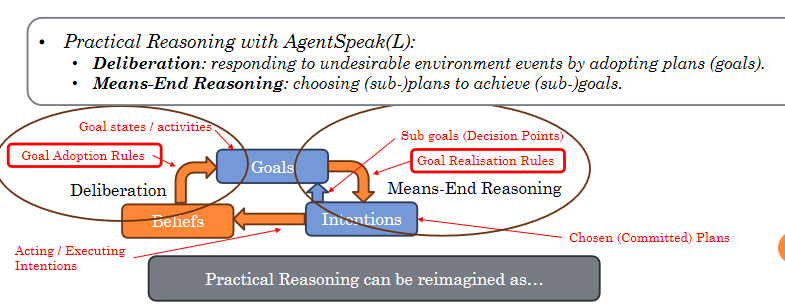
#### ****2.5 Practical Reasoning in AgentSpeak(L)****:

From:

### ****Practical Reasoning with BDI****

* 1. **Deliberation**: identifying what desires you are committed to achieving (your intentions)
  2. **Means-ends Reasoning**:adopting plans or actions to achieve your intentions.

To:



* This involves specifying goals as future states that the agent must achieve. When an undesirable event occurs, the agent must reason through how to handle the event and bring about a desired outcome.

### ****The Light Switch Revisited (Slides)****:

Now, let’s dive into the **Light Switch Revisited** section, which revisits the light switch example to illustrate practical reasoning in AgentSpeak(L).

#### ****Light Switch (Revisited) - Original Program****:

switch(off)light(off)

+switch(on) <- -light(off); +light(on).

+switch(off) <- -light(on); +light(off).

* **Explanation**: This program describes how the agent reacts when the switch is turned on or off. When the switch is turned on, the agent turns the light on, and when the switch is turned off, the agent turns the light off.
* **Problem**: This version assumes the light is always off when the switch is turned on and always on when the switch is turned off. It doesn't account for the case when the light is already on or off, leading to potential redundancy.

#### ****Identifying the Problem****:

* The issue with the original rule is that it assumes the light is in the opposite state whenever the switch changes. For example, when the switch turns on, the program assumes the light is off, which might not always be true.
* The key insight here is that the solution doesn't separate **what the agent wants to achieve** (i.e., turning the light on) from **how it achieves it** (i.e., flipping the switch).

#### ****Light Switch (Alternative)****:

(Domain Model)

switch(off)light(off)

(Goals Rules)

+switch(on) <- !light(on).

+switch(off) <- !light(off).

* **Explanation**: This alternative version introduces **goals** into the program, using the syntax !light(on) and !light(off). This means that instead of directly performing actions, the agent first **adopts a goal** (i.e., ensuring the light is on or off) and then finds a way to achieve it (Achievement Goals).
* The agent no longer makes assumptions about the state of the light but focuses on achieving a desired state.

#### ****Defining Rules to Achieve Goals****:

* The new program defines **rules** that describe how to achieve the goals, such as turning the light on or off.
* This approach distinguishes between **what the agent wants to achieve** (the goal) and **how to achieve it** (the specific actions).

#### ****Detailed Rule Definition to achieve these Goals****:

(Plan Rules)

+!light(on) : light(off) <- -light(off); +light(on).

+!light(on) <- .

(implied that the light is already on, therefore do nothing)

+!light(off) : light(on) <- -light(on); +light(off).

+!light(off) <- .

* **Explanation**: For each goal (!light(on) or !light(off)), we have to add to the program two rules:
  1. One rule specifies **how** to achieve the goal if the light is in the wrong state.

A second rule simply confirms that the goal is already achieved (i.e., no action is necessary if the light is already on or off).

* If no applicable plan is found for a goal, the goal is considered to have failed.

#### ****Program Structure and Efficiency****:

* The revised program now has **six rules**, as opposed to the original two. While this makes the code more **modular** and **goal-driven**, it also introduces more complexity.
* The two additional rules handle "serendipity"—the case where the goal has already been achieved unexpectedly (e.g., the light is already in the desired state).

#### ****Simplification****:

From:

+switch(on) <- ! light(on).

+switch(off) <- ! light(off).

to

+switch(S) <- !light(S).

From:

+ ! light(on) <- .

+ ! light(off) <- .

to

+!light(S) <- .

* The goal here is to simplify the program while maintaining clarity. Instead of using separate rules for turning the light on and off, we use **variables** to represent the state (S), allowing the same rule to handle both cases.
* The program can now handle any switch state (on or off) in a single rule using the variable S.

#### ****Simplifying with Transitions****:

transition(off on)

This transition describes the action of turning the light **from "off" to "on"**

transition(on off)

* This slide introduces the idea of **transitions** as beliefs(predicates), allowing the agent to model the transitions between states.
* A transition defines how the system moves from one state to another, such as turning the light from "off" to "on" or vice versa.

#### ****Final Simplified Program****:

From:

+ ! light(on) : light(off) <- - light(off); +light(on).

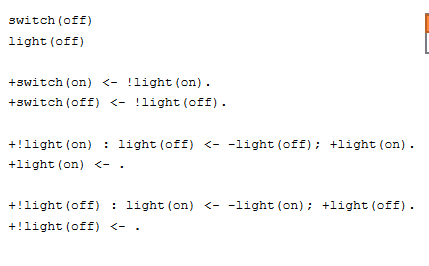
+ ! light(off) : light(on) <- - light(on); +light(off).

to

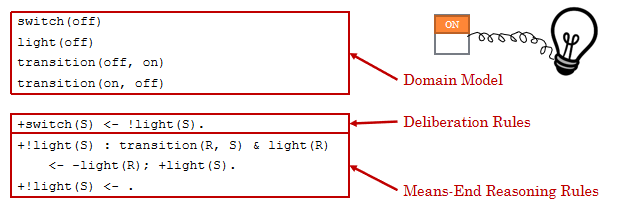
+!light(S) : transition(R S) & light(R) <- -light(R); +light(S).

* In this final version, the agent uses **transitions** to manage the state of the light. It checks if there is a valid transition between the current state (R) and the desired state (S), and then updates the light's state accordingly.
* This structure captures the **essence of means-end reasoning** by separating what the agent wants to achieve (the goal) from how to achieve it (the transition).

Therefore from



To



PS: i have put the transition(on-off) (off-on) in the domain model because it represent the possible

Transitions of state that i can have,

If im doing transition( on-on), its not actually a transition in my domain, therefore the plan resolution !light doesn't start.

#### . ****Means-End Reasoning and Sub-Goals****

* **Means-End Reasoning** is a key concept in practical reasoning where agents determine how to achieve their goals by adopting appropriate sub-goals. Sometimes, preconditions need to be satisfied before a goal can be pursued.

+!mygoal(X) <- !has(X); !mygoal(X).

* If a necessary condition (e.g., having an object) is not met, the agent first adopts a sub-goal to satisfy the condition before attempting to achieve the main goal.

#### 7. ****Lists in AgentSpeak(L)****

* The presentation also discusses the use of **lists** in AgentSpeak(L), which can be useful for representing sequences or collections of items.

tower([c b a])

* The agent can use lists to reason about ordered tasks, such as building a tower of blocks. This extends the agent’s reasoning abilities to handle more complex scenarios.

### Conclusion:

The presentation on **Practical Reasoning with AgentSpeak(L)** highlights the importance of combining **deliberation** and **means-end reasoning** to build rational agents capable of achieving their goals efficiently. By improving the light switch example through the introduction of goals, transitions, and variable states, the presentation demonstrates how to design more sophisticated agents that avoid redundant actions and better adapt to dynamic environments.

The key takeaway is that practical reasoning in agents involves not only deciding what goals to pursue, but also determining the most efficient way to achieve those goals by reasoning about transitions, preconditions, and sub-goals. This leads to more flexible, scalable, and maintainable agent behavior.