SOFTWARE AGENTS

Lec8- June 2020

dr. abbas akram khorsheed

A TYPOLOGY OF AGENTS

A typology refers to the study of types of entities. There are several dimensions to classify existing software agents. Firstly, agents may be classified by their mobility, i.e. by their ability to move around some network. This yields the classes of static or mobile agent.

Secondly, they may be classed as either deliberative or reactive. Deliberative agents derive from the deliberative thinking paradigm: the agents possess an internal symbolic, reasoning model and they engage in planning and negotiation in order to achieve coordination with other agents.

Cont…

- Thirdly, agents may be classified along several ideal and primary attributes which agents should exhibit.
- we have identified a minimal list of three: autonomy, learning and cooperation. We appreciate that any such list is contentious, but it is no more or no less so than any other proposal. Hence, we are not claiming that this is a necessary or sufficient set. Autonomy refers to the principle that agents can operate on their own without the need for human guidance, even though this would sometimes be invaluable. Hence agents have individual internal states and goals, and they act in such a manner as to meet its goals on behalf of its user. A key element of their autonomy is their proactiveness, i.e. their ability to 'take the initiative' rather than acting simply in response to their environment (Wooldridge & Jennings, 1995a). Cooperation with other agents is paramount: it is the raison d'être for having multiple agents in the first place in contrast to having just one. In order to cooperate, agents need to possess a social ability, i.e. the ability to interact with other agents and possibly humans via some communication language (Wooldridge & Jennings, 1995a)

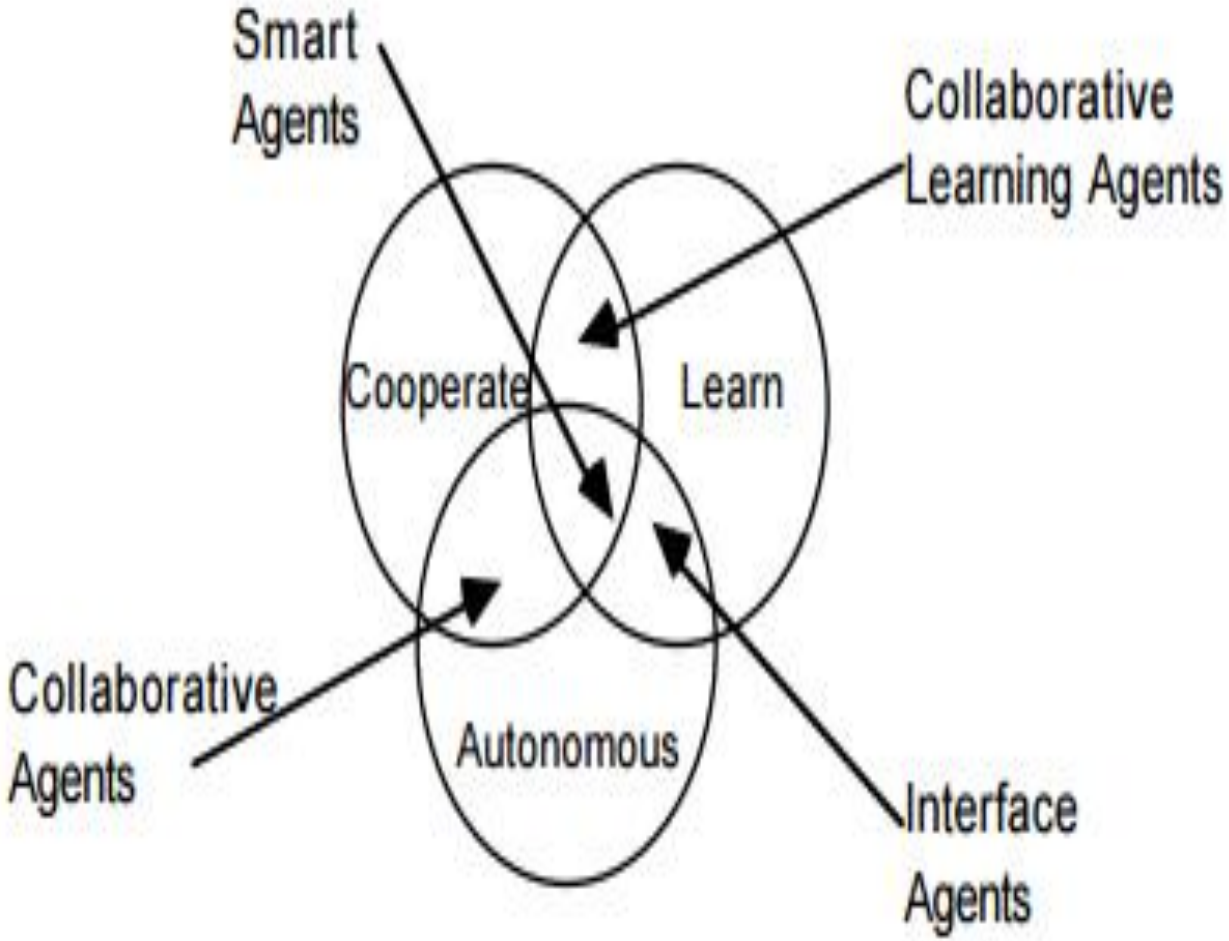


Figure 1 - A Part View of an Agent Typology

Cont….

- Fourthly, agents may sometimes be classified by their roles (preferably, if the roles are major ones), e.g. world wide web (WWW) information agents. This category of agents usually exploits internet search engines such as WebCrawlers, Lycos and Spiders. Essentially, they help manage the vast amount of information in wide area networks like the internet.

Fifthly, we have also included the category of hybrid agents which combine of two or more agent philosophies in a single agent.

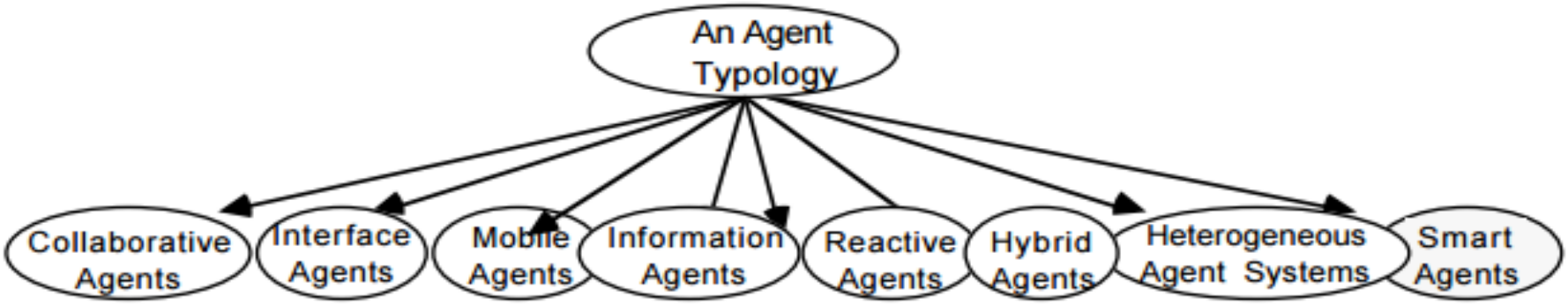


Figure 2 - A Classification of Software Agents

Collaborative Agents:

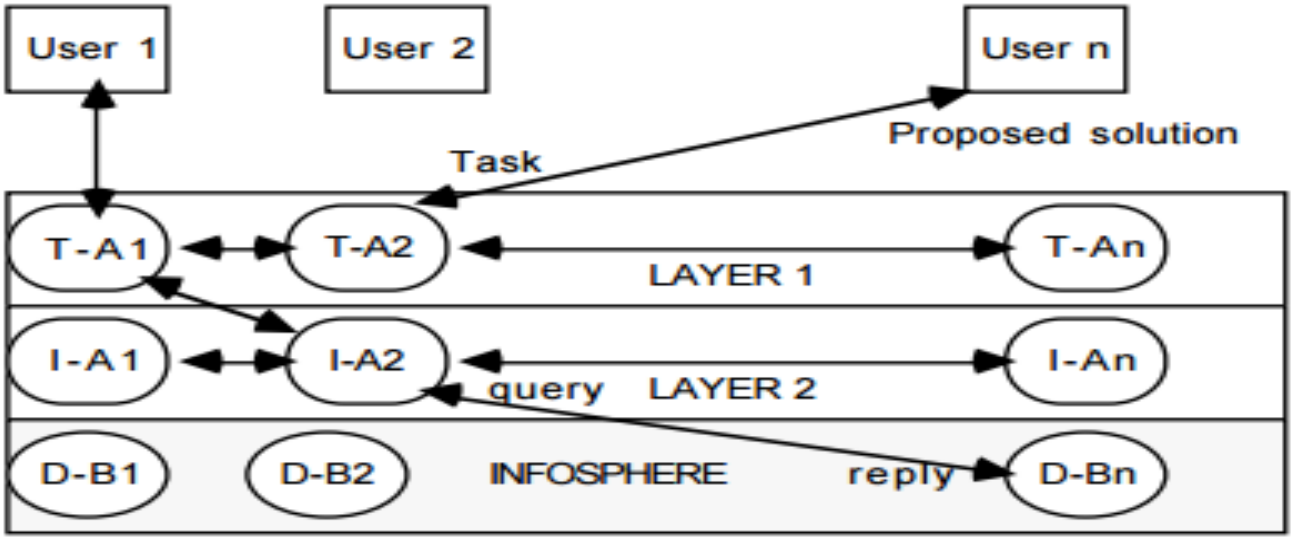
- To solve problems that are too large for a centralised single agent to do due to resource limitations or the sheer risk of having one centralised system;
- To allow for the interconnecting and interoperation of multiple existing legacy systems, e.g. expert systems, decision support systems, etc.;
- To provide solutions to inherently distributed problems, e.g. distributed sensor networks (cf. DVMT, Durfee et al., 1987) or air-traffic control

Cont…

- • To provide solutions which draw from distributed information sources, e.g. for distributed on-line information sources, it is natural to adopt a distributed and collaborative agent approach;
- • To provide solutions where the expertise is distributed, e.g. in health care provisioning;
- • To enhance modularity (which reduces complexity), speed (due to parallelism), reliability (due to redundancy), flexibility (i.e. new tasks are composed more easily from the more modular organisation) and reusability at the knowledge level (hence shareability of resources);
- • To research into other issues, e.g. understanding interactions among human societies.

A Prototypical Example: CMU's Pleiades System

- Pleiades is a distributed collaborative agent-based architecture which has two layers of abstraction: the first layer which contains task-specific collaborative agents and second layer which contains information-specific collaborative agents.



The Pleiades Distributed System Architecture (Adapted from Sycara, 1995)

Collaborative Agents: Some Key Challenges

- Engineering the construction of collaborative agent systems
- Inter-agent coordination
- Stability, Scalability and Performance Issues
- Legacy systems
- How do these systems learn?
- Evaluation of collaborative agent systems

Interface Agents

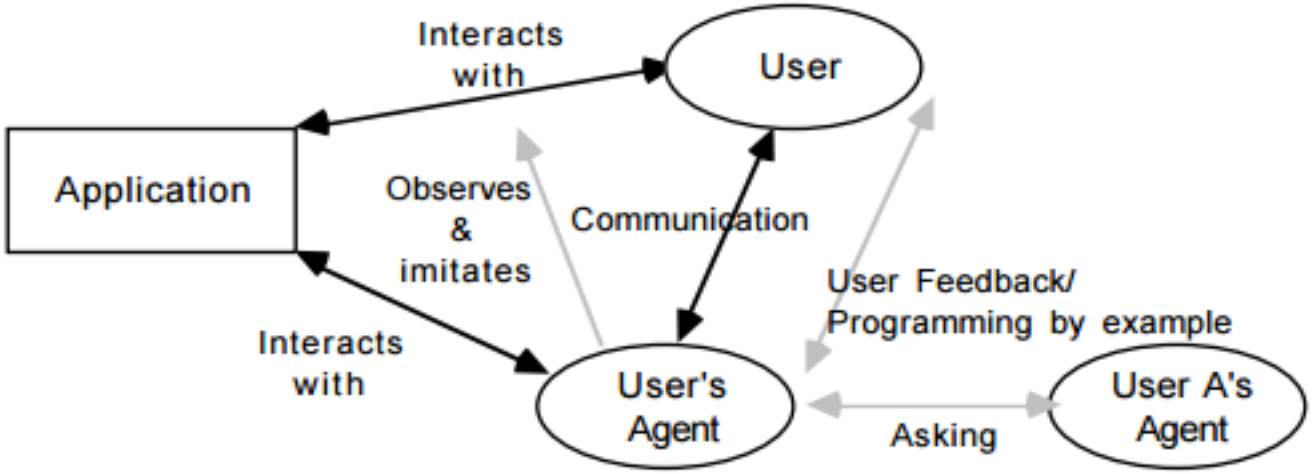


Figure 4 - How Interface Agents Work (adapted from Maes, 1994)

As for learning, interface agents learn typically to better assist its user in four ways (Maes, 1994) all shown in Figure 4:

- By observing and imitating the user (i.e. learning from the user);
- Through receiving positive and negative feedback from the user (learning from the user);
- By receiving explicit instructions from the user (learning from the user);
- By asking other agents for advice (i.e. learning from peers).

Interface Agents: Some Challenges

- Demonstrating that the knowledge learned with interface agents can truly be used to reduce users' workload, and that users, indeed, want them;
- Carrying out hundreds of experiments using various machine learning techniques (including soft and evolutionary learning techniques) over several domains to determine which learning techniques are preferable for what domains and why;
- Analysing the effect of the various learning mechanisms on the responsiveness of agents;
- Extending interface agents to be able to negotiate with other peer agents;
- Enhance continually the competence of interface agents so that their users' trust in them build up over time (Maes, 1994). Other issues which Maes notes include guaranteeing the users' privacy and the legal quagmire which may ensue following the fielding of such agents.
- Extending the range of applications of interface agents into other innovative areas such as entertainment, as ALIVE and HOMR are doing.

benefits of interface agents

The general benefits of interface agents are threefold.

- First, they make less work for the end user and application developer.
- Secondly, the agent can adapt, over time, to its user's preferences and habits.
- Finally, know-how among the different users in the community may be shared (e.g. when agents learn from their peers).

Mobile Agents

Benefits

- Reduced communication costs
- Limited local resources
- Easier coordination
- Asynchronous computing
- A flexible distributed computing architecture
- mobile agents represent an opportunity for a radical and attractive rethinking of the design process in general.

Mobile Agents: Some Challenges

- **Transportation: how does an agent move from place to place? How does it pack up and move?**
- **• Authentication: how do you ensure the agent is who it says it is, and that it is representing who it claims to be representing?**
- **How do you know it has navigated various networks without being infected by a virus?**
- **• Secrecy: how do you ensure that your agents maintain your privacy? How do ensure someone else does not read your personal agent and execute it for his own gains? How do ensure your agent is not killed and its contents 'core-dumped'?**
- **• Security: how do you protect against viruses? How do you prevent an incoming agent from entering an endless loop and consuming all the CPU cycles?**
- **• Cash: how will the agent pay for services? How do you ensure that it does not run amok and run up an outrageous bill on your behalf?**

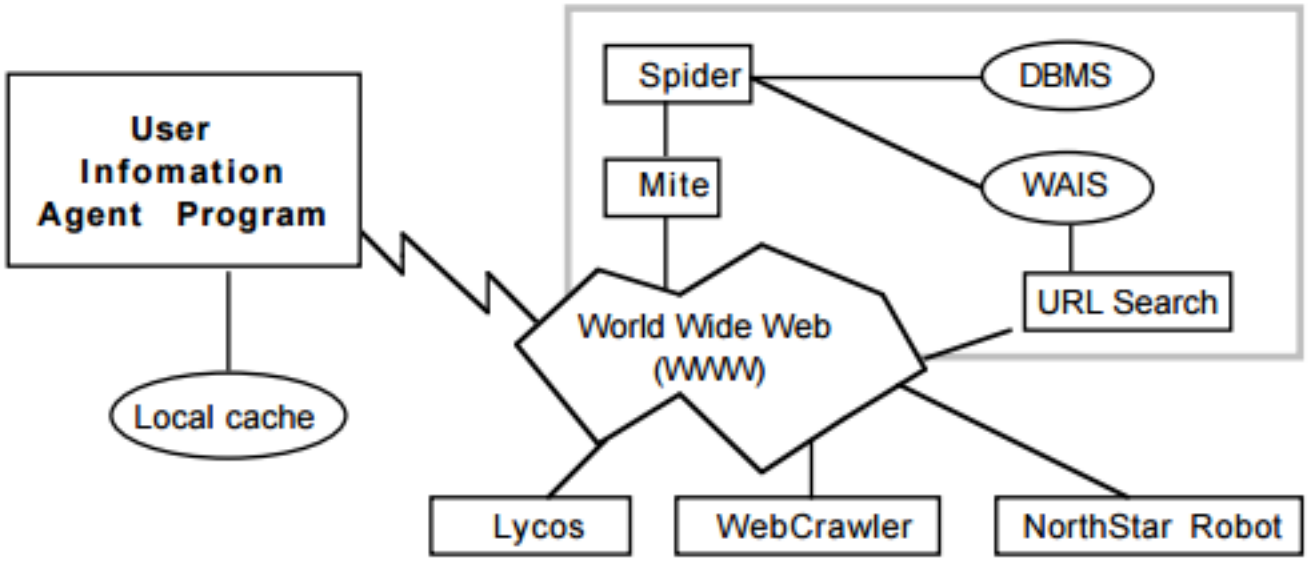
Mobile Agents: Some Challenges

In addition to these are the following:

- **• Performance issues: what would be the effect of having hundreds, thousands or millions of such agents on a WAN?**
- **• Interoperability/communication/brokering services: how do you provide brokering/directory type services for locating engines and/or specific services?**

Information/Internet Agents

- How Information Agents Work



A view of how Information Agents Work (Adapted from Indermaur, 1995)

Reactive Software Agents

- Reactive agents represent a special category of agents which do not possess internal, symbolic models of their environments; instead they act/respond in a stimulus-response manner to the present state of the environment in which they are embedded. Reactive agents work dates back to research such as Brooks (1986) and Agre & Chapman (1987), but many theories, architectures and languages for these sorts of agents have been developed since. However, a most important point of note with reactive agents are not these (i.e. languages, theories or architectures), but the fact that the agents are relatively simple and they interact with other agents in basic ways.

Reactive Software Agents

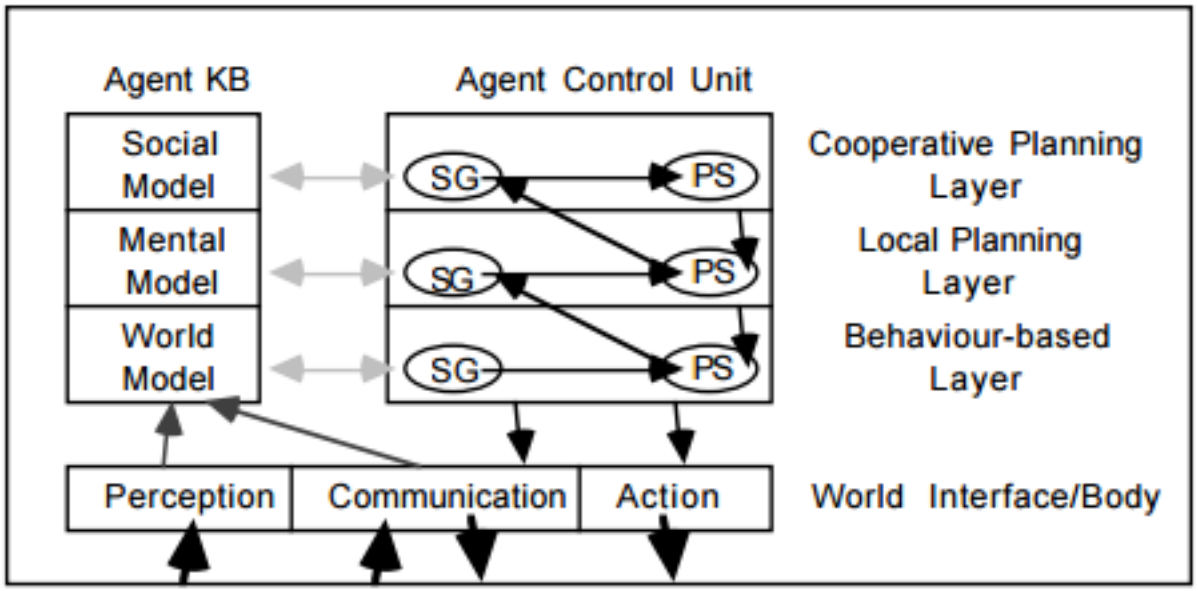
- Some Challenges

- Expanding the range and number of applications based on reactive agents
- Methodology: there is a yearning need for a clearer methodology to facilitate the development of reactive software agent applications. This may or may not require the development of more associated theories, architectures and languages. Much of the current approaches, sadly, smacks of 'trial and error'
- Non-functional issues: issues such as scalability and performance would need to be addressed, though these are unlikely to be important until clearer methodologies have been developed and evaluated.

Hybrid Agents

- A prototypical example of a hybrid example is Muller et al.'s layered InteRRaP architecture shown in Figure 8 developed at the German Research Centre for AI. It is an architecture that implements a layered approach to agent design.
- . There are three control layers in this architecture: the behaviour-based layer (BBL), the local planning layer (LPL) and the cooperative planning layer (CPL).

Cont..



The InteRRaP Hybrid Architecture (from Fischer *et al.*, 1995)

Heterogeneous Agent Systems

- Heterogeneous agent systems, unlike hybrid systems described in the preceding section, refers to an integrated set-up of at least two or more agents which belong to two or more different agent classes. A heterogeneous agent system may also contain one or more hybrid agents.

The potential benefits for having heterogeneous agent technology

- • Standalone applications can be made to provide 'value-added' services by enhancing them in order to participate and interoperate in cooperative heterogeneous set-ups
- • The legacy software problem may be ameliorated because it could obviate the need for costly software rewrites as they be given 'new leases of life' by getting them to interoperate with other systems.
- At the very least, heterogeneous agent technology may cushion or lessen the blow or effect of routine software maintenance, upgrade or rewrites
- • Agent-based software engineering provides a radical new approach to software design, implementation and maintenance in general, and software interoperability in particular. Its ramifications (e.g. moving from passive modules in traditional software engineering to proactive agent-controlled ones) would only be clear as this methodology and its tools become clearer.

Size and Intelligence

- - Agents can be of various sizes and can possess various amounts of intelligence. Generally, intelligence of a software agent is proportional to its size, so we can distinguish: *big-sized*, *middle-sized* and *micro agents*. It is difficult to make clear boundaries among these categories.

  1. A big-sized agent occupies and controls one or more computers. It possesses enough competence to be useful even if it acts alone,

Without the other agents in a MAS. A big-sized agent can be as big and as intelligent as an expert system ,with competences for expert problem solving, e.g. distributed medical care or plane ticket reservation.

2. A middle-sized agent is the one that is not useful without the other agents

in a MAS or without additional software However, it is able to perform some non-trivial task(s). A user-interface agent that acts without other agents and performs some simple actions can also be classified as a middle-sized agent. Mobile agents are usually middle-sized agents.

- 3. Micro agents (also called the Society of Mind agents), do not possess any intelligence. Minsky followed the idea that the intelligence emerges as a global effect of the overall activity of many simple and unintelligent agents.

- .

Adaptation – Adaptive agents can adapt their behavior to different situations and changes in the environment. For example, a navigation system can adapt to changes in traffic (e.g. a traffic jam) and propose alternative routes. This makes adaptive agents more robust to non-predicted changes in a dynamic environment.

Learning - Agents can use learning capabilities for better performance.

Learning can be done online, e.g. by data mining from data which are constantly collected through interaction with users (e.g. for profiles). Offline learning refers to training processes (e.g. for pattern recognition) prior to productive agent usage.

Adaptivity	*Agents can adapt to unpredicted changes.*
Autonomy	An agent can act without direct intervention by humans or other agents and that it has control over its own actions and internal state
Benevolence	It is the assumption that agents do not have conflicting goals and that every agent will therefore always try to do what is asked of it.
Character (personality)	An agent has a well-defined, believable "personality" and emotional state.
Competitiveness	An agent is able to coordinate with other agents except that the success of one agent may imply the failure of others.
Cooperation or collaboration	An agent is able to coordinate with other agents to achieve a common purpose; non-antagonistic agents that succeed or fail together.
Coordination	An agent is able to perform some activity in a shared environment with other agents. Activities are often coordinated via plans, workflows, or some other process management mechanism.
Credibility	An agent has a believable personality and emotional state.
Deliberation	A deliberative agent decides for its actions by reasoning processes which may involve mental categories like goals, plans etc.
Embodiment	An embodied agent can interact with its environment by physical processes. This allows for emergent controls guided by sensor data without internal representations.
Emergent behavior	More complex behavior emerges by interaction of (simple) agents with each other (swarm intelligence) or with the environment (embodied agents, situated agents).
Flexibility	The system is responsive (the agents should perceive their environment and respond in a timely fashion to changes that occur in it), pro-active and social.
Goal directed	Agent behavior is guided by mental qualities like goals, which are results of deliberation. Then the agent tries to achieve the goal by appropriate actions.

Hybrid architecture	Combination of different architectures. Often with simple (reactive, stimulus response) control for low level behavior and deliberative control for high level behavior.
Inferential capability	An agent can act on abstract task specification using prior knowledge of general goals and preferred methods to achieve flexibility; goes beyond the information given, and may have explicit models of self, user, situation, and/or other agents.
Intelligence	An agent's state is formalized by knowledge and the agent interacts with other agents using symbolic language.
Interpretation ability	An agent is interpretive if it can correctly interpret its sensor readings.
"Knowledge-level" communication ability	The ability to communicate with persons and other agents with language more resembling human-like "speech acts" than typical symbol-level program-to-program protocols.
Learning	An agent is capable of learning from its own experience, its environment, and interactions with others.
Mobility	an agent is able to transport itself from one machine to another and across different system architectures and platforms.
Prediction ability	An agent is predictive if its model of how the world works is sufficiently accurate to allow it to correctly predict how it can achieve the task.
Proxy ability	An agent can act on behalf of someone or something acting in the interest of, as a representative of, or for the benefit of, some entity.
Personality (character)	An agent has a well-defined, believable "personality" and emotional state.
Proactiveness	An agent does not simply act in response to its environment; it is able to exhibit goal-directed behavior by taking the initiative.
Rationality	It is the assumption that an agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved — at least insofar as its beliefs permit.
Reactivity	An agent receives some form of (sensory) input from its environment, and it performs some action that changes its environment in some way.

- **Multi-Agent Systems and Agent Communication**
  "*Distributed Problem Solving*" is performed by agents working together towards a solution of a common problem (e.g. for expert systems).
  MultiAgent Systems (MAS) take a more general view of agents which have contact with each other in an environment (e.g. the Internet) . The rules of the environment as well as the agent controls determine the form of *coordination*.
  The agents may be *cooperative* or *competitive*.
  Relations between local and global behavior in such MAS have been studied using game theory and social theories.

- *Communication* via exchange of messages is the usual prerequisite for coordination. Nevertheless, cooperation is possible even without communication, by observing the environment. The two most important approaches to communication are using protocols and using an evolving language. Both have their advantages and disadvantages. For industrial applications, communication protocols are the best practice, but in systems where homogeneous agents can work together, language evolution is the more acceptable option *Agent Communication Languages* (ACLs)provide important features like technical declarations (sender, receiver, broadcasting, peer-to-peer, …), speech act (query, inform, request,acknowledge,…) and content language (e.g. predicate logic).

Languages for coordination and communication between agents.

Agent communication language	***Description***
KQML ("Knowledge Query and Manipulation Language")	It is perhaps the most widely used agent communication language [102], [45]. KQML uses speech-act performatives such as reply, tell, deny, untell, etc. Every KQML message consists of a performative and additional data written in several slots. Some slots are :content, :in-reply-to, :sender, :receiver, :ontology, etc. The set of performatives in KQML and their slots should be general enough to enable agent communication in every agent application. There are claims that there might be some problems with the semantics of performatives. Various agents may interpret the same performative in various ways.
FIPA-ACL ("FIPA Agent Communication Language")	It is an agent communication language that is largely influenced by ARCOL [102]. FIPA ACL has been defined by FIPA - Foundation for Intelligent Physical Agents. Together FIPA-ACL [47], ARCOL, and KQML establish a quasi standard for agent communication languages [102]. Syntax and semantics of FIPA ACL are very similar to the syntax and semantics of KQML. Time will show which one of these two standards will prevail.
ARCOL ("ARTIMIS COmmunication Language")	ARCOL has a smaller set of communication primitives than KQML, but these can be composed. This communication language is used in the ARTIMIS system [102].

KIF ("Knowledge Interchange Format")	This logic-based comprehensive language with declarative semantics has been designed to express different kinds of knowledge and meta-knowledge [102]. KIF is a language for content communication, whereas languages like KQML, ARCOL, and FIPA-ACL are for intention communication.
COOL ("Domain independent COOrdination Language")	COOL relies on speech-act based communication, aims at explicitly representing and applying coordination knowledge for multi-agent systems and focuses on rule-based conversation management (conversation rules, error rules, continuation rules, ...) [102]. Languages like COOL can be considered as supporting a coordination/communication (or "protocol-sensitive") layer above intention communication.

MAS with many agents are often used for simulations to study *Swarm Intelligence* and for social simulations in the field of *Socionics*. Social simulations include simulations of financial markets, traffic scenarios, and social relationships. Swarm intelligence can lead to complex "intelligent" behavior which emerges from the interaction of very simple agents, e.g. in ant colonies or in trade simulations. Complex problems, e.g. the well-known travelling salesman problem can be solved with swarm techniques.

Languages for constructing Agent-based systems

According to the usage of mental attitudes.

We can classification the languages to :

- agent-oriented programming (AOP) languages;
- belief-desire-intention (BDI) languages,
- hybrid languages (that combine AOP and BDI within a single model),
- and other (prevalently declarative) languages.

Agent-oriented programming model

- **The term *Agent-oriented Programming* (AOP) was coined in to define a novel programming paradigm. It represents a computational framework whose central compositional notion is an agent, viewed as a software component with *mental qualities*, *communicative skills* and a *notion of time*. AOP is considered to be a specialization of object-oriented programming (OOP), but there are some important differences between these concepts.**
  Objects and agents differ in their *degree of autonomy*. Unlike objects, which directly *invoke* actions of other objects, agents *express their desire* for an action to be executed. In other words, in OOP the decision lies within the requesting entity, while in AOP the receiving entity has the control over its own behavior, by deciding whether an action is executed or not. Also, agents can often have *conflicting* interests, so it might be harmful for an agent to execute an action request from another agent. An additional difference is *flexibility*. Agents often exhibit pro-active and adaptive behavior and use learning to improve their performance over time. *The thread of control* is the final major difference. While multi-agent systems are multi-threaded by default, there is usually a single thread of control in OOP.

The AOP paradigm was very influential for the further development of agent programming languages, resulting in a number of languages.

- **AGENT0**

  was the first agent-oriented programming language that has been developed, providing a direct implementation of the agent-oriented paradigm, In AGENT0, an agent definition consists of four parts: a set of capabilities(describing what the agent can do), a set of beliefs, a set of commitments or intentions, and a set of commitment rules containing a message condition, a mental condition and an action.

- **PLACA**

  Planning Communicating Agents - PLACA is an improvement of the AGENT0 language, extending it with planning facilities, which significantly reduce the intensity of communication In PLACA, an agent doesn't need to send a separate message each time it requests another agent to perform some action. Instead, it can provide another agent with a description of the desired final state.

Agent-K

Agent-K is another extension of the AGENT0 . It replaces custom communication messages (i.e. *request*, *unrequest* and *inform*) with the standardized KQML., This improves the general interoperability of agents and enables them to communicate with different types of agents.

Agent-K uses the KAPI1 library for agent communication, which can transport KQML messages over TCP/IP and e-mail to remote systems.

MetateM

is probably one of the oldest programming languages for multi-agent systems. It was based on the direct execution of logical formulae. A MetateM agent program consists of a set of temporal rules that are built according to the paradigm: *declarative past and imperative future*

April and MAIL

Agent PRocess Interaction Language - April is a process oriented symbolic language that was not designed specifically for agent oriented programming, but rather as a general-purpose multi-process development tool.

April has a simple communication infrastructure that uses TCP/IP and permits access to non-April based applications. Agents communicate by exchanging messages identified by their handles.

BDI based languages

- A significant and influential trend in designing agent programming languages stemmed from the success of the practical reasoning agent architectures, among which the most notable is probably the PRS – Procedural Reasoning System . PRS became the first system embodying a belief, desire, and intention (BDI) architecture. Based on that, approximately around the same time with Shoham, Rao proposed the AgentSpeak(L) language .
  AgentSpeak(L) employs the metaphors of belief, desire, and intention of the BDI architecture to shape the design of an innovative agent programming language. However, AgentSpeak(L) was only a proposal, while Jason programming language became in 2004 the first implementation of an interpreter for an extended version of AgentSpeak(L).
  AgentSpeak(L) is often described as a BDI agent programming language, as it is assumed to convey the most important ideas of BDI agent architectures (including the PRS).
  In this section we will present several important agent programming languages which support BDI architecture and belong to the hybrid paradigm.

Agent Speak

The primary goal of the authors of Agent Speak was to join BDI architectures for agents and for object-based concurrent programming and to develop a language that would capture the essential features of both. They identified the primary characteristics of agents: complex internal mental state, proactive or goal-directed behavior, communication through structured messages or speech acts, distribution over a wide-area network, adequate reaction to changes in the environment, concurrent execution of plans and reflective or meta-level reasoning capabilities.

In AgentSpeak there are three distinct types of services for different purposes:

- Achieve-service: used to achieve a certain state of the world
- Query-service: used to check whether something is true, considering the associated database
- Told-service: used to share some information with another agent.

Agents communicate in AgentSpeak by exchanging messages, either asynchronously (default) or synchronously. A message can be sent to a specific agent or to an agent family, in which case it is forwarded to all instances of that family. If a message sent to another agent contains some information, but puts no obligation upon the receiving agent, it is called an inform speech-act. Otherwise, it's a request. In addition, a message can have a priority assigned to it, thus giving it an overall importance.

Jason

is probably the first implementation of AgentSpeak(L) using the Java programming language and belongs to the hybrid agent paradigm . The syntax of Jason exhibits some similarities with Prolog.
it is tightly integrated with Java with the following immediate consequences:

- (i) the behavior of the Jason interpreter can be tailored using Java
- (ii) Jason can be used to build situated agents by providing a Java API for integration with an environment model that is developed with Java
- (iii) Jason has been integrated with some existing agent frameworks, including JADE ,AgentScape ,and Agent Factory.

JACK Agent Language

- JACK Intelligent Agents, or simply JACK, is a commercial agent platform provided by Autonomous Decision Making Software – AOS. The main JACK components are: JACK Agent Language (also known as JAL), JACK compiler, JACK kernel, and JACK Development Environment.
  JAL is a superset of Java that incorporates the full Java language and provides the necessary constructs for building agent-oriented programs according to the BDI model

JADEX

JADEX is a Java-based agent platform that tries to respond to three categories of requirements: openness, middleware, and reasoning, thus bridging the gap between middleware-centered and reasoning-centered systems .

The architecture of a JADEX agent follows the Procedural Reasoning System computational model of practical reasoning.

Agents in JADEX communicate by exchanging messages. Internally, an agent reacts to events in its execution cycle that combines reaction and deliberation processes.

A JADEX agent uses the concepts of BDI agents: beliefs, desires (goals in JADEX), and intentions (plans in JADEX). JADEX employs an object-oriented representation of beliefs. Additionally, beliefs have an active role, i.e. their update can trigger generation of events or adoption/dismissing of goals.

JADEX uses four types of goals:

- (i) perform goal designating action execution;
  (ii) achieve goal designating a point-wise condition in the lifecycle of an agent that must be reached;
- (iii) query goal that is an introspection mechanism by which an agent is inspecting its own internal state;
- (iv) maintain goal designating a process-wise condition that must be maintained during the agent's execution. JADEX plans represent the behavioral aspect of an agent and they have a procedural flavor. A plan consists of a head and a body, similarly to a procedure in a procedural language.

Tools and Platforms

- **more than 100 agent infrastructures have been developed in the previous two decades. For portability and usability reasons most of them are built on top of and are integrated with Java. Unfortunately, only few of them are still currently available, others either becoming obsolete or not being developed anymore.**
  Futhermore, this prominent technology inspired some authors to go a step further. In authors extrapolated future trends in multi-agent systems and presented a thorough and outstanding approach to the future of multi-agent systems. Finally, it is important to mention that in order to be accepted by the industrial community, MAS applications need to be successfully demonstrated in complex real world pilot systems .

ZEUS

ZEUS developed by British Telecommunications Labs, is a collaborative agent building environment that has excellent GUI and debugging, provides library of predefined coordination strategies, general purpose planning and scheduling mechanism, self-executing behavior scripts,etc.

JADE

Java Agent DEvelopment Framework - JADE is probably one of the most popular agent platforms that are currently available to the open source community.

Agent Tool

Agent Tool is a Java-based graphical development environment/tool that supports the Multi-agent Systems Engineering (MaSE) methodology ,originally developed at the Artificial Intelligence Lab of the Air Force Institute of Technology,

RETSINA

JATLite

MADKIT

JAFMAS

Examples of previous topics

- The problem-solving agent perfoms precisely by defining problems and its several solutions.
- According to psychology, "*a problem-solving refers to a state where we wish to reach to a definite goal from a present state or condition.*"
- According to computer science, *a problem-solving is a part of artificial intelligence which encompasses a number of techniques such as algorithms, heuristics to solve a problem.*
- Therefore, a problem-solving agent is a **goal-driven agent** and focuses on satisfying the goal.

Steps performed by Problem-solving agent

- **Goal Formulation:** It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent's performance measure (discussed below).
- **Problem Formulation:** It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal. There are following five components involved in problem formulation:
- **Initial State:** It is the starting state or initial step of the agent towards its goal.
- **Actions:** It is the description of the possible actions available to the agent.

- **Transition Model:** It describes what each action does.
- **Goal Test:** It determines if the given state is a goal state.
- **Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure. Remember, **an optimal solution has the lowest path cost among all the solutions.**

Note:

Initial state, actions, and **transition model** together define the **state-space** of the problem implicitly. State-space of a problem is a set of all states which can be reached from the initial state followed by any sequence of actions. The state-space forms a directed map or graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.

- **Search:** It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an input and returns solution as its output.
- **Solution:** It finds the best algorithm out of various algorithms, which may be proven as the best optimal solution.
- **Execution:** It executes the best optimal solution from the searching algorithms to reach the goal state from the current state.

Problem Types

Deterministic, fully observable → single-state problem

- Agent knows exactly what state it will be in; solution is a sequence

Non-observable → conformant problem

- Agent may have no idea where it is; solution (if any) is a sequence

Non-deterministic and/or partially observable

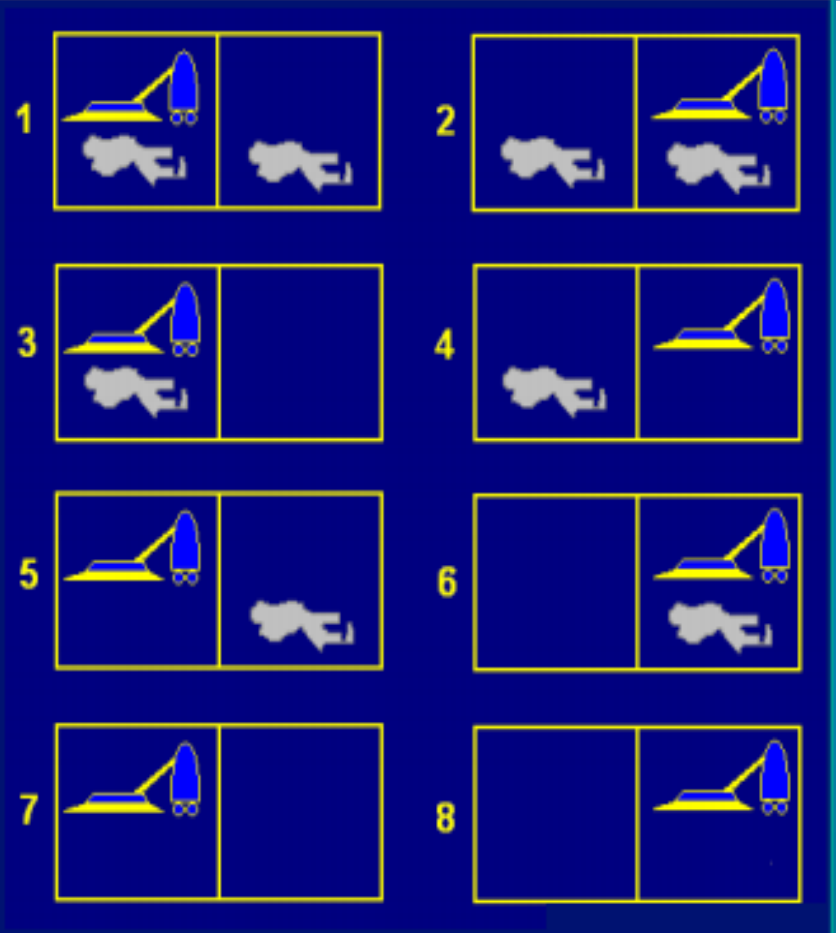
- Percepts provide *new* information about current state
- Solution is a *tree* or *policy*
- Often *interleave* search, execution

Unknown state space → exploration problem ("online")

Example: Vacuum World

Single-state, start in #5.

- Solution??



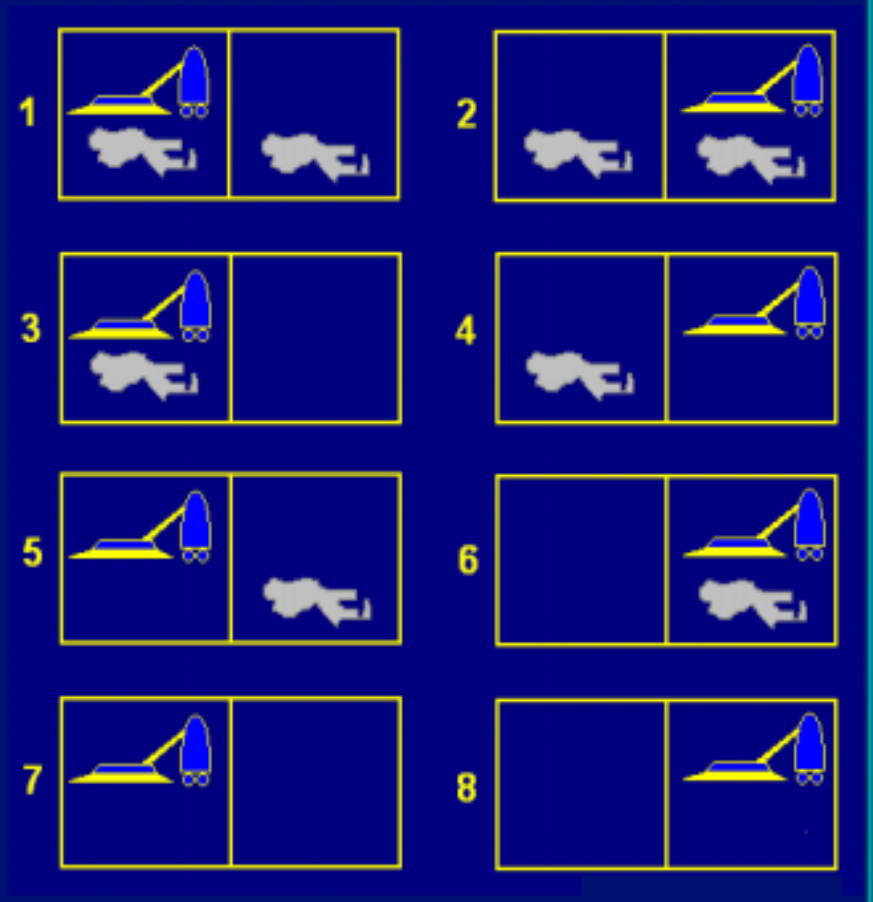
Example: Vacuum World

Single-state, start in #5.

- Solution: [Right, Suck]

Conformant, start in {1,2,3,4,5,6,7,8}

- E.g., right goes to {2,4,6,8}
- Solution??



Example: Vacuum World

Single-state, start in #5.

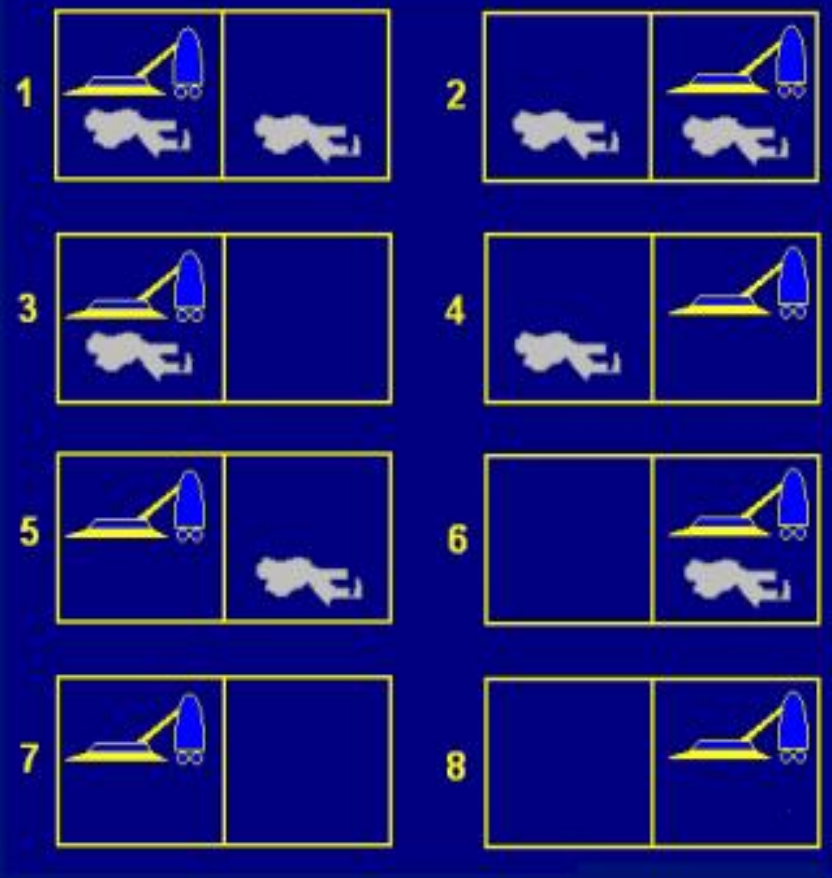
- Solution: *[Right, Suck]*

Conformant, start in {1,2,3,4,5,6,7,8}

- E.g., right goes to {2,4,6,8}
- Solution: *[Right, Suck, Left, Suck]*

Contingency, start in #5

- Murphy's Law: *Suck* can dirty a clean carpet
- Local sensing: dirt, location only
- Solution??



Example: Vacuum World

Single-state, start in #5.

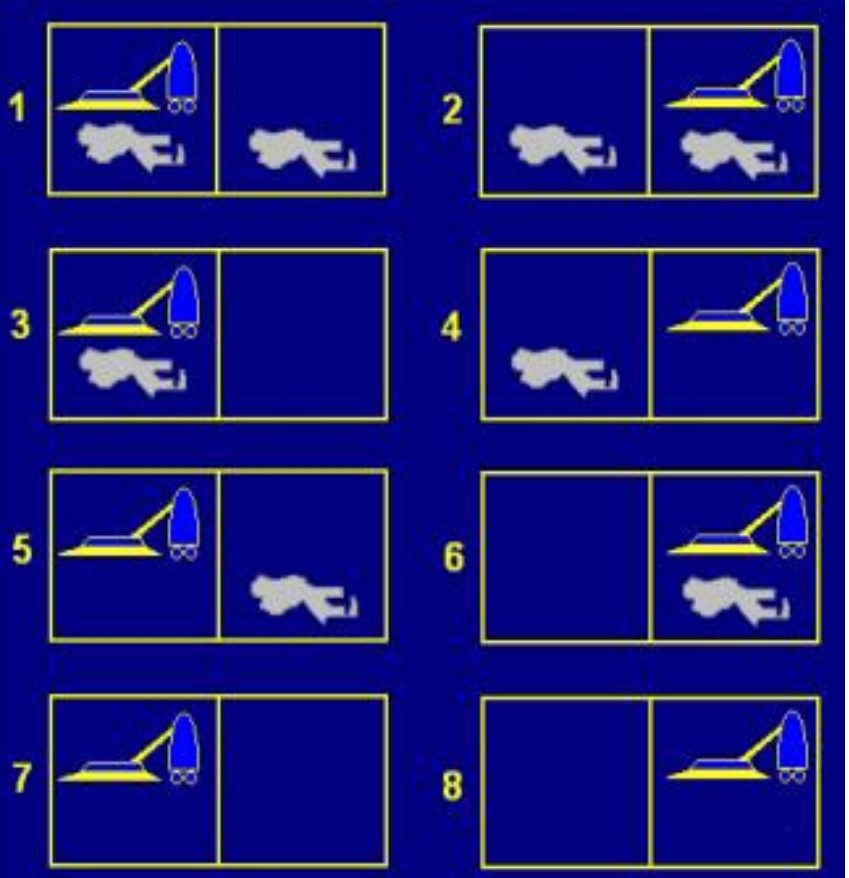
- Solution: [*Right, Suck*]

Conformant, start in {1,2,3,4,5,6,7,8}

- E.g., right goes to {2,4,6,8}
- Solution: [*Right, Suck, Left, Suck*]

Contingency, start in #5

- Murphy's Law: *Suck* can dirty a clean carpet
- Local sensing: dirt, location only
- Solution: [*Right,* if *dirt* then *Suck*]



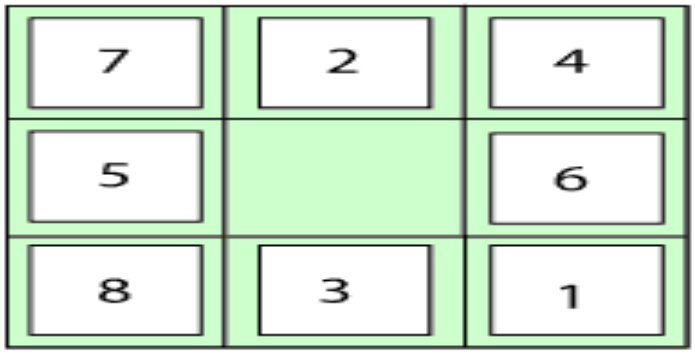
Example Problems

Basically, there are two types of problem approaches:

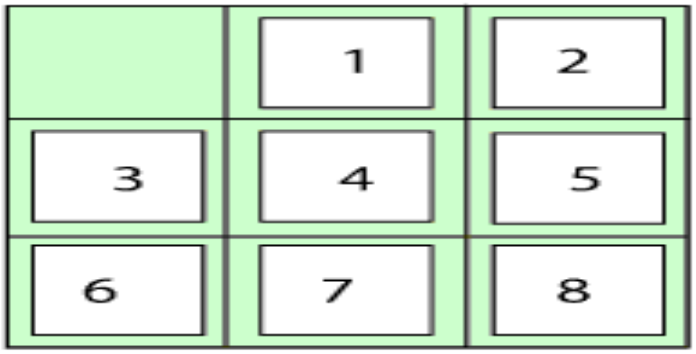
- **Toy Problem:** It is a concise and exact description of the problem which is used by the researchers to compare the performance of algorithms.
- **Real-world Problem:** It is real-world based problems which require solutions. Unlike a toy problem, it does not depend on descriptions, but we can have a general formulation of the problem.

Toy Problems

8 Puzzle Problem



Start State

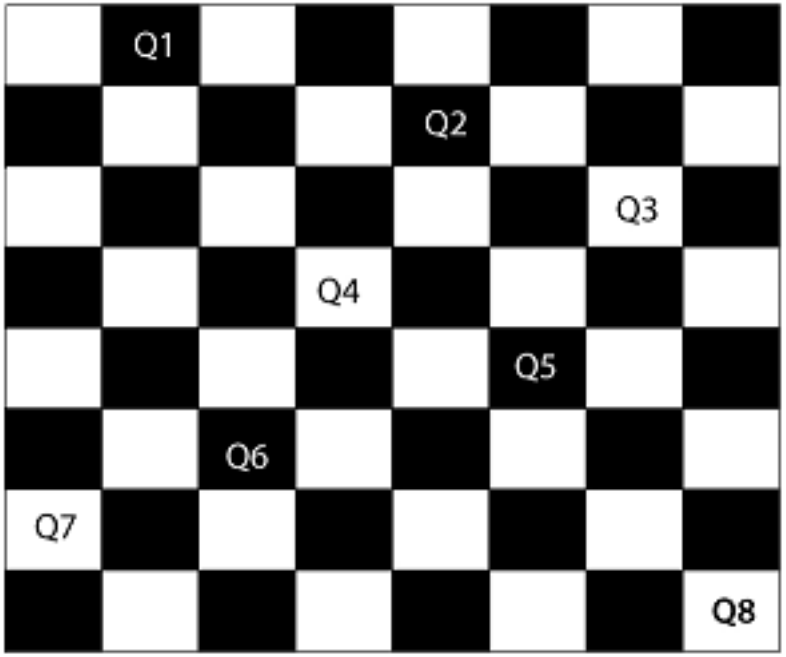


Goal State

States: .?
Initial State: ?
Actions: ?
Transition Model: ?
Goal test: ?
Path cost:?

Toy Problems

8-queens problem:



The aim of this problem is to place eight queens on a chessboard in an order where no queen may attack another. A queen can attack other queens either **diagonally or in same row and column.**

Toy Problems

- **For this problem, there are two main kinds of formulation:**
- **Incremental formulation:** It starts from an empty state where the operator augments a queen at each step.
- **Complete-state formulation:** It starts with all the 8-queens on the chessboard and moves them around, saving from the attacks.

Toy Problems

- **Following steps are involved in this formulation (Incremental formulation):**
- **States:** Arrangement of any 0 to 8 queens on the chessboard.
- **Initial State:** An empty chessboard
- **Actions:** Add a queen to any empty box.
- **Transition model:** Returns the chessboard with the queen added in a box.
- **Goal test:** Checks whether 8-queens are placed on the chessboard without any attack.
- **Path cost:** There is no need for path cost because only final states are counted.

Toy Problems

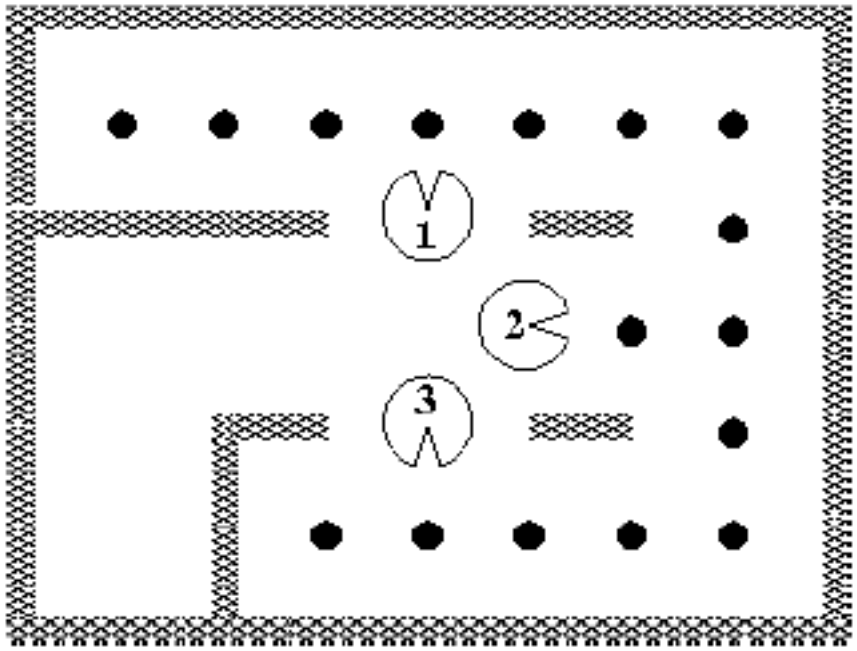
- **Complete-state formulation steps are**
- **States:** ?
- **Actions:** ?
- In This formulation the state space is ?

H.W

- Pacman is trying eat all the dots, but he now has the help of his family! There are initially k dots, at positions

(f1, . . . fk). There are also n Pac-People, at positions (p1, . . . , pn); initially, all the Pac-People start in the bottom left corner of the maze. Consider a search problem in which all Pac-People move simultaneously; that is, in each step each Pac-Person moves into some adjacent position (N, S, E, or W, no STOP). Note that any number of Pac-People may occupy the same position.

H.W cont….



Define the state space of the search problem??

What is the goal test?

What is the maximum branching factor of the successor function in a general grid?