



LECTURE 1: INTRODUCTION TO AGENTS AND MAS

Introduction to Multi-Agent Systems (MESIIA, MIA)

Universitat Rovira i Virgili

Outline of the lecture

1. Main trends in Computer Science
2. Agents and multi-agent systems
3. Objections to MAS
4. Agents vs. related technologies
5. Viewpoints on agent technology

Important aspects in Computer Science

- Five trends are marking the history of computing:
 - *ubiquity*
 - *interconnection*
 - *intelligence*
 - *delegation*
 - *human-orientation in programming methodologies*

Ubiquity

- The continuous reduction in size and cost of computing capability has made it possible to introduce processing power into places and devices that would have once been uneconomic / unimaginable (e.g. smartwatches)
- As processing capability spreads, sophistication and intelligence become ubiquitous



Ambient Intelligence

- Aimed at seamless delivery of services and applications, based on ubiquitous computing, ubiquitous communication and intelligent user interfaces.
- The vision
 - An environment of potentially **thousands of embedded** and **mobile devices** (or software components) interacting to support user-centered goals and activity.
 - **Autonomy, distribution, adaptation** and **responsiveness** are key characteristics of these independent and distributed components, and in this sense they share similar characteristics to intelligent **agents**.

Transportation
Intelligent Station



Home
Intelligent Living-room



Commerce
Intelligent Exhibition



Work
Intelligent Office



Leisure
Intelligent Playground



Education
Intelligent Classroom



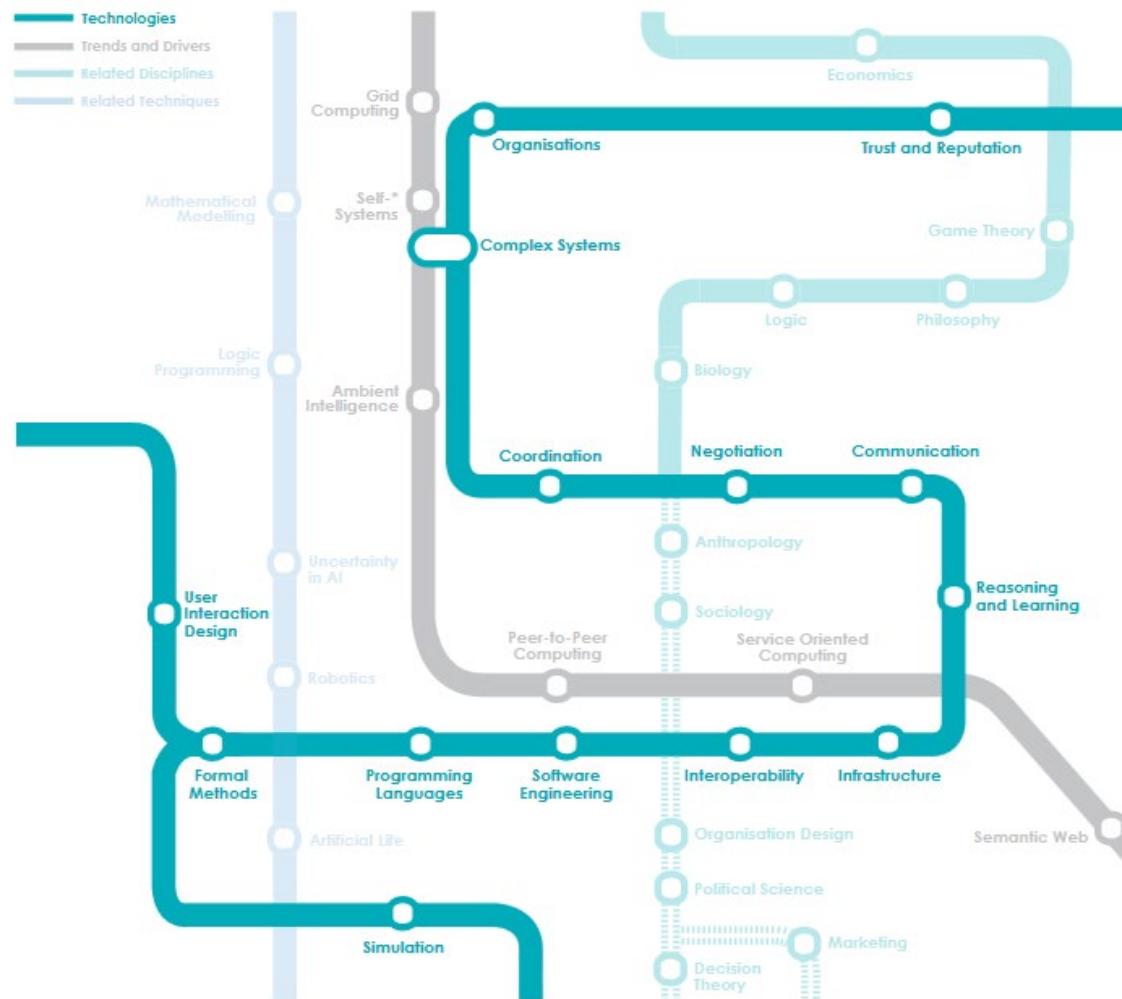
**Ambient
Intelligence
Environments**

Interconnection

- Computer systems today no longer stand alone, but are networked into large distributed systems
- Obvious example: Internet
- Since distributed and concurrent systems have become the norm, some researchers are putting forward theoretical models that portray *computing as primarily a process of interaction*

Agent Technology: Computing as Interaction

A Roadmap for Agent Based Computing



Compiled, written and edited by

Michael Luck, Peter McBurney, Onn Shehory, Steve Willmott and the AgentLink Community

Connectivity -Technological context

■ Semantic Web

- The Semantic Web is based on the idea that the data on the Web can be defined and linked in such a way that it can be used by machines for the automatic processing and integration of data across different applications (Berners-Lee et al., 2001).

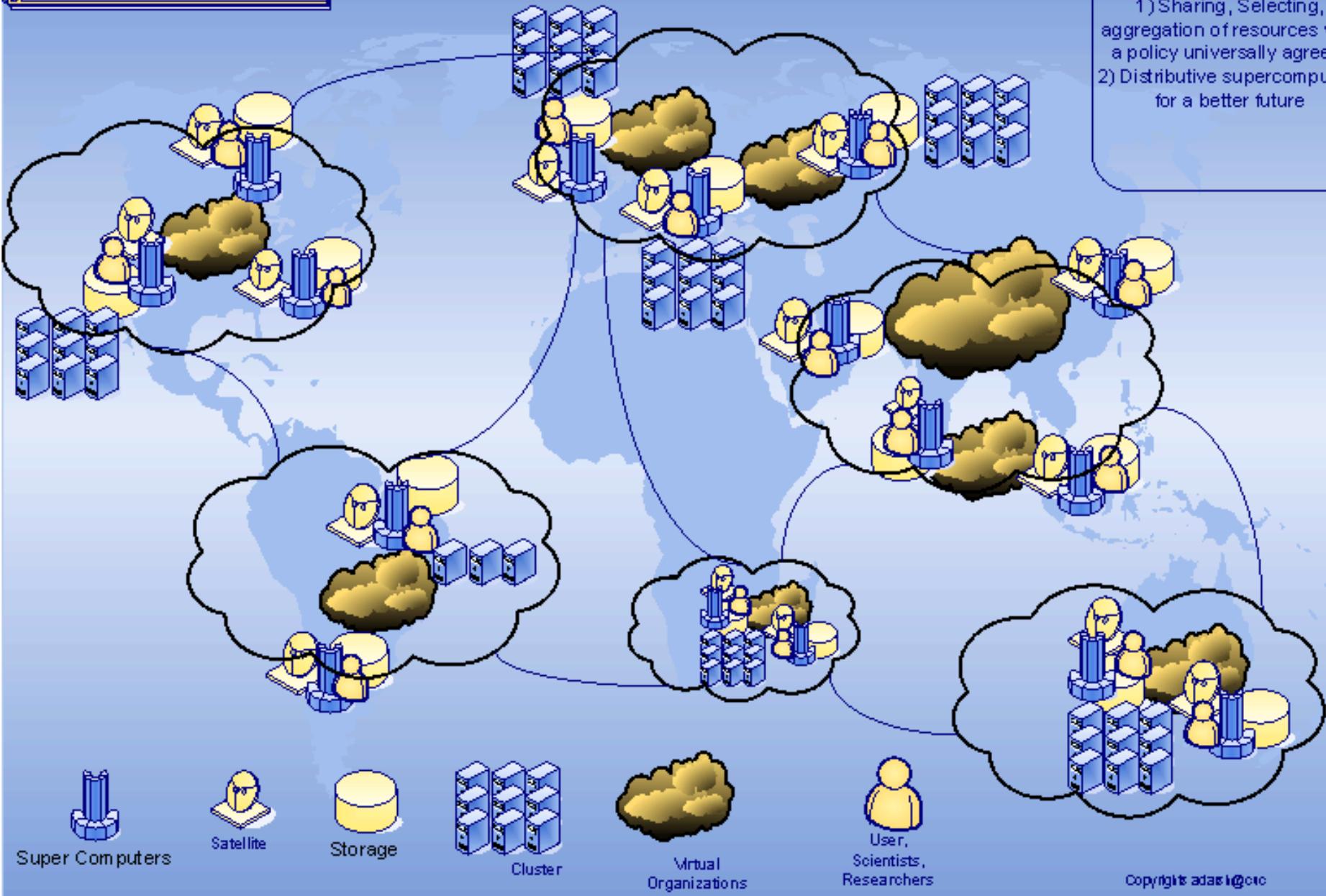
■ Peer-to-Peer computing

■ Grid computing

Grid Computing

What is all about Grid ?

- 1) Sharing, Selecting, aggregation of resources with a policy universally agreed.
- 2) Distributive supercomputing for a better future



Intelligence

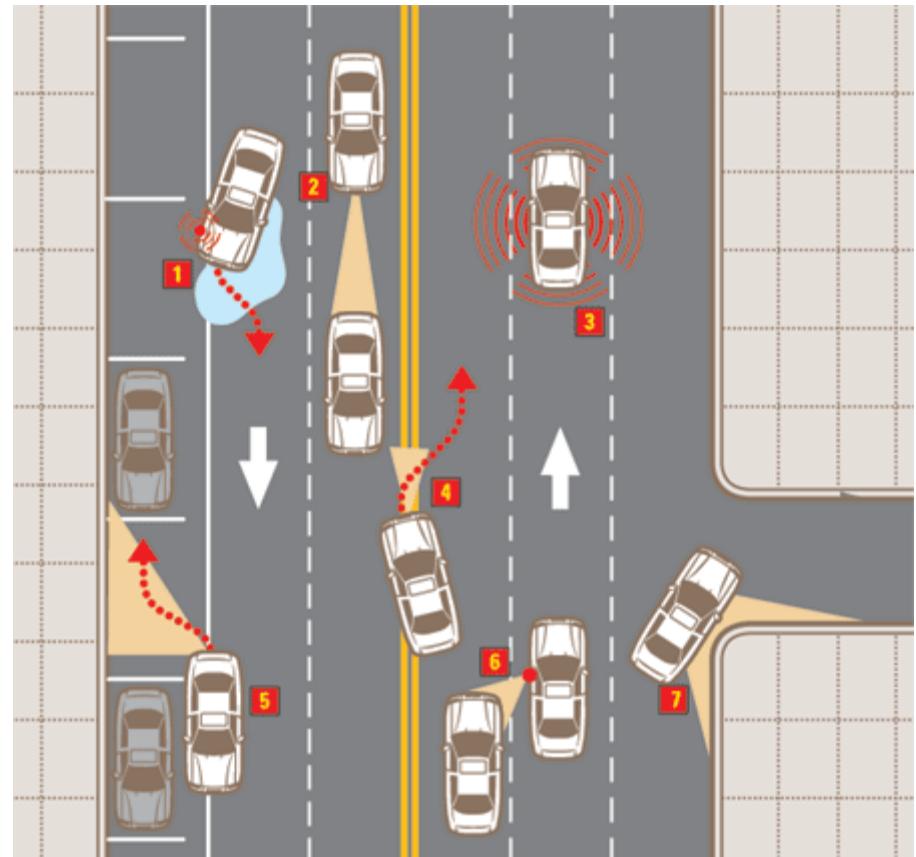
- The **complexity** of tasks that we are capable of automating and delegating to computers has grown steadily
- That requires an increasing amount of domain knowledge and inference capabilities, in short, more **intelligent** problem solving techniques

Delegation

- Computers are doing more for us – without our intervention
- We are *giving control* to computers, even in safety critical tasks
- One example: fly-by-wire aircraft, where the machine's judgment may be trusted more than an experienced pilot
- Next on the agenda: self-driving cars, intelligent braking systems, cruise control that maintains distance from car in front...

Driving automation

1. Road Condition Reporting
2. Adaptive Cruise Control
3. Omnidirectional Collision System
4. Lane-Departure Prevention
5. Auto Parallel Park
6. Blind-Spot Sensors
7. Corner Speed



Human-Oriented Programming

- The movement away from machine-oriented views of programming toward concepts and metaphors that more closely reflect the way we ourselves understand the world
- Programmers (and users!) relate to the machine differently
- Programmers conceptualize and implement software in terms of higher-level – more human-oriented – abstractions

Agent-Oriented Software Engineering

- Programming has progressed through
 - machine code
 - assembly language
 - machine-independent programming languages
 - sub-routines
 - procedures & functions
 - abstract data types
 - objects
- to *agents*

Where does it bring us?

- *Delegation* and *Intelligence* imply the need to build computer systems that can act effectively on our behalf
- This implies:
 - The ability of computer systems to act *independently*
 - The ability of computer systems to act in a way that *represents our best interests* while interacting with other humans or systems

Interconnection and Distribution

- *Interconnection* and *Distribution* have become core motifs in Computer Science
- But Interconnection and Distribution, coupled with the need for systems to represent our best interests, implies systems that can *cooperate* and *reach agreements* (or even *compete*) with other systems that have different interests (much as we do with other people)

So Computer Science expands...

- All of these trends have led to the emergence of *multiagent systems*



2 Agents and Multiagent Systems

- An agent is a computer system that is located in a dynamic environment and is capable of *independent* action on behalf of its user or owner (figuring out what needs to be done to satisfy design objectives, rather than constantly being told)
- **Intelligent** action, probably using Artificial Intelligence tools and techniques
- An agent should satisfy a certain number of properties **[lecture 3]**

Multiagent Systems

- A multiagent system is one that consists of a number of agents, which *interact* with one-another
- In the most simple case, all agents are programmed by the same team and they collaborate to complete a task
- In the most general case, agents will be acting on behalf of users with different goals and motivations
- To successfully interact, they will require the ability to *cooperate*, *coordinate*, and *negotiate* with each other, much as people do

Related areas

- Several sub-disciplines of information technology are related to software agent technology:
 - computer networks,
 - software engineering,
 - Artificial Intelligence,
 - human-computer interaction,
 - distributed and concurrent systems,
 - ...

Micro and macro aspects

- Agent technologies can be grouped into three categories
 - Agent-level [micro level]
 - technologies and techniques concerned only with individual agents — for example, procedures for agent reasoning and learning [Agent architectures – lecture 2]
 - Interaction-level
 - technologies and techniques that concern the communication between agents
 - communication languages, interaction protocols and resource allocation mechanisms [Agent communication-lecture 4. Coordination and cooperation mechanisms-lectures 5 to 8]
 - Organization-level [macro level]
 - technologies and techniques related to agent societies as a whole
 - structure, trust, norms, obligations, etc. [Agent societies-lecture 9]

Multiagent Systems

- In Multiagent Systems, the main questions we address are:
 - What kinds of languages can agents use to communicate?
 - How can **cooperation** emerge in societies of self-interested agents?
 - How can self-interested agents recognize conflict, and how can they (nevertheless) reach **agreement**?
 - How can autonomous agents **coordinate** their activities so as to cooperatively achieve goals?

Multiagent Systems

- While these questions are all addressed in part by other disciplines (notably Economics and Social Sciences), what makes the multiagent systems field unique is that it emphasizes that the agents in question are *computational, information processing* entities.

Example 1 - Spacecraft Control

- When a space probe makes a flight to the outer planets, a ground crew is usually required to continually track its progress, and decide how to deal with unexpected eventualities. This is costly and, if decisions are required *quickly*, it is simply not practicable. NASA is investigating the possibility of making probes more *autonomous* — giving them richer decision making capabilities and responsibilities.
- NASA's *DS1 started with it in 1999 and continues improving this approach*

[Topics](#)[Missions](#)[Galleries](#)[NASA TV](#)[Follow NASA](#)[Downloads](#)[About](#)[NASA Audiences](#)

Search

[Latest](#)[Related](#)[SpaceCraft Oxygen Recovery \(SCOR\)](#)

5 months ago

[Explore: Transformative Missions and Discoveries](#)

7 months ago

[Cooperative Autonomous Distributed Robotic Explorers \(CADRE\)](#)

7 months ago

[Distributed Spacecraft Autonomy \(DSA\)](#)

10 months ago

[Deployable Composite Booms \(DCB\)](#)

a year ago

[High Performance Spaceflight Computing \(HPSC\)](#)

a year ago

[Autonomous Pop-Up Flat Folding](#)

Feb 5, 2021

Cooperative Autonomous Distributed Robotic Explorers (CADRE)

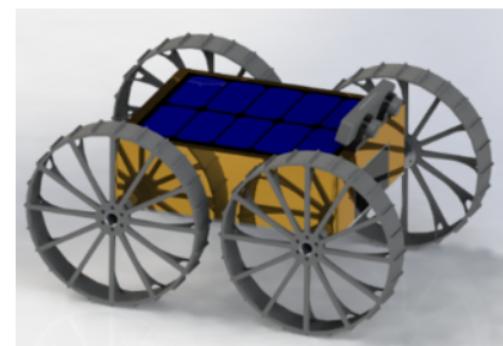


NASA's Cooperative Autonomous Distributed Robotic Explorers (CADRE) project is developing a network of shoe-box-sized mobile robots that could enable future autonomous robotic exploration of the Moon, Mars, and beyond. The CADRE robots are the latest version of NASA's A-PUFFER technology. Each robot contains an onboard computer with a wireless radio for communication and a stereo camera – which has multiple lenses and image sensors – for sensing the environment in front of it and capturing 3D imagery. NASA's Jet Propulsion Laboratory in Southern California is developing the robotic scouts, designing them to explore as a group and collect data in hard-to-reach places such as craters and caves on the Moon.

"CADRE robots could complement NASA's larger planetary robots and rovers," said Sonny Mitchell, a program element manager for NASA's Game Changing Development Program, which funds the project. "Multiple small, autonomous robots could cover more ground, potentially helping us map unexplored regions on the Moon."

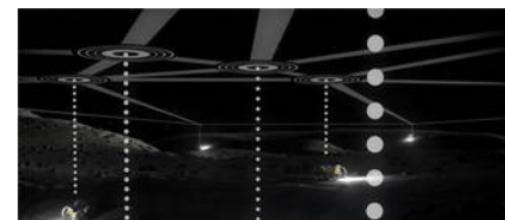
How CADRE Robots Work

During launch and the trip through space, the robots would be stored in a compact position in a larger lander. Once the lander arrives at its destination, the CADRE scouts would eject onto the surface. The explorers are designed to be agile and smart enough to explore extreme terrain on the Moon independently, without being controlled by astronauts or engineers on Earth.



Computer-aided design of the Cooperative Autonomous Distributed Robotic Exploration (CADRE) robot concept.

Credits: NASA/JPL-Caltech



Example 2 - Air Traffic Control

“A key air-traffic control system suddenly fails, leaving flights in the vicinity of the airport with no air-traffic control support. Fortunately, autonomous air-traffic control systems in nearby airports recognize the failure of their peer, and cooperate to track and deal with all affected flights.”

- Systems **taking the initiative** when necessary
- Agents **cooperating** to solve problems beyond the capabilities of any individual agent

AgentFly ATM Simulation Suite

The AgentFly ATM simulation suite is a complex tool for modeling and simulation of air traffic and air traffic management. The ATM AgentFly is fast-time gate-to-gate simulation used in NextGen and SESAR programs. The simulation consists of IFR, VFR and unmanned air traffic and main actors - air traffic controllers, pilots, and airline operation centers. The AgentFly platform is agent-based simulation framework designed to be used as Fast-Time Simulation (FTS) as well as Real-Time Simulation (RTS).



The FTS mode is suitable for what-if studies and analysis, validation of new concepts or interconnection with other FTS systems. The system allows running quickly various options, settings and parameters to evaluate changes. The RTS mode is suitable for connection with other real-time systems and can include humans-in-the-loop, e.g., pseudo-pilots or air traffic controllers.

Agent-based simulation allows precise control of simulation time, large-scale scenarios with various actors (thousands of air traffic controllers and tens of thousands of aircraft) and controlled uncertainty and randomization. The

Example 3 – e-personal assistants

- E-personal agents can assist people in different kinds of tasks
- With access to Internet, they can plan, arrange, buy, negotiate – carry out arrangements of all sorts that would normally be done by their human user
- For example, preparing a holiday trip

Multiagent Systems is Interdisciplinary

- The field of Multiagent Systems is influenced and inspired by many other fields:
 - Economics
 - Philosophy
 - Game Theory
 - Logic
 - Ecology
 - Social Sciences
- This can be both a strength (infusing well-founded methodologies into the field) and a weakness (there are many different views as to what the field is about)
- This has analogies with Artificial Intelligence itself

3-Objections to MAS (I)

- Isn't it all just Distributed/Concurrent Systems?
There is much to learn from this community,
but:
 - Agents are assumed to be autonomous, capable of making independent decision – so they need mechanisms to synchronize and coordinate their activities **at run time**
 - Agents are (can be) self-interested, so their interactions are “economic” encounters (negotiation)

Objections to MAS (II)

■ Isn't it all just AI?

- We don't need to solve all the problems of Artificial Intelligence (i.e., all the components of intelligence) in order to build really useful agents
- Classical AI ignored *social* aspects of agency. These are important parts of intelligent activity in real-world settings

Objections to MAS (III)

■ Isn't it all just Economics/Game Theory?

These fields also have a lot to teach us in multiagent systems, but:

- Insofar as game theory provides *descriptive* concepts, it doesn't always tell us *how* to compute solutions; we're concerned with computational, resource-bounded agents
- Some assumptions in economics/game theory (such as a *rational agent*) may not be valid or useful in building artificial agents

Objections to MAS (IV)

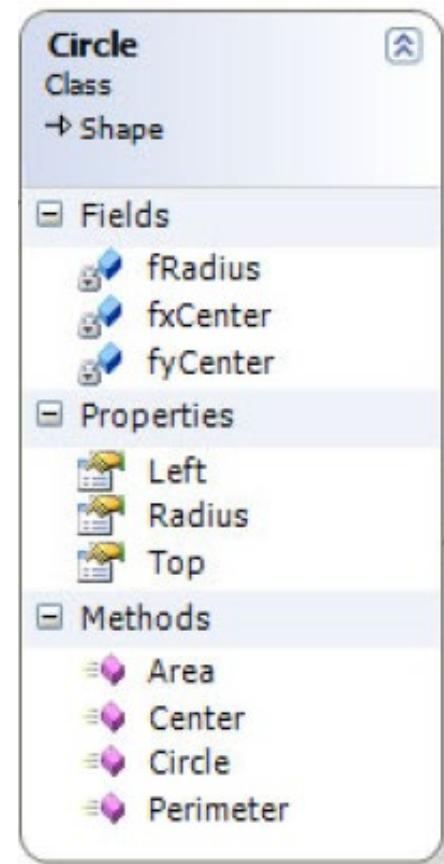
- Isn't it all just Social Science?
 - We can draw insights from the study of human societies, but there is no particular reason to believe that artificial societies will be constructed in the same way
 - Again, we have inspiration and cross-fertilization, but hardly subsumption

4-Agents versus related technologies

- If agents are autonomous entities that display an intelligent behaviour, what makes them so different from other well known techniques, like object-oriented programming or intelligent (knowledge-based) systems?

Agents and Objects

- Are agents just objects by another name?
- Object:
 - encapsulates some state
 - communicates via message passing
 - has methods, corresponding to operations that may be performed on the state



Agents vs Objects (I)

□ *Agents are autonomous*

- Agents embody a stronger notion of autonomy than objects
- They decide for themselves whether or not to perform an action on request from another agent
- When a method is invoked on an object, it is always executed

Agents vs Objects (II)

- **Agents are *smart, intelligent***
 - Capable of flexible (reactive, proactive, social) behaviour
 - The standard object model has nothing to say about such types of behaviour
- **Agents are *active***
 - A multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of active control

Agents vs Objects (III)

Objects

- Encapsulate state and control over it via methods
- **Passive** – have no control over a method execution
- Non autonomous
- Reactive to events

Agents

- Encapsulate state, control over it via *actions and goals*
- **Active** – can decide when to act and how
- *Autonomous*
- *Proactive*

In summary ...

- *Objects do it for free*

An object cannot refuse a method invocation

- *Agents do it because they want to*

The service requester is authorised, the agent has enough resources available, the action is convenient for the agent, ...

- *Agents do it for money*

The agent can get an economic profit

Agents and Expert Systems

- Aren't agents just **expert systems** by another name?
- Expert systems contain typically disembodied 'expertise' about some (abstract) domain of discourse
- Example: MYCIN knows about blood diseases in humans
 - It has a wealth of knowledge about blood diseases, in the form of rules
 - A doctor can obtain expert advice about blood diseases by giving MYCIN facts, answering questions, and posing queries

Agents and Expert Systems

■ Main differences:

- agents *situated in an environment*:
MYCIN is not aware of the world — the only information that it obtains is by asking questions to the user
- agents *act*:
MYCIN does not operate on patients
- Expert systems typically work alone, whereas agents *cooperate* to achieve a common goal.

■ Sometimes an expert system is *agentified* and included in a MAS

5-Some Views of the Field

- *Agents as a paradigm for software engineering:*

Software engineers have derived a progressively better understanding of the characteristics of complexity in software. It is now widely recognized that *interaction* is probably the most important single characteristic of complex software

Agents as design metaphor

■ Agent-oriented software engineering

- Agents allow software designers and developers to structure an application using autonomous, communicative components.
- In this sense, software agents offer a new and often more appropriate route to the development of complex computational systems, especially in open and dynamic environments.
 - Modularity, reusability, collaborative behaviour



Agents as source of technologies

- Software agent technologies span a range of specific **techniques and algorithms**
 - balancing **reaction** and **deliberation** in individual agent architectures [**lecture 2**]
 - **learning** from and about other agents in the environment
 - eliciting and acting upon **user preferences**
 - finding ways to **negotiate** and **cooperate** with other agents
 - developing appropriate means of forming and managing **coalitions** (and other organizations)

Social simulation

- *Agents as a tool for understanding human societies:*

Multiagent systems provide a new tool for simulating societies, which may help shed some light on various kinds of social processes.

Agent-based simulation

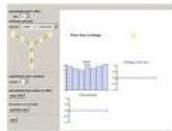
- Multi-agent systems provide strong models for representing complex and dynamic real-world environments.
 - modelling of the impact of climate change on biological populations
 - modelling the impact of public policy options on social or economic behaviour
 - modelling intelligent buildings
 - modelling traffic systems
 - modelling biological populations

Multi-agent Modeling

DEMONSTRATIONS

Demonstrations 1 - 18 of 18

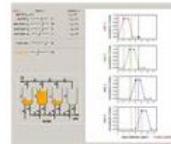
 Subscribe to RSS feed



Exchange Networks



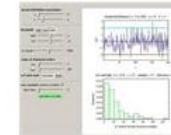
Fishing with Long Line or Gill Net?



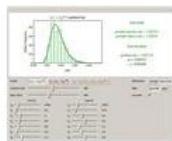
Estimating Processing Times in a Row of Storage Tanks



Ant Colony Optimization (ACO)



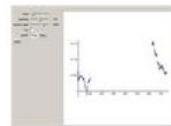
Estimating the Time between Mishaps from Quality Control Data



Assessing Total Risk from Interacting Factors



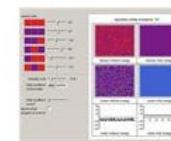
Boids: Simulated Flocking Behavior



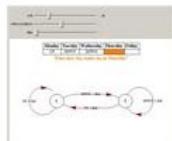
Mean Distance of Agents in Diffusion-Limited Aggregation



Sugarscape: Agent-Based Modeling



Emulating Land Use Evolution with a Cellular Automaton



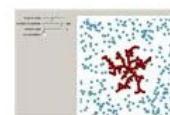
Trader Dynamics in Minimal Models of Financial Complexity



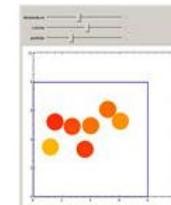
Garbage Collection by Ants



Predator-Prey Ecosystem: A Real-Time Agent-Based Simulation



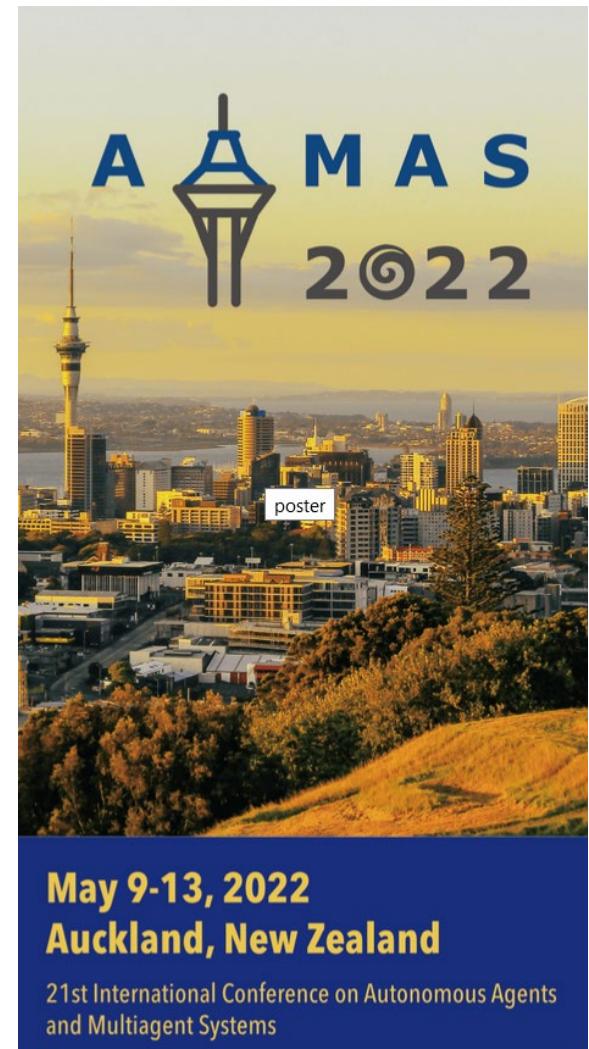
Diffusion-Limited Aggregation: A Real-Time Agent-Based Simulation



Molecular Dynamics of Lennard-Jones Particles Using the Velocity Verlet Algorithm

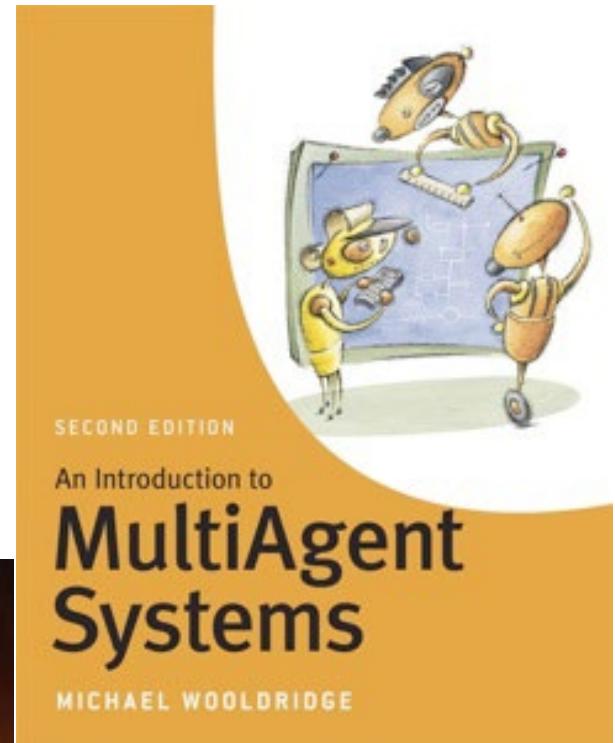
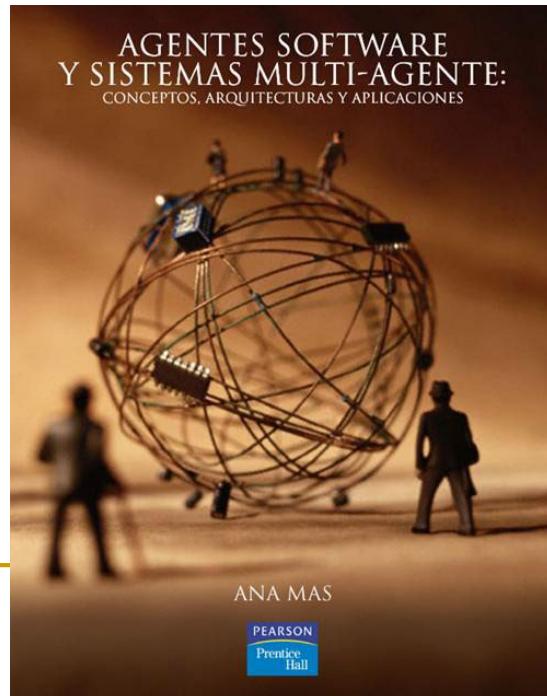
Research on MAS

- Organizations: [IFAAMAS](#)
- Conferences: [AAMAS](#), [PAAMS](#)
- Summer schools: [EASSS](#), [ISSIA](#)
- Journals: [Autonomous Agents and Multi-Agent Systems \(Springer\)](#)

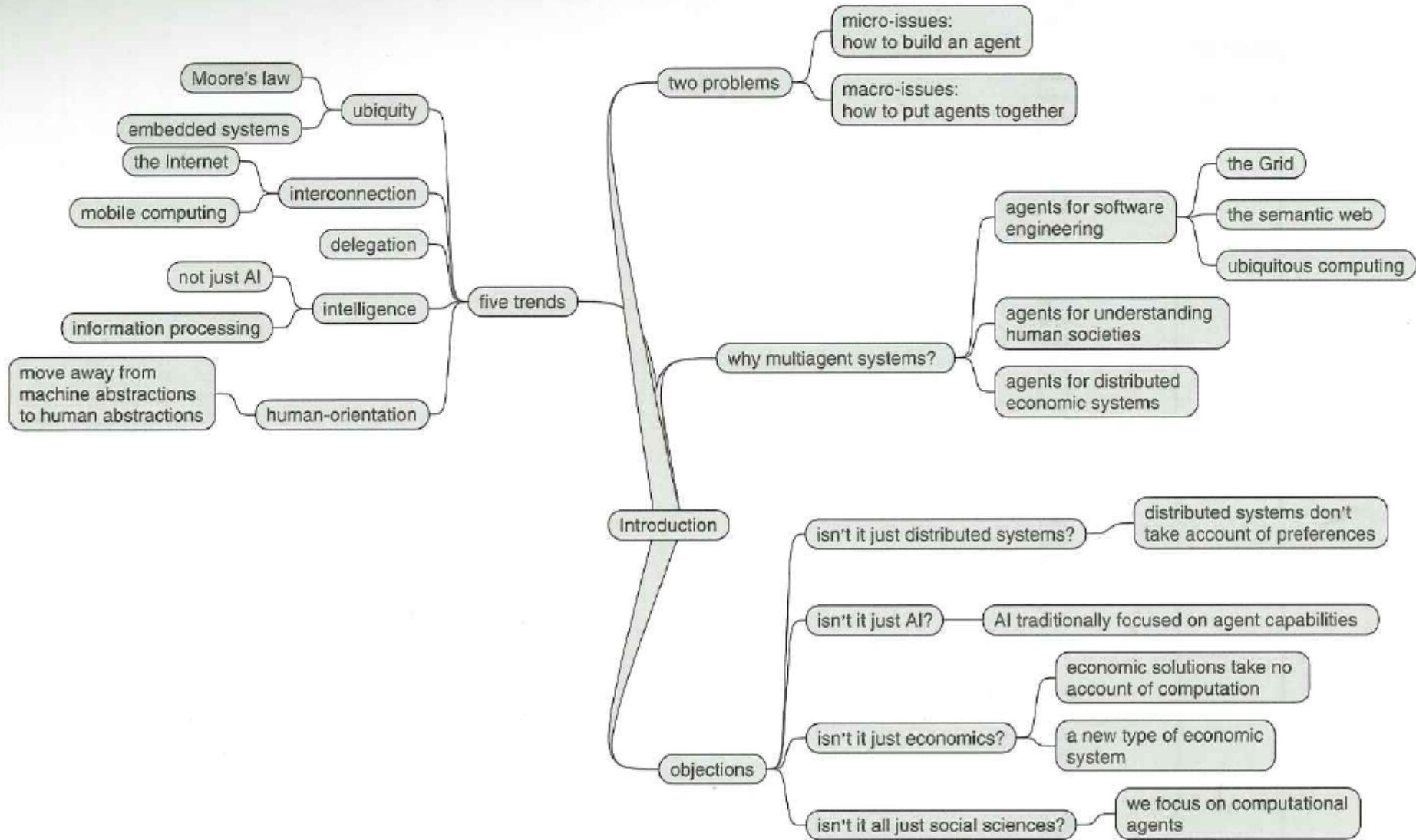


Readings for this week

- M.Wooldridge: *An introduction to MultiAgent Systems* – *chapter 1, beginning c.2*
- A.Mas: *Agentes software y sistemas multi-agente: conceptos, arquitecturas y aplicaciones* – *initial part of chapter 1*



Mindmap, chapter 1 (Wooldridge)



Videos (M.Wooldridge)

- Computer Science trends leading to Agent Technology.
- Basic definition of agent and multiagent system.
- Common objections to Agent Technology.
- Objects vs. Agents.

LECTURE 2: Intelligent Agents – Properties & Abstract Architecture

Introduction to Multi-Agent Systems (MESIIA, MIA)
URV

What is an Agent?

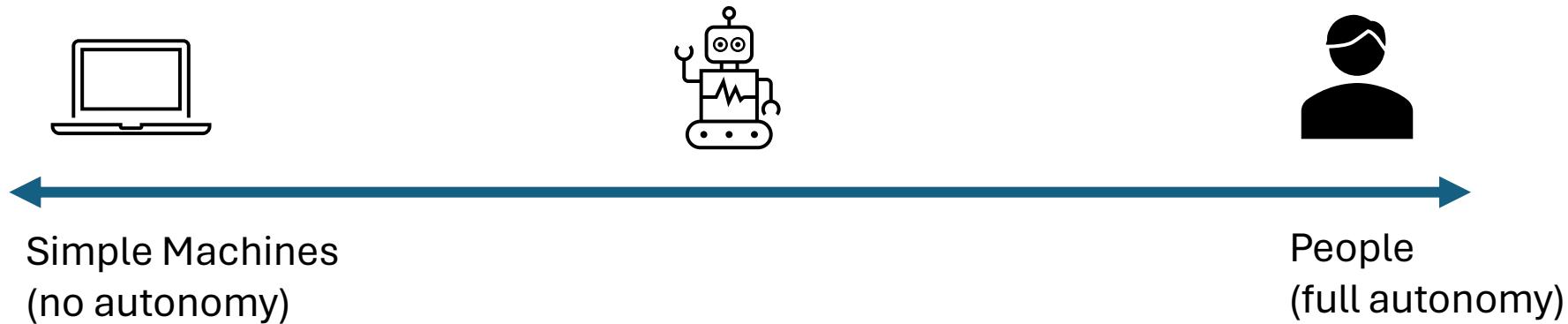
- There is no commonly accepted definition of the term *Agent*.
- The central point about agents is they are *autonomous*.

“An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous actions* in this environment in order to meet its delegated objectives.”

- It is about decisions
 - An agent must:
 - Choose *what* action to perform.
 - Decide *when* to perform an action.

Autonomy

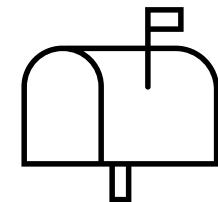
- Autonomy is a *spectrum*



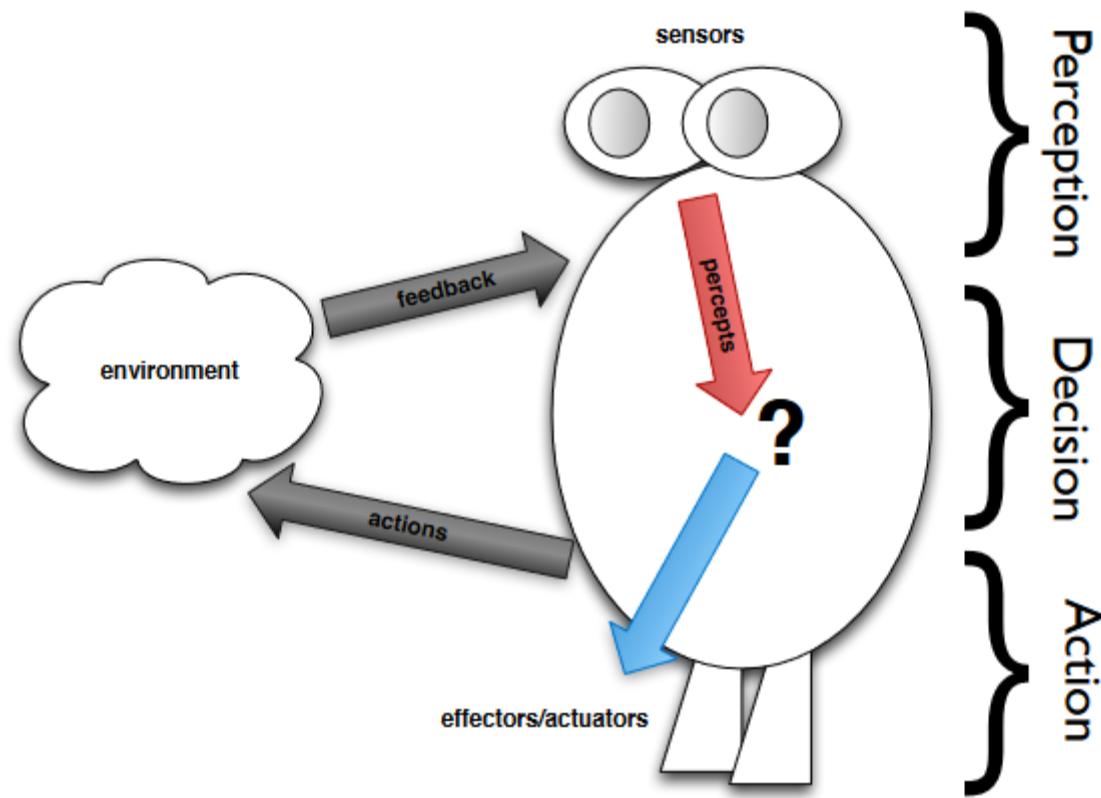
- Autonomy is adjustable
 - Human will make a decision with a remarkable higher benefit
 - High degree of uncertainty about the environment
 - The decision might cause harm, or
 - The agent lacks the capabilities to decide itself

Simple Agents

- Control Systems
 - Example: *Thermostat* (physical environment)
 - Delegated goal: maintain room temperature
 - Actions: switch on/off
 - Can be adjusted?
- Software Demons
 - Example: *UNIX biff program* (software/virtual environment)
 - Delegated goal: monitor for incoming email and flag it
 - Actions: display GUI icons



Agent and Environment



Properties of Environments

- Why do we need the properties of the environment?
 - Affect the agents, and how we build them
- Common to categorise environments along some different dimensions.
 - Fully observable vs partially observable
 - Deterministic vs non-deterministic
 - Static vs dynamic
 - Discrete vs continuous
 - Episodic vs non-episodic

Properties of Environments

- Fully observable vs partially observable
 - An accessible or fully observable environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state.
 - Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible, or partially observable.
 - The more accessible an environment is, the simpler it is to build agents to operate in it.

Properties of Environments



- Fully observable vs partially observable
 - Example – 1: Consider a multi-agent system for playing a cooperative card game, such as bridge or spades. The agents are the players, and their **goal** is to **maximize** their **team's score**. The agents *can see* their own cards, but not the cards of the other players. The agents can **communicate** with their partner, but not with their opponents. Is this environment accessible or inaccessible for the agents? Explain your answer.
 - Example – 2: Consider a multi-agent system for playing a tic-tac-toe game. The agents are the players, and their **goal** is to **win** the game or avoid losing. The agents can **see** the entire board, and they **know** the rules of the game. The agents take turns to place their symbol (X or O) on an empty cell of the board. Is this environment accessible or inaccessible for the agents? Explain your answer.

Properties of Environments

- Deterministic vs non-deterministic
 - A deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action.
 - The physical world can to all intents and purposes be regarded as non-deterministic.
 - Fully observable, deterministic environment => No worry about uncertainty.
 - We'll follow Russell and Norvig in calling environments stochastic if we quantify the non-determinism using probability theory.
 - Non-deterministic environments present greater problems for the agent designer.

Properties of Environments

- Static vs dynamic
 - A **static** environment is one that can be assumed to remain unchanged except by the performance of actions by the agent.
 - A **dynamic** environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control.
 - The physical world is a highly dynamic environment.
 - One reason an environment may be dynamic is the presence of other agents.

Properties of Environments

- Discrete vs continuous
 - An environment is discrete if there are a fixed, finite number of actions and percepts in it.
 - Otherwise, it is continuous
 - Examples:
 - Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.
 - We often assume that the environment is discrete for simplicity.

Properties of Environments

- Episodic vs non-episodic
 - An **episodic** environment is one in which the current decision of the agent affects only the current episode, and not the future episodes.
 - An episode is a single experience that the agent has with the environment
 - No need to consider the long-term consequences
 - Act based on the current percept
 - A **non-episodic** (sequential) environment is one in which the current decision of the agent affects not only the current episode, but also the future episodes.
 - Agent must consider the long-term consequences of its actions
 - Act based on the history of percepts and actions

Intelligent Agents

- What is an intelligent agent?
- “What is Intelligence?” itself.
- Need to list the capabilities (properties) that we might expect an intelligent agent to have.
- We typically think of an intelligent agent as exhibiting 3 types of behaviour:
 - Reactive (environment aware)
 - Pro-active (goal-driven)
 - Social Ability

Reactivity

- If a program's environment is guaranteed to be fixed, the program need never worry about its own success or failure
- Program just executes blindly.
 - Example of fixed environment: compiler
- The real world is not like that: most environments are dynamic and information is incomplete
- Software is hard to build for dynamic domains: program must take into account possibility of failure
- A reactive system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful)

Proactiveness

- Reacting to an environment is easy
 - e.g., stimulus → response rules
- But we generally want agents to *do things for us.*
 - Hence goal directed behaviour
- *Pro-activeness* = generating and attempting to achieve goals; not driven solely by events; taking the initiative.

Social Ability

- The real world is a *multi-agent* environment: we cannot go around attempting to achieve goals without taking others into account.
 - Some goals can only be achieved by interacting with others.
- *Social ability* in agents is the ability to interact with other agents (and possibly humans) via *cooperation*, *coordination* and *negotiation*.
 - At the very least, it means the ability to communicate ...

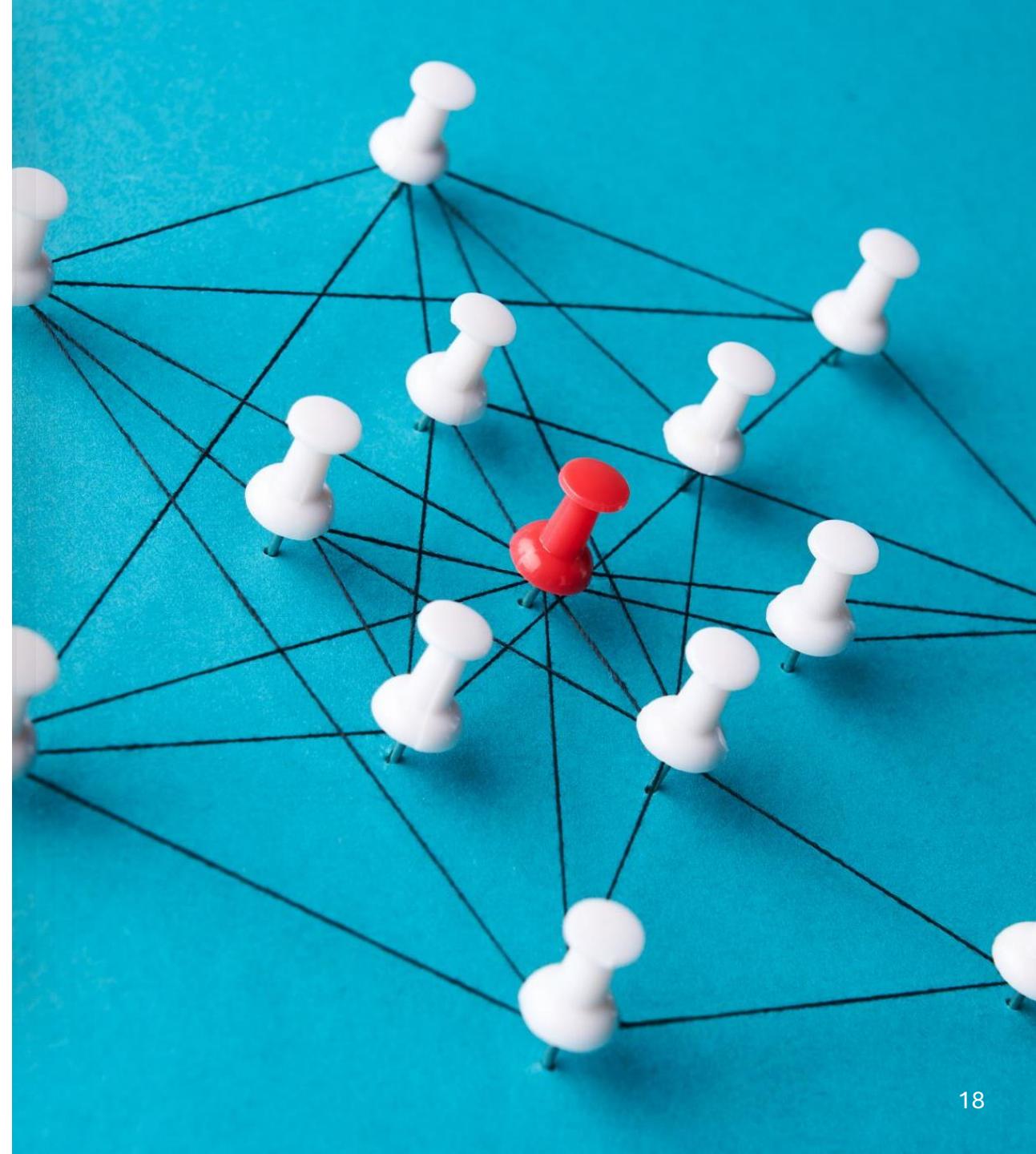


Social Ability: Cooperation

- Cooperation is working together as a team to achieve a shared goal.
- Often prompted either by the fact that no one agent can achieve the goal alone, or that cooperation will obtain a better result (e.g., get result faster)

Social Ability: Coordination

- Coordination is managing the interdependencies between activities.
- For example, if there is a non-sharable resource that you want to use and I want to use, then we need to coordinate.



Social Ability: Negotiation



- Negotiation is the ability to reach agreements on matters of common interest.
- For example:
 - You have one TV in your house; you want to watch a movie, your housemate wants to watch football.
 - A possible deal: watch football tonight, and a movie tomorrow.
- Typically involves offer and counter-offer, with compromises made by participants.

Some Other Properties...

- Mobility
 - The ability of an agent to move.
For software agents this movement is around an electronic network.
- Rationality
 - Whether an agent will act in order to achieve its goals, and will not deliberately act so as to prevent its goals being achieved.
- Learning
 - Whether agents improve performance over time.
- Veracity
 - Whether an agent will knowingly communicate false information.

Abstract Architectures for Agents

- Assume the environment may be in any of a finite set E of discrete states:

$$E = \{e, e', \dots\}$$

- Agents are assumed to have a collection of possible actions, Ac , available to them, which transform the state of the environment:

$$Ac = \{\alpha, \alpha', \dots\}$$

- A *run*, r , of an agent in an environment is a sequence of environment states and actions:

$$r: e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_2 \xrightarrow{\alpha_3} \dots \xrightarrow{\alpha_{u-1}} e_u$$

Abstract Architectures for Agents - Runs

- Let:

- R be the set of all such possible finite sequences (over E and Ac)
- $R^{Ac} \subseteq R$ be the subset of these that end with an action, $\alpha_i \in Ac$
- $R^E \subseteq R$ be the subset of these that end with an environment state, $e_j \in E$

- We will use r, r', \dots to stand for all members of R

Abstract Architectures for Agents – Env.

- A *state transformer* function represents behaviour of the environment:

$$\tau: R^{Ac} \rightarrow 2^E$$

- Assumptions on the environments:
 - history dependent: the next state not only dependent on the action of the agent, but an earlier action may be significant
 - non-deterministic: There is some uncertainty about the result
- If $\tau(r) = \phi$, then there are no possible successor states to r . In this case, we say that the system has *ended* its run

Abstract Architectures for Agents – Agents

- We can think of an agent as being a *function* which maps runs to actions

$$Ag: R^E \rightarrow Ac$$

- Thus, an agent makes a decision about what action to perform
 - based on the history of the system that it has witnessed to date
- Environments assumed to be non-deterministic, what about agents?
- We can now define AG to be the set of all (possible) agents.

Abstract Architectures for Agents – System

- A system is a pair containing an agent and an environment, $\langle Ag, Env \rangle$.
- Any system will have associated with it a set of possible runs.
 - We denote the set of runs of agent Ag in environment Env by:

$$R(Ag, Env)$$

- Assume that this only contains *terminated* runs.

Abstract Architectures for Agents – Systems

Formally, a sequence

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \dots)$$

represents a run of an agent Ag in environment $Env = \langle E, e_0, \tau \rangle$ if

1. $e_0 \in E$ is the initial state of Env
2. $\alpha_0 = Ag(e_0)$; and
3. For all $u > 0$,

$$e_u \in \tau((e_0, \alpha_0, \dots, \alpha_{u-1})), \text{ and}$$
$$\alpha_u = Ag((e_0, \alpha_0, \dots, e_u))$$

Why the notation?

- It allows us to get a precise handle on some ideas about agents.
 - For example, we can tell when two agents are the same
- Of course, there are different meanings for “same”. Here is one specific one.

Two agents, Ag_1 and Ag_2 are said to be behaviorally equivalent with respect to Env iff $R(Ag_1, Env) = R(Ag_2, Env)$

- We won't be able to tell two such agents apart by watching what they do.

Purely Reactive Agents

- These agents decide what to do without reference to their history
 - they base their decision making entirely on the present, with no reference at all to the past
- Then, we can formally define such an agent as:

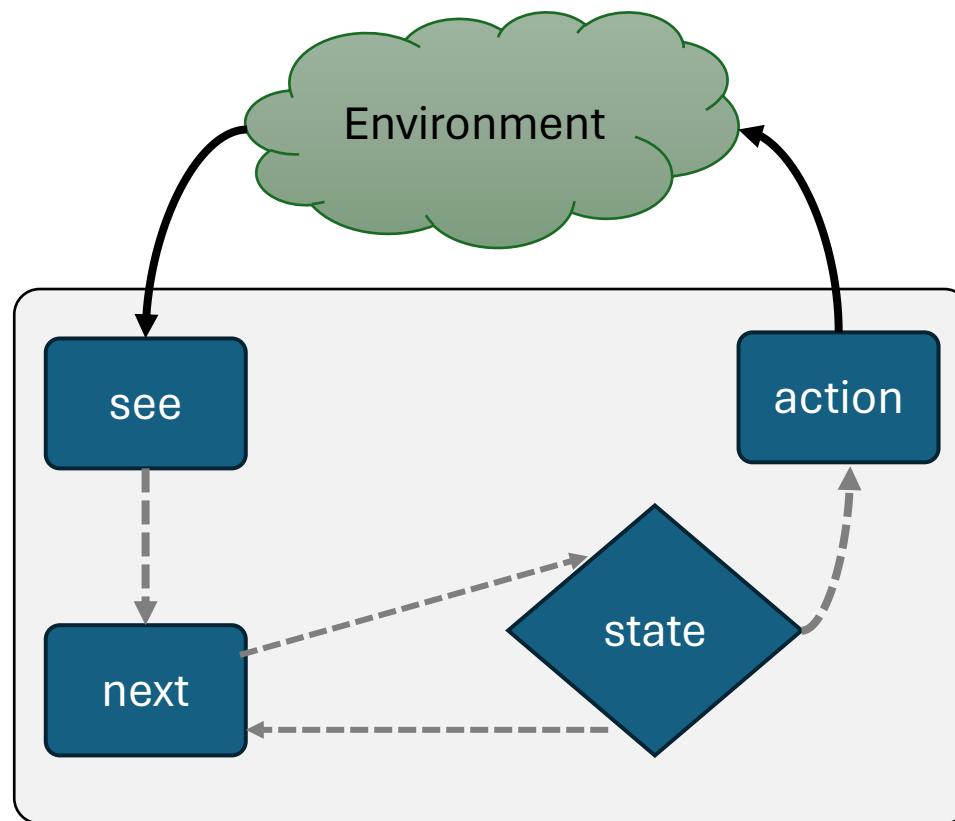
$$Ag: E \rightarrow Ac$$

- Example: A *thermostat* is a purely reactive agent

$$Ag(e) = \begin{cases} \text{off} & \text{if } e = \text{temperature OK} \\ \text{on} & \text{otherwise} \end{cases}$$

Agents with state

- Such kind of agents have some internal data structure, known as **state**.



Perception

- The see function is the agent's ability to observe its environment, whereas the action function represents the agent's decision making process.
- Output of the see function is a percept:

$$\text{see}: E \rightarrow \text{Per}$$

Actions and Next State Functions

- An additional function `next` is introduced, which maps an internal state and percept to an internal state:

$$next: I \times Per \rightarrow I$$

- This says how the agent updates its view of the world *when it gets a new percept*
- The action-selection function `action` is now defined as a mapping from internal states to actions:

$$action: I \rightarrow Ac$$

Agent Control Loop

1. Agent starts at some initial internal state i_0
2. Observes its environment state e , and generates a percept $see(e)$
3. Internal state of the agent is then updated via $next$ function,
becoming $i_0 = next(i_0, see(e))$
4. The action selected by the agent is $action(i_0)$. This action is then performed.
5. Goto (2).

Tasks for Agents

- We build agents in order to carry out tasks for us
 - The task must be specified by us
- But we want to tell agents what to do without telling them how to do it.
 - How can we make this happen???

Utility functions

- One idea:
 - associated rewards with states that we want agents to bring about.
 - We associate utilities with individual states (the task of the agent is then to bring about states that maximise utility)
- A task specification is then a function which associates a real number with every environment state:

$$u: E \rightarrow \mathbb{R}$$

Local Utility Functions

- But, what is the value of a run?
 - minimum utility of state on run?
 - maximum utility of state on run?
 - sum of utilities of states on run?
 - average?
- Disadvantage:
 - difficult to specify *a long term* view when assigning utilities to individual states

Example of local utility function

- Goal is to select actions to maximise future rewards
 - Each action results in moving to a state with some assigned reward
 - Allocation of that reward may be immediate or delayed (e.g. until the end of the run)
 - It may be better to sacrifice immediate reward to gain more long-term reward
 - What actions maximise the reward?

$r = -0.04$ (unless stated otherwise)

1				$r=+1$ 👍
2				$r=-1$ 👎
3				

1 2 3 4

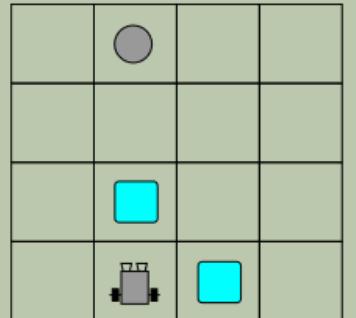
Utilities over Runs

- Another possibility: assigns a utility not to individual states, but to runs themselves:

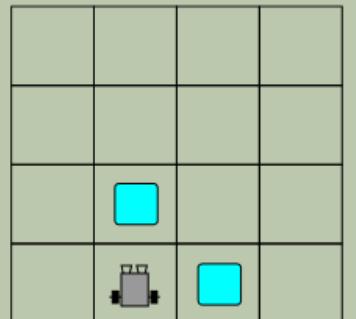
$$u: R \rightarrow \mathbb{R}$$

- Such an approach takes an inherently *long-term view*.
- To see where utilities might come from, let's look at an example

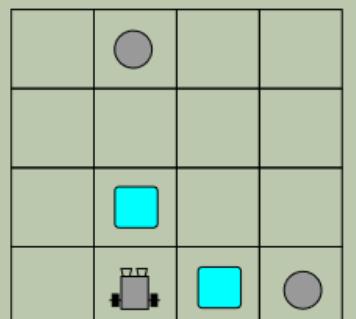
The agent starts to push a tile towards the hole.



But then the hole disappears!!!



Later, a much more convenient hole appears (bottom right)



Example: Utility in the Tileworld

- Simulated two-dimensional grid environment on which there are agents, tiles, obstacles, and holes.
- An agent can move in four directions:
 - up, down, left, or right
 - If it is located next to a tile, it can push it.
- Holes have to be filled up with tiles by the agent
 - An agent scores points by filling holes with tiles, with the aim being to fill as many holes as possible.
- TILEWORLD changes with the random appearance and disappearance of holes.

Example: Utilities in the Tileworld

- Utilities are associated over runs, so that more holes filled is a higher utility.
- Utility function defined as follows:
- Thus:
 - if agent fills all holes, utility = 1
 - if agent fills no holes, utility = 0
$$u(r) \doteq \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

Expected Utility

- Let $P(r|Ag, Env)$ be the probability that run r occurs when agent Ag is placed in environment Env .

$$\sum_{r \in R(Ag, Env)} P(r|Ag, Env) = 1$$

- The expected utility (EU) of agent Ag in environment Env (given P , u), is then:

$$EU(Ag, Env) = \sum_{r \in R(Ag, Env)} u(r) P(r|Ag, Env)$$

Optimal Agents

- The optimal agent $Ag_{opt} \in AG$ in an environment Env is the one that maximizes expected utility, i.e., EU :

$$Ag_{opt} = \arg \max_{Ag \in AG} EU(Ag, Env)$$

Example 1

- Consider the environment $Env = \langle E, e_0, \tau \rangle$ defined as follows:
 - $E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$
 - $\tau((e_0, \alpha_0)) = \{e_1, e_2\}, \tau((e_0, \alpha_1)) = \{e_3, e_4, e_5\}$
- There are two agents possible with respect to this environment
 - $Ag_1(e_0) = \alpha_0, Ag_2(e_0) = \alpha_1$
- The probabilities and utility function of the various runs are as follows:
 - $P((e_0, \alpha_0, e_1) | Ag_1, Env) = 0.4, u((e_0, \alpha_0, e_1)) = 8$
 - $P((e_0, \alpha_0, e_2) | Ag_1, Env) = 0.6, u((e_0, \alpha_0, e_2)) = 11$
 - $P((e_0, \alpha_1, e_3) | Ag_2, Env) = 0.1, u((e_0, \alpha_1, e_3)) = 70$
 - $P((e_0, \alpha_1, e_4) | Ag_2, Env) = 0.2, u((e_0, \alpha_1, e_4)) = 9$
 - $P((e_0, \alpha_1, e_5) | Ag_2, Env) = 0.7, u((e_0, \alpha_1, e_5)) = 10$
- What are the expected utilities of the agents for this utility function?

Example 2

- Consider the environment $Env = \langle E, e_0, \tau \rangle$ defined as follows:
 - $E = \{e_0, e_1, e_2, e_3, e_4, e_5\}$
 - $\tau((e_0, \alpha_0)) = \{e_1, e_2\}, \tau((e_1, \alpha_1)) = \{e_3\}, \tau((e_2, \alpha_2)) = \{e_4, e_5\}$
- There are two agents possible with respect to this environment
 - $Ag_1(e_0) = \alpha_0, Ag_1(e_1) = \alpha_1$
 - $Ag_2(e_0) = \alpha_0, Ag_2(e_2) = \alpha_2$
- The probabilities and utility function of the various runs are as follows:
 - $P((e_0, \alpha_0, e_1) | Ag_1, Env) = 0.5, u((e_0, \alpha_0, e_1)) = 4$
 - $P((e_0, \alpha_0, e_2) | Ag_1, Env) = 0.5, u((e_0, \alpha_0, e_2)) = 3$
 - $P((e_1, \alpha_1, e_3) | Ag_1, Env) = 1, u((e_1, \alpha_1, e_3)) = 7$
 - $P((e_0, \alpha_0, e_1) | Ag_2, Env) = 0.1, u((e_2, \alpha_2, e_4)) = 3$
 - $P((e_0, \alpha_0, e_2) | Ag_2, Env) = 0.9, u((e_2, \alpha_2, e_5)) = 2$
 - $P((e_2, \alpha_2, e_4) | Ag_2, Env) = 0.4$
 - $P((e_2, \alpha_2, e_5) | Ag_2, Env) = 0.6$
- What are the expected utilities of the agents for this utility function?

Predicate Task Specifications

- A special case of assigning utilities to histories is to assign 0 (false) or 1 (true) to a run.
 - If a run is assigned 1, then the agent succeeds on that run, otherwise it fails.
- Call these predicate task specifications.
 - Denote predicate task specification by Ψ

$$\Psi: R \rightarrow \{0, 1\}$$

Task Environments

- A task environment is a pair $\langle Env, \Psi \rangle$ where Env is an environment and, Ψ is the task specification.
- Let the set of all task environments be defined by: $T\mathcal{E}$
- A task environment specifies:
 - the properties of the system the agent will inhabit;
 - the criteria by which an agent will be judged to have either failed or succeeded.

Task Environments

- To denote set of all runs of the agent Ag in environment Env that satisfy Ψ , we write:

$$R_\Psi(Ag, Env) = \{r \mid r \in R(Ag, Env) \text{ and } \Psi(r) = 1\}$$

- We then say that an agent Ag succeeds in task environment $\langle Env, \Psi \rangle$ if:

$$R_\Psi(Ag, Env) = R(Ag, Env)$$

We could also write this as:

$$\forall r \in R(Ag, Env), \text{ we have } \Psi(r) = 1$$

However, this is a bit **pessimistic**: if the agent fails on a single run, we say it has failed overall.

A more **optimistic** idea of success is:

$$\exists r \in R(Ag, Env), \text{ we have } \Psi(r) = 1$$

which counts an agent as successful as soon as it completes a single successful run.

The Probability of Success

- If the environment is non-deterministic, the τ returns a set of possible states.
 - We can define a probability distribution across the set of states.
 - Let $P(r|Ag, Env)$ denote probability that run r occurs if agent Ag is placed in environment Env .
 - Then the probability $P(\Psi|Ag, Env)$ that Ψ is satisfied by Ag in Env would then simply be:

$$P(\Psi|Ag, Env) = \sum_{r \in R_\Psi(Ag, Env)} P(r|Ag, Env)$$

Achievement and Maintenance Tasks

- The idea of a predicate task specification is admittedly abstract.
- It generalises two common types of tasks, *achievement* tasks and *maintenance* tasks

Achievement and Maintenance Tasks

- An **achievement task** is specified by a set G of “good” or “goal” states: $G \subseteq E$.
 - The agent succeeds if it is guaranteed to bring about at least one of these states (we don’t care which, as all are considered good).
 - The agent **succeeds** if in an achievement task it can **force the environment** into one of the goal states $g \in G$
- An **maintenance task** is specified by a set B of “bad” or “undesirable” states: $B \subseteq E$.
 - The agent succeeds if it manages to avoid all states in B - if it never performs actions which result in any state $b \in B$ occurring

Readings for this week

- M.Wooldridge: An introduction to MultiAgent Systems – Ch. 2 Intelligent Agents
- Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents”, Stan Franklin and Art Graesser. ECAI ‘96 Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages. pp 21-35

LECTURE 3: Deductive Reasoning Agents

Introduction to Multi-Agent Systems (MESIIA, MIA)
URV

Agent Architectures

Pattie Maes (1991)

“... [A] particular methodology for building [agents]. It specifies how . . . the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions . . . and future internal state of the agent. An architecture encompasses **techniques** and **algorithms** that support this methodology ...”

Leslie Kaebling (1991)

“... [A] specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the **data and control flow** among the modules. A more abstract view of an architecture is as a general methodology for designing particular modular decompositions for particular tasks ...”

Classes of Architecture

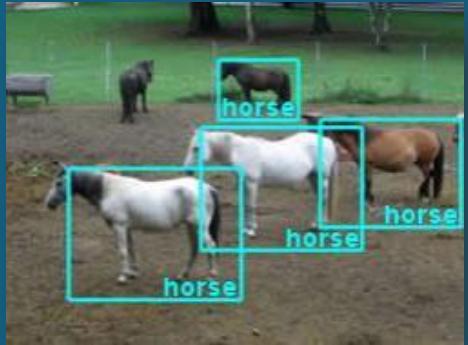
- 1956–present: Symbolic Reasoning Agents
 - Agents make decisions about what to do via symbol manipulation.
 - Its purest expression, proposes that agents use explicit logical reasoning in order to decide what to do.
- 1985–present: Reactive Agents
 - Problems with symbolic reasoning led to a reaction against this
 - led to the reactive agents movement
- 1990–present: Hybrid Agents
 - Hybrid architectures attempt to combine the best of reasoning and reactive architectures

Symbolic Reasoning Agents

- The classical approach to building agents is to view them as a particular type of knowledge-based system and bring all the associated methodologies of such systems to bear.
 - This paradigm is known as symbolic AI.
- We define a deliberative agent or agent architecture to be one that:
 - contains an explicitly represented, symbolic model of the world;
 - makes decisions (for example about what actions to perform) via symbolic reasoning.

Two issues

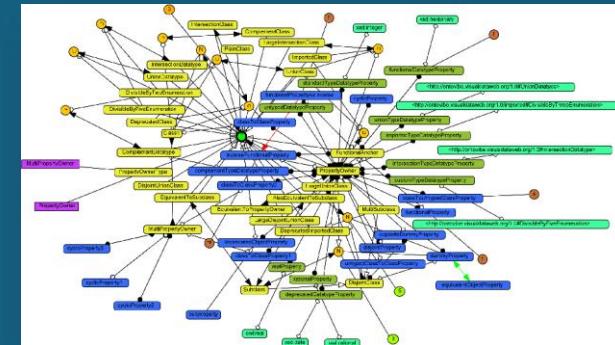
The Transduction Problem Identifying objects is hard!!!



The transduction problem is that of translating the real world into an accurate, adequate symbolic description, in time that description to be useful.

This has led onto research into vision, speech, language understanding, etc.

The Representation/Reasoning Problem Representing objects is harder!!!



How to symbolically represent information about complex real-world entities and processes, and how to get agents to reason with this information in time for the results to be useful.

This has led onto research into knowledge representation, automated reasoning, planning, etc.

Most researchers accept that neither problem is anywhere near solved.

The representation / reasoning problem

- The underlying problem with knowledge representation/ reasoning lies with the complexity of symbol manipulation algorithms.
 - In general, many (most) search-based symbol manipulation algorithms of interest are highly intractable.
 - Hard to find compact representations.
- Because of these problems, some researchers have looked to alternative techniques for building agents; we look at these later.

Deductive Reasoning Agents

- How can an agent decide what to do using theorem proving?
 - Basic idea is to use logic to encode a theory stating the best action to perform in any given situation.
- Let:
 - ρ be the theory (typically a set of rules);
 - Δ be a logical database that describes the current state of the world;
 - Ac be the set of actions the agent can perform;
 - $\Delta \vdash_{\rho} \varphi$ means that φ can be proved from Δ using ρ .

Deductive Reasoning Agents

- How does this fit into the abstract description we talked about last time?
 - The perception function is as before:

$$\text{see}: E \rightarrow \text{Per}$$

- of course, this is (much) easier said than done.
- The next state function revises the database Δ :

$$\text{next}: \Delta \times \text{Per} \rightarrow \Delta$$

- And the action function?

Action Function

for each $\alpha \in Ac$ do ; *try to find an action explicitly prescribed*

 if $\Delta \vdash_{\rho} Do(\alpha)$ then

 return α

 end if

end for

for each $\alpha \in Ac$ do ; *try to find an action not excluded*

 if $\neg \Delta \vdash_{\rho} Do(\alpha)$ then

 return α

 end if

end for

return *null* ; *no action found*

An example: The Vacuum World

The Vacuum World

The goal is for the robot to clear up all the dirt.

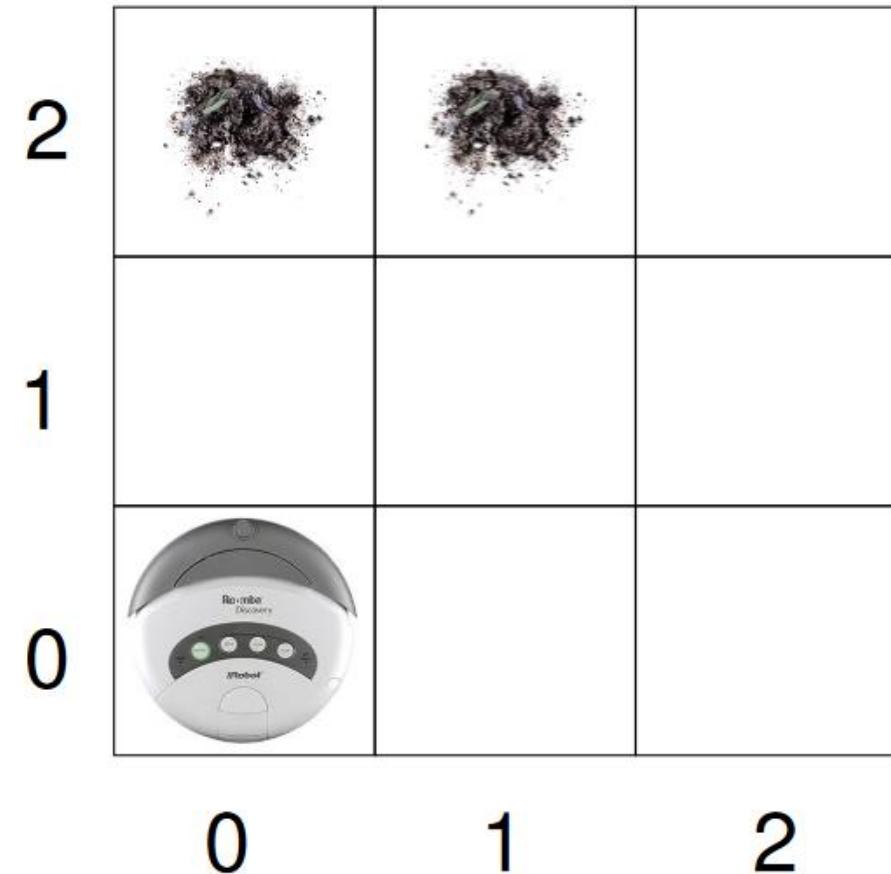
Uses 3 domain predicates in this exercise:

- $In(x, y)$ Agent is at (x, y) cell
- $Dirt(x, y)$ There is a dirt at (x, y)
- $Facing(d)$ The agent is facing direction d

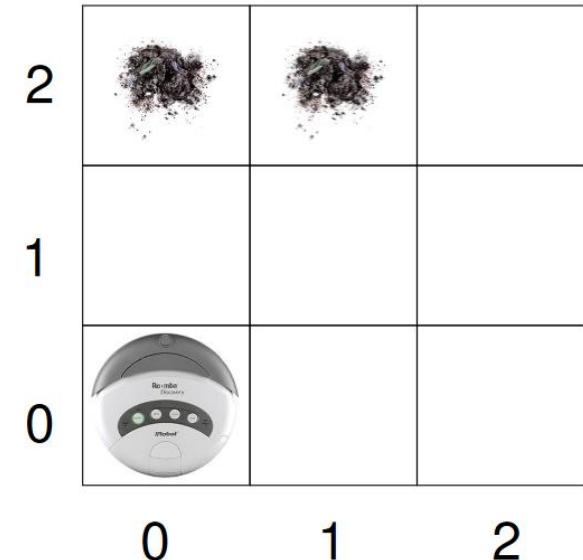
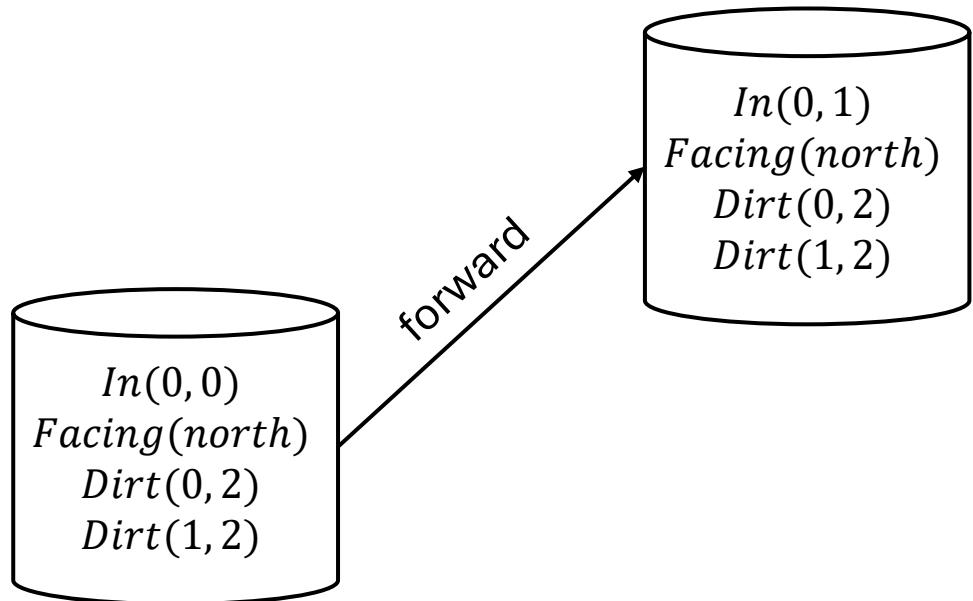
Possible Actions:

$$Ac = \{turn, forward, suck\}$$

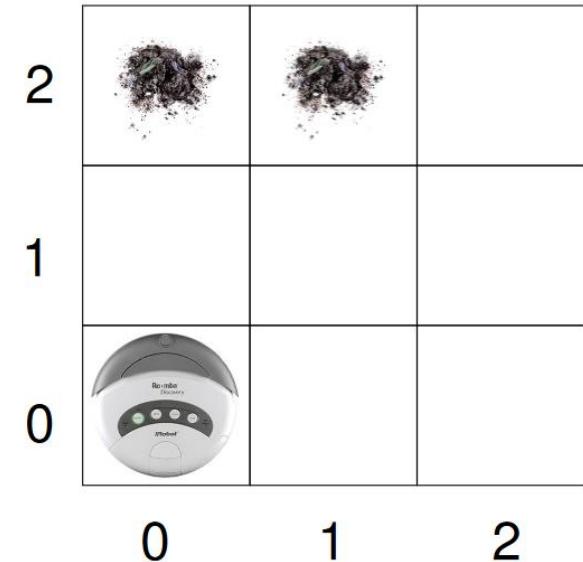
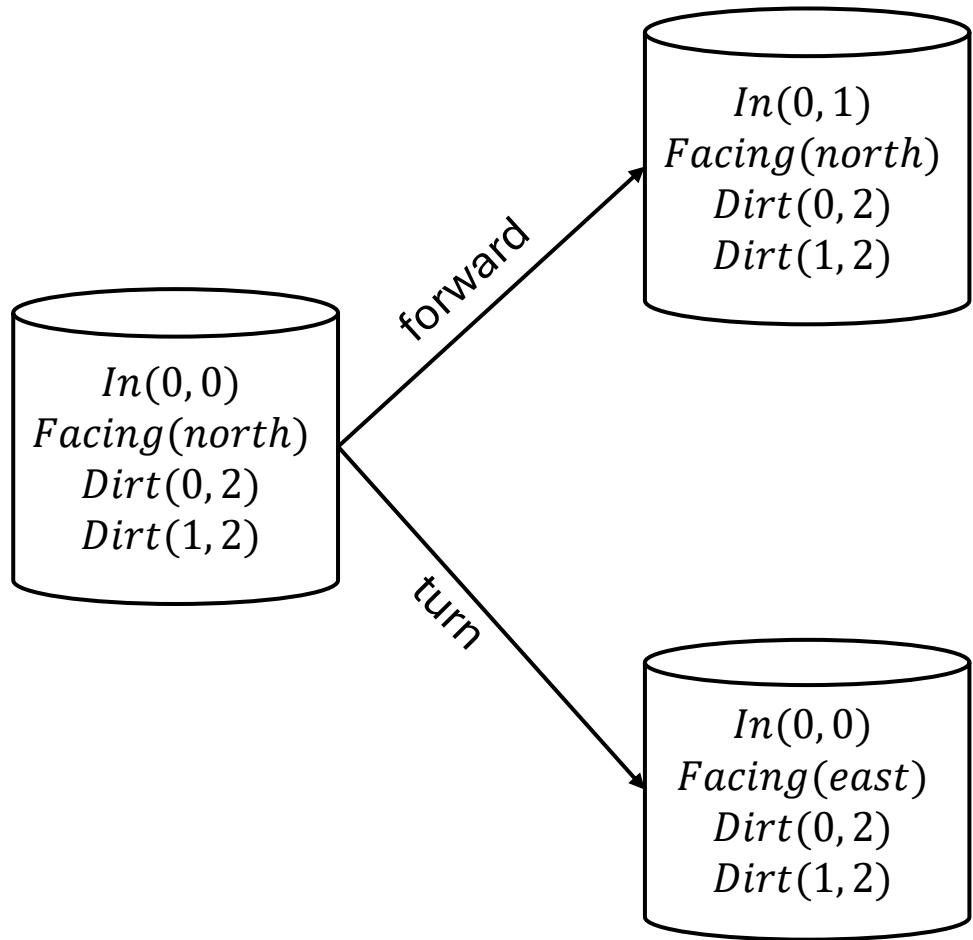
Note: *turn* means “*turn right*”



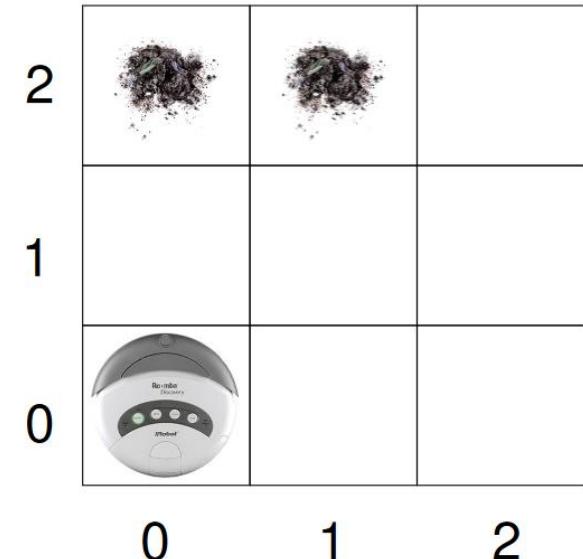
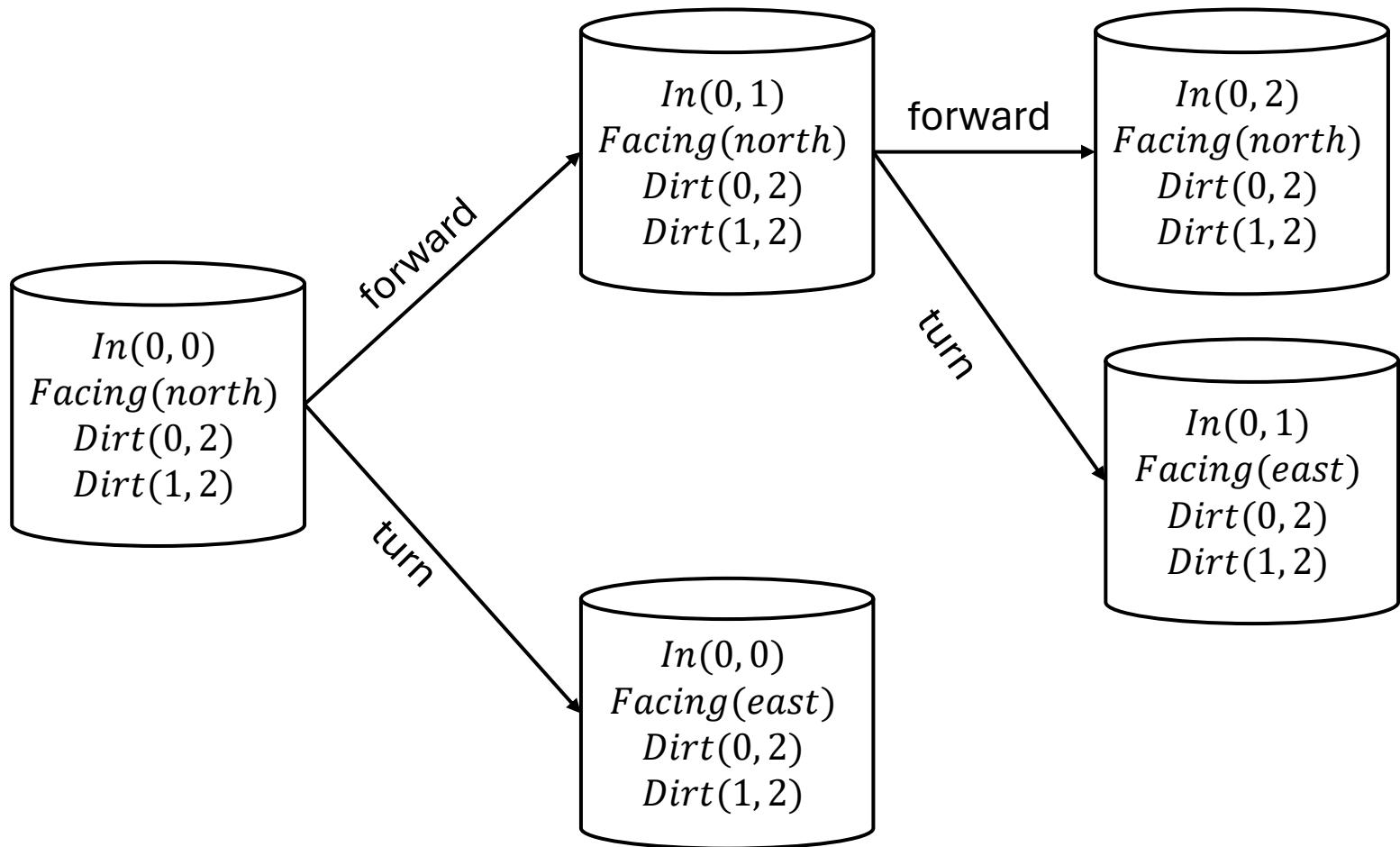
The Vacuum World



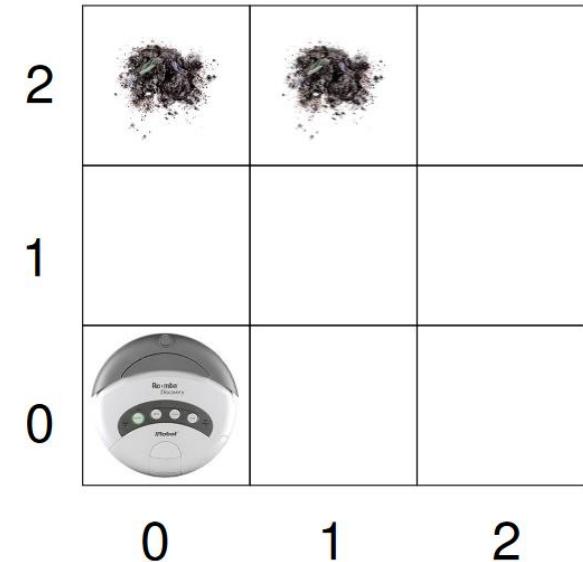
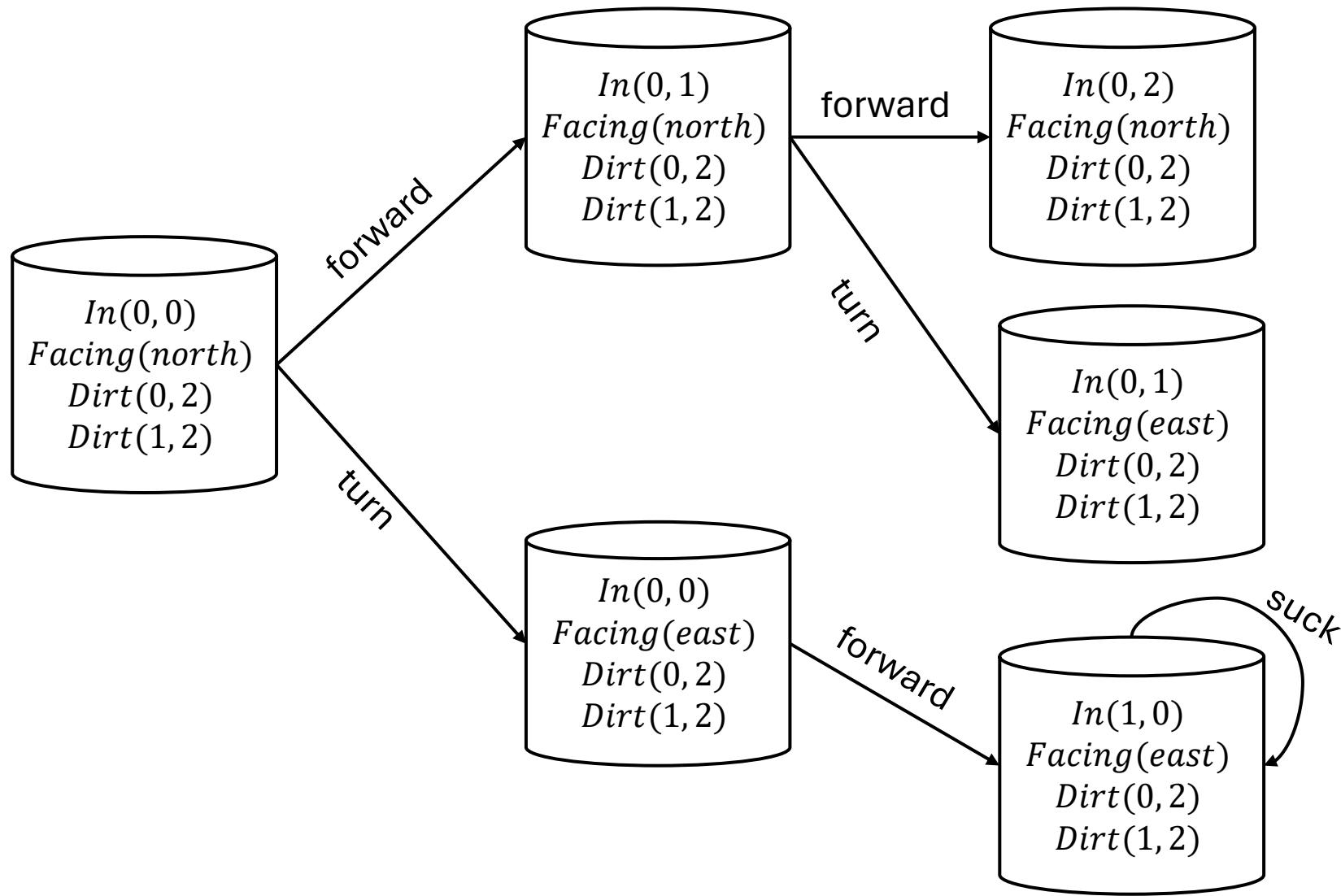
The Vacuum World



The Vacuum World

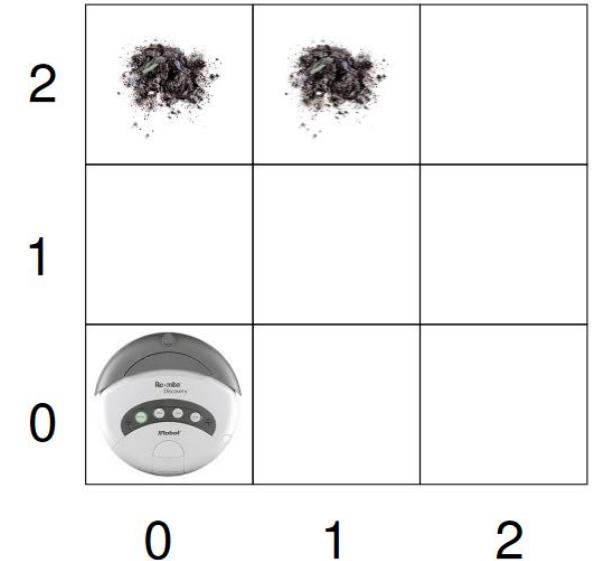


The Vacuum World



The Vacuum World

- Rules ρ for determining what to do:

$$\begin{aligned} In(0, 0) \wedge Facing(north) \wedge \neg Dirt(0, 0) &\rightarrow Do(forward) \\ In(0, 1) \wedge Facing(north) \wedge \neg Dirt(0, 1) &\rightarrow Do(forward) \\ In(0, 2) \wedge Facing(north) \wedge \neg Dirt(0, 2) &\rightarrow Do(forward) \\ In(0, 2) \wedge Facing(east) &\rightarrow Do(forward) \end{aligned}$$


- ... and so on!
- Using these rules (+ other obvious ones), starting at (0, 0) the robot will clear up dirt.

The Vacuum World

- Problems:
 - how to convert video camera input to $\text{Dirt}(0, 1)$?
 - decision making assumes a static environment (calculative rationality)

The Vacuum World

Microsoft Bing SEARCH CHAT

More Creative More Balanced More Precise

This is an image for the vacuum problem which can be represented using symbolic logic by defining some predicates and a database of facts based on the image.

List the facts from the image given the following predicates:

In(x,y); The vacuum is in (x, y) cell
Dirt(x, y); There is dirt in (x, y) cell
Adjacent(x, y); The vacuum is adjacent to the cell (x, y) cell.

2			
1			
0			
	0	1	2

Ask me anything... 0/4000 ➤

✓ Analysing the image: F

Based on the image, the following facts can be listed using the predicates:

- In(0,0); The vacuum is in the cell (0,0).
- Dirt(0,2); There is dirt in the cell (0,2).
- Dirt(1,2); There is dirt in the cell (1,2).
- Adjacent(0,1); The vacuum is adjacent to the cell (0,1).
- Adjacent(1,0); The vacuum is adjacent to the cell (1,0).

1 of 30 ●

What is symbolic logic?

Ask me anything... 0/4000 ➤

Agent-oriented programming

- Yoav Shoham introduced “agent-oriented programming” (AOP) in 1990

“... new programming paradigm, based on a societal view of computation ...”
- The key idea:
 - directly programming agents in terms of intentional notions
 - like belief, desire, and intention
 - Adopts the same abstraction as humans
 - Resulted in the Agent0 programming language

Agent0

- AGENT0 is implemented as an extension to LISP.
- Each agent in AGENT0 has 4 components:
 - a set of *capabilities* (things the agent can do);
 - a set of *initial beliefs*;
 - a set of *initial commitments* (things the agent will do); and
 - a set of *commitment rules*.
- The key component, which determines how the agent acts, is the commitment rule set.
 - Each commitment rule contains:
 - a message condition;
 - a mental condition; and
 - an action.

Agent0 Decision Cycle

On each decision cycle . . .

- The message condition is matched against the messages the agent has received;
- The mental condition is matched against the beliefs of the agent.
- If the rule fires, then the agent becomes committed to the action (the action gets added to the agents commitment set).

Actions may be . . .

- Private
 - An externally executed computation
- Communicative
 - Sending messages

Messages are constrained to be one of three types . . .

- requests
 - To commit to action
- unrequests
 - To refrain from action
- Informs
 - Which pass on information

Commitment Rules

- This rule may be paraphrased as follows:
 - if I receive a message from *agent* which requests me to do *action* at *time*, and I believe that:
 - *agent* is currently a friend;
 - I can do the *action*;
 - at *time*, I am not committed to doing any other *action*,
 - then commit to doing *action* at *time*.

```
1   COMMIT(  
2     (agent, REQUEST, DO(time, action)  
3     ), ;;; msg condition  
4     (B,  
5       [now, Friend agent] AND  
6       CAN(self, action) AND  
7       NOT [time, CMT(self, anyaction)]  
8     ), ;;; mental condition  
9     self,  
10    DO(time, action)  
11  )
```

Agent0

- Agent0 language can model *complex* agent behaviours.
- However, it is essentially a *prototype*, not intended for building large scale production systems.

Concurrent MetateM

- Concurrent METATEM is a multi-agent language, developed by [Michael Fisher](#)
 - Each agent is programmed by giving it a temporal logic specification of the behaviour it should exhibit.
 - These specifications are executed directly in order to generate the behaviour of the agent.
- Temporal logic is classical logic augmented by modal operators for describing how the truth of propositions changes over time.
 - Think of the world as being a number of discrete states.
 - There is a single past history, but a number of possible futures

MetateM Agents

- A Concurrent MetateM system contains a number of agents (objects)
 - Each object has 3 attributes:
 - A name
 - An interface
 - A MetateM program
 - An agent's interface contains two sets:
 - messages the agent will accept;
 - messages the agent may send.

MetateM

- The root of the MetateM concept is Gabbay’s separation theorem:
 - Any arbitrary temporal logic formula can be rewritten in a logically equivalent past \Rightarrow future form
- Execution proceeds by a process of continually matching rules against a “history”, and firing those rules whose antecedents are satisfied.
 - The instantiated future-time consequents become commitments which must subsequently be satisfied.

Examples

$\Box \text{important}(\text{agents})$	<i>means “it is now, and will always be true that agents are important”</i>
$\Diamond \text{important}(\text{ConcurrentMetateM})$	<i>means “sometime in the future, ConcurrentMetateM will be important”</i>
$\lozenge \text{ important}(\text{Prolog})$	<i>means “sometime in the past it was true that Prolog was important”</i>
$(\neg \text{friends}(\text{us})) \mathcal{U} \text{apologise}(\text{you})$	<i>means “we are not friends until you apologise”</i>
$\bigcirc \text{apologise}(\text{you})$	<i>means “tomorrow (in the next state), you apologise”</i>
$\bullet \text{apologise}(\text{you}) \Rightarrow \bigcirc \text{friends}(\text{us})$	<i>means “if you apologised yesterday, then tomorrow we will be friends”</i>
$\text{friends}(\text{us}) \mathcal{S} \text{apologise}(\text{you})$	<i>means “we have been friends since you apologised”</i>

Readings for this week

- M.Wooldridge: An introduction to MultiAgent Systems – Ch. 3
Deductive Reasoning Agents
- “[Agent Oriented Programming](#)”, Yoav Shoham

LECTURE 3b: Practical Reasoning Agents

Introduction to Multi-Agent Systems (MESIIA, MIA)
URV

Classes of Architecture

- 1956–present: Symbolic Reasoning Agents
 - Agents make decisions about what to do via symbol manipulation.
 - Its purest expression, proposes that agents use explicit logical reasoning in order to decide what to do.
- 1985–present: Reactive Agents
 - Problems with symbolic reasoning led to a reaction against this
 - led to the reactive agents movement
- 1990–present: Hybrid Agents
 - Hybrid architectures attempt to combine the best of reasoning and reactive architectures

What is Practical Reasoning?

- Practical reasoning is reasoning directed towards actions — the process of figuring out what to do:

Bratman (1990)

“... Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes...”

- Distinguish practical reasoning from *theoretical reasoning*
 - Theoretical reasoning is directed towards *beliefs*.

The components of Practical Reasoning

- Human practical reasoning consists of two activities:
 - Deliberation:
 - Deciding *what* state of affairs we want to achieve
 - The output of deliberation is *intentions*;
 - Means-ends reasoning
 - Deciding *how* to achieve these state of affairs
 - The outputs of this step are *plans*
- Intentions are a key part of this.
 - The interaction between *beliefs*, *desires* and *intentions* defines how the model works.

Intentions in Practical Reasoning

- Future-directed intentions
 - Intentions that an agent has towards some future state affairs.
- Intentions are:
 - *Pro-attitudes*, they tend to lead to action (drive means-ends reasoning).
 - *Persist*, attempt to achieve it (but not for too long).
 - *Consistent*/ constrain the future deliberation while holding some particular intentions, no options that are inconsistent with the intentions can be considered.
 - Influence beliefs upon which future practical reasoning is based.

Intentions in Practical Reasoning

1. Intentions pose problems for agents, who need to determine ways of achieving them. Type equation here.

If I have an intention to ϕ , you would expect me to devote resources to deciding how to bring about ϕ

2. Intentions provide a “filter” for adopting other intentions, which must not conflict.

If I have an intention to ϕ , you would not expect me to adopt an intention ψ that was incompatible with ϕ .

3. Agents track the success of their intentions, and tend to try again if their attempts fail.

If an agent’s first attempt to achieve ϕ fails, then all other things being equal, it will try an alternative plan to achieve ϕ .

Intentions in Practical Reasoning

4. Agents believe their intentions are possible.

That is, they believe there is at least some way that the intentions could be brought about.

5. Agents do not believe they will not bring about their intentions.

It would not be rational of me to adopt an intention to ϕ if I believed I would fail with ϕ .

6. Under certain circumstances, agents believe they will bring about their intentions.

If I intend ϕ , then I believe that under “normal circumstances” I will succeed with ϕ .

Intentions in Practical Reasoning

7. Agents need not intend all the expected side effects of their intentions.

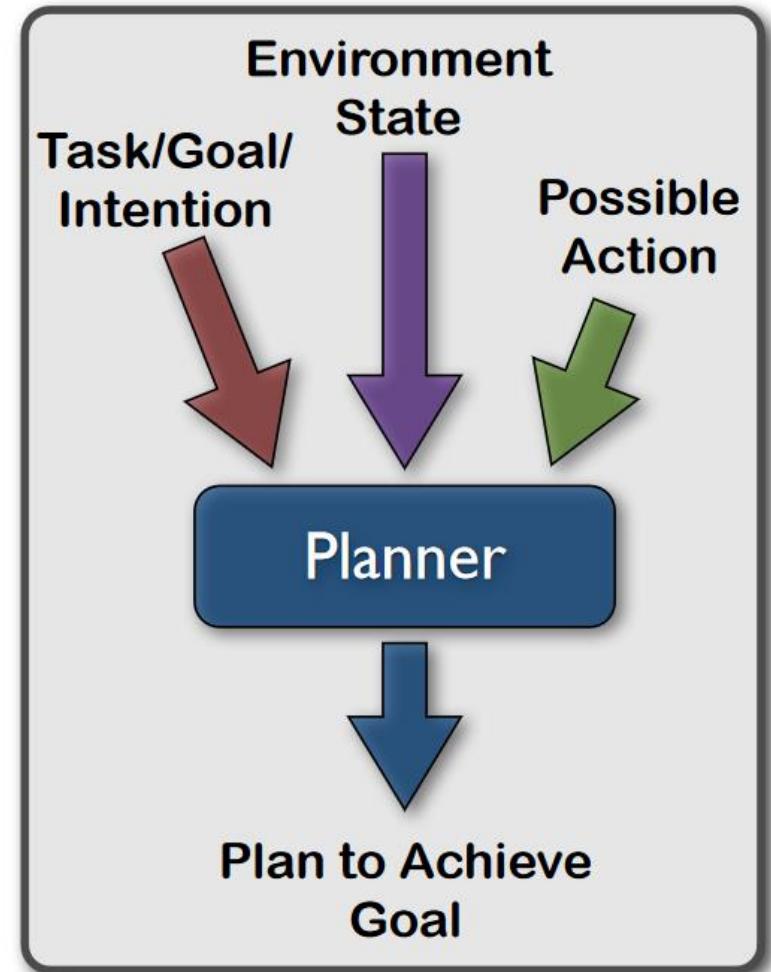
If I believe $\phi \Rightarrow \psi$ and I intend that ϕ , I do not necessarily intend ψ also.

8. Intentions are not closed under implication.

I may believe that going to the dentist involves pain, and I may also intend to go to the dentist — but this does not imply that I intend to suffer pain!

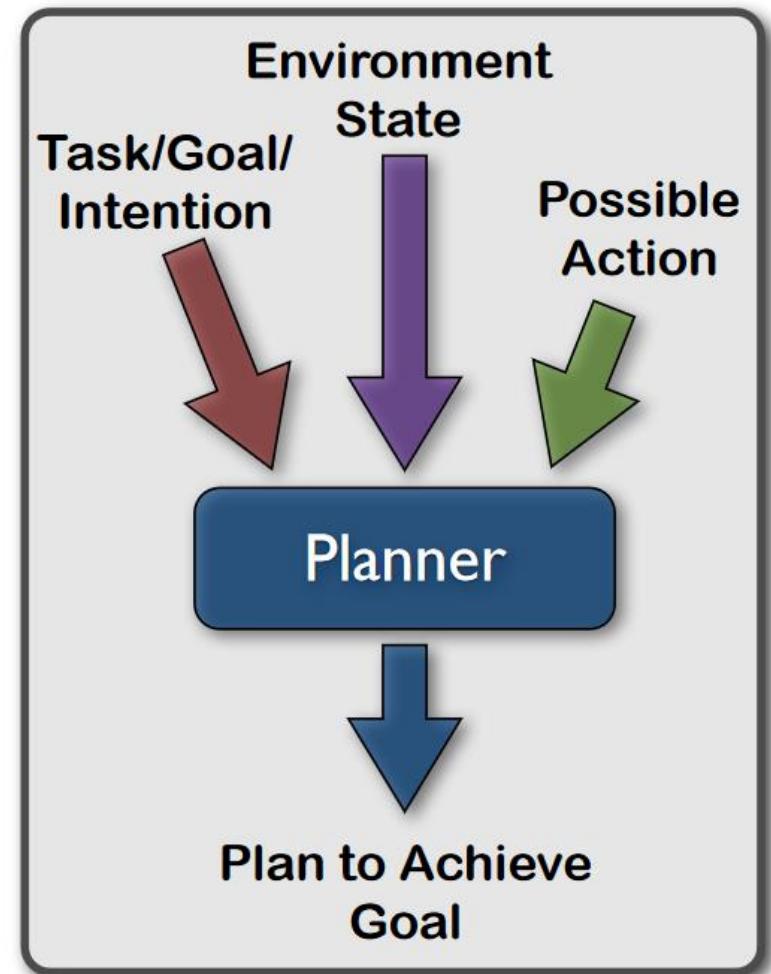
Means-ends Reasoning/Planning

- Planning is the design of a course of action that will achieve some desired goal.
 - Basic idea is to give a planning system:
 - (representation of) goal/intention to achieve;
 - (representation of) actions it can perform;
 - (representation of) the environment.
 - and have it generate a *plan* to achieve the goal.
- This is ***automatic programming***.



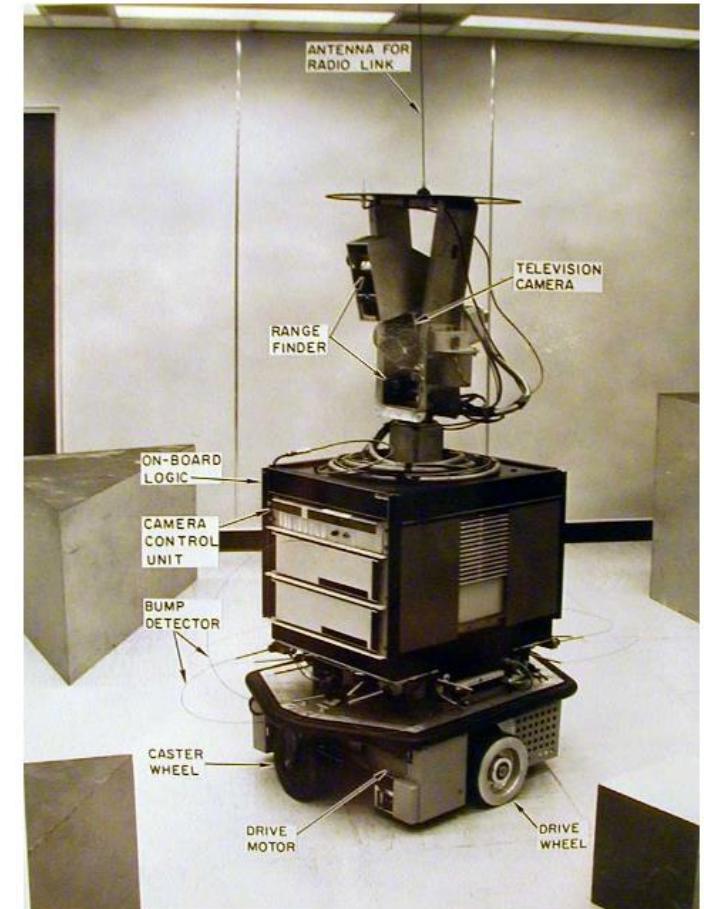
Means-ends Reasoning/Planning

- Don't have to directly tell the system what to do!
 - Let it *figure out* how to achieve the goal on its own.



STRIPS Planner

- The Stanford Research Institute Problem Solver
- Used by Shakey, the robot developed by Richard Fikes and Nils Nilsson in 1971 at SRI International.

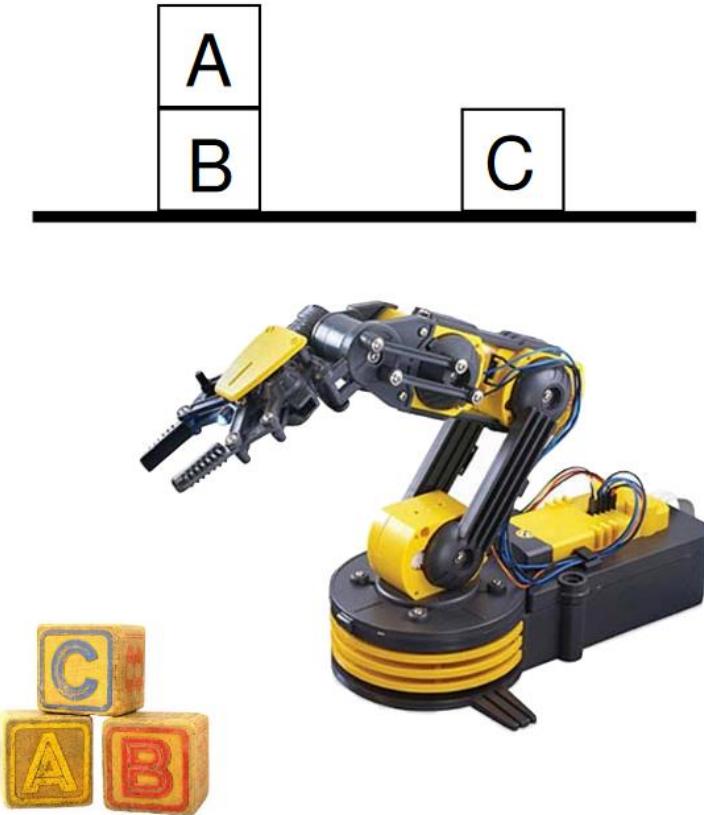


Representations

- Question: How do we represent . . .
 - *goal to be achieved;*
 - *state of environment;*
 - *actions available to agent;*
 - *plan itself.*
- Answer: We use *logic*, or something that looks a lot like logic.

Blocksworld

- We'll illustrate the techniques with reference to the blocks world.
 - A simple (toy) world, in this case one where we consider toys.
- The blocks world contains a robot arm, 3 blocks (A, B and C) of equal size, and a tabletop.
- The aim is to generate a plan for the robot arm to build towers out of blocks.



Blocksworld

- The environment is represented by an *ontology*.
- The closed world assumption is used
 - Anything not stated is assumed to be false.
- A *goal* is represented as a set of formulae

Blocksworld Ontology

$On(x,y)$	object x on top of object y
$OnTable(x)$	object x is on the table
$Clear(x)$	nothing is on top of object x
$Holding(x)$	arm is holding x

Representation of the following blocks

$Clear(A)$
 $On(A, B)$
 $OnTable(B)$
 $Clear(C)$
 $OnTable(C)$
 $ArmEmpty$

The goal:

{ $OnTable(A)$, $OnTable(B)$, $OnTable(C)$, $ArmEmpty$ }

Blocksworld Actions

- Each action has:
 - a *name*: which may have arguments
 - a *pre-condition list*: list of facts which must be true for action to be executed;
 - a *delete list*: list of facts that are no longer true after action is performed;
 - an *add list*: list of facts made true by executing the action.
- Each of these may contain variables
- What is a plan?
 - A sequence (list) of actions, with *variables replaced by constants*.

Blocksworld Actions

$Stack(x, y)$
pre $Clear(y) \wedge Holding(x)$
del $Clear(y) \wedge Holding(x)$
add $ArmEmpty \wedge On(x, y)$

The **stack** action occurs when the robot arm places the object x it is holding is placed on top of object y .

$Pickup(x)$
pre $Clear(x) \wedge OnTable(x) \wedge ArmEmpty$
del $OnTable(x) \wedge ArmEmpty$
add $Holding(x)$

The **pickup** action occurs when the arm picks up an object x from the table.

$UnStack(x, y)$
pre $On(x, y) \wedge Clear(x) \wedge ArmEmpty$
del $On(x, y) \wedge ArmEmpty$
add $Holding(x) \wedge Clear(y)$

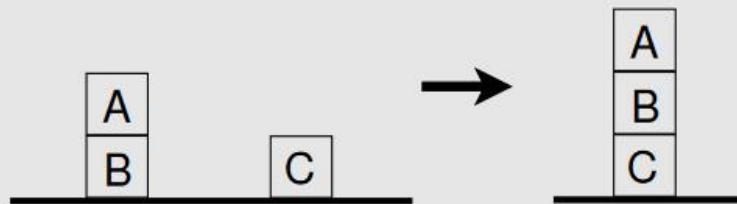
The **unstack** action occurs when the robot arm picks an object y up from on top of another object y .

$PutDown(x)$
pre $Holding(x)$
del $Holding(x)$
add $OnTable(x) \wedge ArmEmpty \wedge Clear(x)$

The **putdown** action occurs when the arm places the object x onto the table.

Using Plans

To get from here (left) to here (right)...



...we need this set of actions:

UnStack(A,B)

Putdown(A)

Pickup(B)

Stack(B, C)

Pickup(A)

Stack(A, B)

Stack(x,y)

pre *Clear(y) ∧ Holding(x)*

del *Clear(y) ∧ Holding(x)*

add *ArmEmpty ∧ On(x,y)*

UnStack(x,y)

pre *On(x,y) ∧ Clear(x) ∧ ArmEmpty*

del *On(x,y) ∧ ArmEmpty*

add *Holding(x) ∧ Clear(y)*

Pickup(x)

pre *Clear(x) ∧ OnTable(x) ∧ ArmEmpty*

del *OnTable(x) ∧ ArmEmpty*

add *Holding(x)*

PutDown(x)

pre *Holding(x)*

del *Holding(x)*

add *OnTable(x) ∧ ArmEmpty ∧ Clear(x)*



Plan Validity

- Thus, a plan is simply a sequence of steps
- However, how can we:
 - Generate the plan?
 - Ensure that it is correct?

Formal Representation

- As before we assume that the agent has a set of actions Ac , and we will write individual actions as α_1, α_2 and so on.
- Now, we define a new component called action *descriptor*.

Formal Representation

- A descriptor for an action $\alpha \in Ac$ is a triple $\langle P_\alpha, D_\alpha, A_\alpha \rangle$ where
 - P_α is the set of formulae of first-order logic that describes the *preconditions* of action α .
 - D_α is the set of formulae of first-order logic that represents those facts will be *false* by applying the action α (the *delete* list).
 - A_α is the set of formulae of first-order logic that represents those facts will be *true* by applying the action α (the *add* list).
- For simplicity, we assume these lists constrained to only contain *ground atoms*.

Formal Representation

- A *planning problem* is determined by a triple $\langle \Delta, O, \gamma \rangle$ where
 - Δ is the beliefs of the agent about the *initial* state of the world.
 - $O = \{\langle P_\alpha, D_\alpha, A_\alpha \rangle \mid \alpha \in Ac\}$ is an indexed set of operator descriptors, one for each available action \alpha.
 - γ is a set of formulae of first-order logic, representing the *goal/task/intention* to be achieved.
- So, a *plan* π is a sequence of actions $\pi = (\alpha_1, \alpha_2, \dots, \alpha_n)$ where α_i is a member of Ac .

Formal Representation

- With respect to a planning problem $\langle \Delta, 0, \gamma \rangle$ a plan $\pi = (\alpha_1, \alpha_2, \dots, \alpha_n)$ determines a sequence of $n + 1$ belief databases $\Delta_0, \Delta_1, \dots, \Delta_n$, where $\Delta_0 = \Delta$, and

$$\Delta_i = (\Delta_{i-1} \setminus D_{\alpha_i}) \cup A_{\alpha_i} \text{ for } 1 \leq i \leq n$$

Formal Representation

- Acceptable Plan:

A (linear) plan $\pi = (\alpha_1, \alpha_2, \dots, \alpha_n)$ is said to be acceptable w.r.t the problem $\langle \Delta, 0, \gamma \rangle$ iff the preconditions of every action is satisfied in preceding belief database, i.e., if $\Delta_{i-1} \models P_{\alpha_i}$, for all $1 \leq i \leq n$.

- Correct Plan:

A plan $\pi = (\alpha_1, \alpha_2, \dots, \alpha_n)$ is correct w.r.t $\langle \Delta, 0, \gamma \rangle$ iff:

1. It is acceptable
2. $\Delta_n \models \gamma$ (i.e., if the goal is achieved in the final belief database generated by π)

Action Definitions are important!!!

Stack(x, y)	
pre	$Clear(y) \& Holding(x)$
del	$Clear(y) \& Holding(x)$
add	$ArmEmpty \& On(x, y)$

UnStack(x, y)	
pre	$On(x, y) \& Clear(x) \& ArmEmpty$
del	$On(x, y) \& ArmEmpty$
add	$Holding(x) \& Clear(y)$

Pickup(x)	
pre	$Clear(x) \& OnTable(x) \& ArmEmpty$
del	$OnTable(x) \& ArmEmpty$
add	$Holding(x)$

Release(x)	
pre	$Holding(x)$
del	$Holding(x)$
add	$OnTable(x) \& ArmEmpty$

One of these definitions works. The other doesn't!

Grap(x)	
pre	$Clear(x) \& OnTable(x) \& ArmEmpty$
del	$OnTable(x) \& ArmEmpty$
add	$Holding(x)$

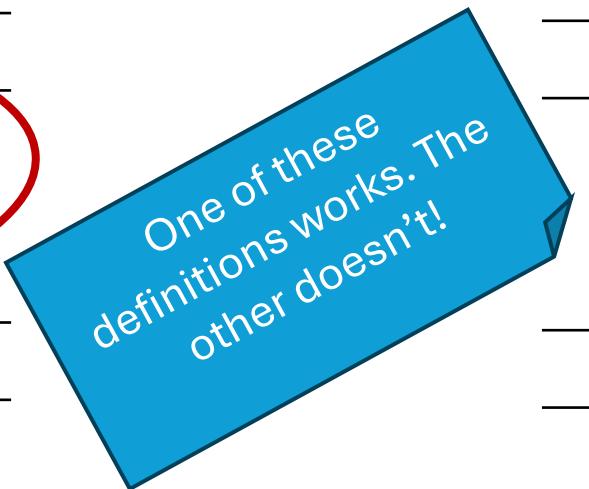
Build(x, y)	
pre	$Clear(y) \& Holding(x)$
del	$Clear(y) \& Holding(x)$
add	$ArmEmpty \& On(x, y)$

Drop(y)	
pre	$Holding(y)$
del	$Holding(y)$
add	$OnTable(y) \& ArmEmpty \& Clear(y)$

Demolish(x, y)	
pre	$On(x, y) \& Clear(y) \& ArmEmpty$
del	$On(x, y) \& ArmEmpty$
add	$Holding(x) \& Clear(y)$

Action Definitions are important!!!

Stack(x, y)	
pre	$Clear(y) \& Holding(x)$
del	$Clear(y) \& Holding(x)$
add	$ArmEmpty \& On(x, y)$
UnStack(x, y)	
pre	$On(x, y) \& Clear(x) \& ArmEmpty$
del	$On(x, y) \& ArmEmpty$
add	$Holding(x) \& Clear(y)$
Pickup(x)	
pre	$Clear(x) \& OnTable(x) \& ArmEmpty$
del	$OnTable(x) \& ArmEmpty$
add	$Holding(x)$
Release(x)	
pre	$Holding(x)$
del	$Holding(x)$
add	$OnTable(x) \& ArmEmpty \& Clear(x)$



Grap(x)	
pre	$Clear(x) \& OnTable(x) \& ArmEmpty$
del	$OnTable(x) \& ArmEmpty$
add	$Holding(x)$
Build(x, y)	
pre	$Clear(y) \& Holding(x)$
del	$Clear(y) \& Holding(x)$
add	$ArmEmpty \& On(x, y)$
Drop(y)	
pre	$Holding(y)$
del	$Holding(y)$
add	$OnTable(y) \& ArmEmpty \& Clear(y)$
Demolish(x, y)	
pre	$On(x, y) \& Clear(y) \& ArmEmpty$
del	$On(x, y) \& ArmEmpty$
add	$Holding(x) \& Clear(y)$

Implementing Practical Reasoning Agents

- A first pass at an implementation of a practical reasoning agent:
- For now, we will not be concerned with stages 2 or 3.
 - These are related to the functions **see** and **next**

Agent Control Loop Version 1

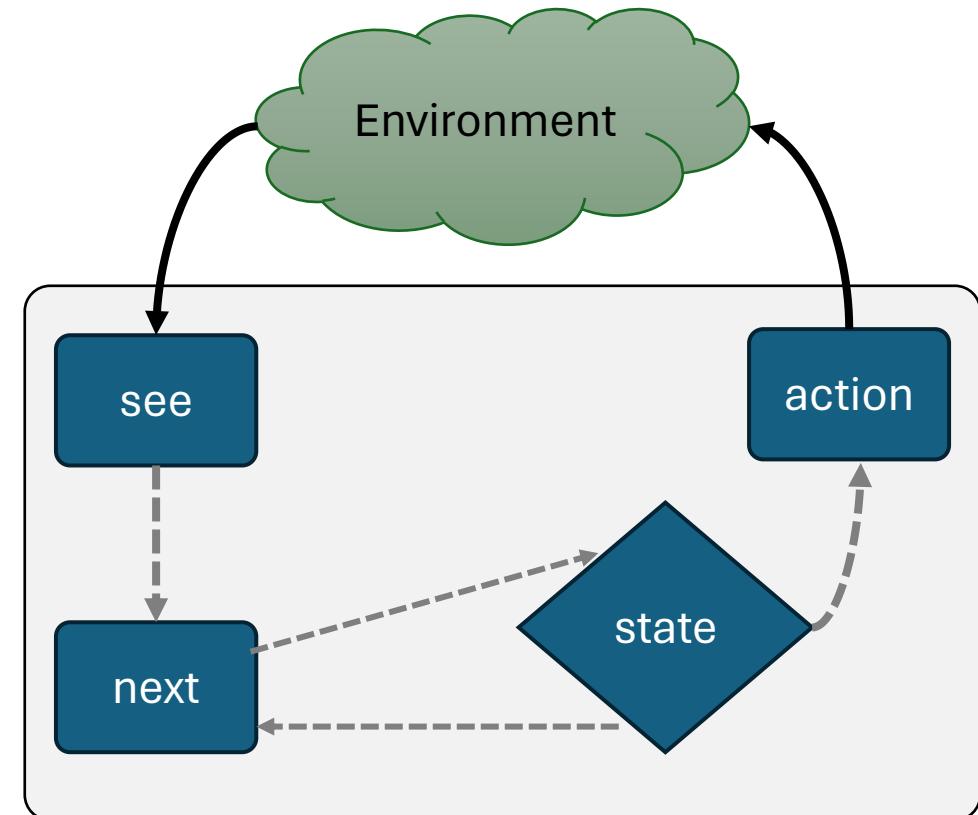
1. while true
2. observe the world;
3. update internal world model;
4. deliberate about what intention
 to achieve next;
5. use means-ends reasoning to get
 a plan for the intention;
6. execute the plan
7. end while

Implementing Practical Reasoning Agents

- see is as before
 $see: E \rightarrow Per$
- Instead of the function next...
 - which takes a percept and used it to update the internal state of an agent
- ...we have a belief revision function:

$$brf: 2^{Bel} \times Per \rightarrow 2^{Bel}$$

Where, Bel is the set of all possible beliefs that an agent might have



Implementing Practical Reasoning Agents

- Problem:
 - deliberation and means-ends reasoning processes are not *instantaneous*.
 - They have a *time cost*.
- Suppose that deliberation is *optimal*
 - The agent selects the optimal intention to achieve, then this is the best thing for the agent.
 - i.e., it maximises the expected utility.
- So, the agent selects an intention to achieve that would have been optimal at the time it observed the world.
 - This is calculative rationality.
- The *world may change* in the meantime.
 - Even if the agent can compute the right thing to do, it may not do the right thing.
 - Optimality is hard.

Implementing Practical Reasoning Agents

- Let's make the algorithm more formal
 - The term $I \subseteq Int$, the set of intentions
 - $\text{Plan}()$ is the planning function,
 - $\text{brf}()$ is the belief revision function
 - and $\text{execute}()$ is a function that executes each action in a plan.
- How might we implement these functions?

Agent Control Loop Version 2

```
1.  $B := B_0; /* initial beliefs */$ 
2. while true do
3.   get next percept  $\rho$ ;
4.    $B := \text{brf}(B, \rho);$ 
5.    $I := \text{deliberate}(B);$ 
6.    $\pi := \text{plan}(B, I);$ 
7.   execute( $\pi$ )
8. end while
```

Deliberation

- How does an agent *deliberate*?
 - begin by trying to understand what the *options* available to it are;
 - *choose* between them, and *commit* to some.
- Chosen options are then *intentions*.
- The *deliberate* function can be decomposed into two distinct functional components:
 - *option generation*; and
 - *filtering*

Option Generation and Filtering

- Option Generation
 - The agent generates a set of possible alternatives
 - Represent option generation via a function, **options()**, which takes the agent's current beliefs and current intentions, and from them determines a set of options
 - **desires**
- Filtering
 - the agent chooses between competing alternatives, and commits to achieving them.
 - In order to select between competing options, an agent uses a **filter()** function.
 - **Intentions**

$$filter : \mathcal{P}(Bel) \times \mathcal{P}(Des) \times \mathcal{P}(Int) \rightarrow \mathcal{P}(Int)$$

options : $\mathcal{P}(Bel) \times \mathcal{P}(Int) \rightarrow \mathcal{P}(Des)$

Implementing Practical Reasoning Agents

Agent Control Loop Version 3

1. $B := B_0;$
2. $I := I_0;$
3. **while** true **do**
4. get next percept ρ ;
5. $B := brf(B, \rho);$
6. $D := options(B, I);$
7. $I := filter(B, D, I);$
8. $\pi := plan(B, I);$
9. $execute(\pi)$
10. **end while**

Degrees of Commitment

- Blind commitment
 - A blindly committed agent will continue to maintain an intention until it believes the intention has actually been achieved. Blind commitment is also sometimes referred to as *fanatical* commitment.
- Single-minded commitment
 - A single-minded agent will continue to maintain an intention *until* it believes that *either* the intention has been achieved, *or else that it is no longer possible to achieve* the intention.
- Open-minded commitment
 - An open-minded agent will maintain an intention *as long as it is still believed* possible.

Degrees of Commitment

- An agent has commitment both to:
 - *ends* (i.e., the state of affairs it wishes to bring about), and
 - *means* (i.e., the mechanism via which the agent wishes to achieve the state of affairs).
- Currently, our agent control loop is **overcommitted**, both to means and ends.
 - Modification: *replan* if ever a plan goes wrong.
 - However, to write the algorithm down we *need to refine* our notion of plan execution.

Degrees of Commitment

- The previous version was *blindly committed to its means and its ends*
- If π is a plan, then:
 - $\text{empty}(\pi)$ is true if there are no more actions in the plan.
 - $\text{hd}(\pi)$ returns the first action in the plan.
 - $\text{tail}(\pi)$ returns the plan minus the head of the plan.
 - $\text{sound}(\pi, I, B)$ means that π is sound plan for I given B .

Agent Control Loop Version 4

```
1.    $B := B_0;$ 
2.    $I := I_0;$ 
3.   while true do
4.       get next percept  $\rho$ ;
5.        $B := \text{brf}(B, \rho);$ 
6.        $D := \text{options}(B, I);$ 
7.        $I := \text{filter}(B, D, I);$ 
8.        $\pi := \text{plan}(B, I);$ 
9.       while not  $\text{empty}(\pi)$  do
10.           $\alpha := \text{hd}(\pi);$ 
11.           $\text{execute}(\alpha);$ 
12.           $\pi := \text{tail}(\pi);$ 
13.          get next percept  $\rho$ ;
14.           $B := \text{brf}(B, \rho);$ 
15.          if not  $\text{sound}(\pi, I, B)$  then
16.               $\pi := \text{plan}(B, I)$ 
17.          end-if
18.      end-while
19.  end-while
```

Degrees of Commitment

- Makes the control loop more reactive, able to change intention when the world changes.
 - i.e. it is not committed to its means (line 16)
- Still overcommitted to intentions (ends).
 - Never stops to consider whether or not its intentions are appropriate.

Agent Control Loop Version 4

```
1.    $B := B_0;$ 
2.    $I := I_0;$ 
3.   while true do
4.       get next percept  $\rho$ ;
5.        $B := brf(B, \rho);$ 
6.        $D := options(B, I);$ 
7.        $I := filter(B, D, I);$ 
8.        $\pi := plan(B, I);$ 
9.       while not empty( $\pi$ ) do
10.           $\alpha := hd(\pi);$ 
11.          execute( $\alpha$ );
12.           $\pi := tail(\pi);$ 
13.          get next percept  $\rho$ ;
14.           $B := brf(B, \rho);$ 
15.          if not sound( $\pi, I, B$ ) then
16.               $\pi := plan(B, I)$ 
17.          end-if
18.      end-while
19.  end-while
```

Single Minded Commitment

- Modification:
 - stop to determine whether intentions have succeeded or whether they are impossible
- Our agent now gets to reconsider its intentions once every time around the outer control loop (line 9), i.e., after:
 - it has completely executed a plan to achieve its current intentions; or
 - it believes it has achieved its current intentions; or
 - it believes its current intentions are no longer possible

```

1.    $B := B_0;$ 
2.    $I := I_0;$ 
3.   while true do
4.       get next percept  $\rho$ ;
5.        $B := brf(B, \rho);$ 
6.        $D := options(B, I);$ 
7.        $I := filter(B, D, I);$ 
8.        $\pi := plan(B, I);$ 
9.       while not(empty( $\pi$ )
           or succeeded( $I, B$ )
           or impossible( $I, B$ )) do
10.           $\alpha := hd(\pi);$ 
11.          execute( $\alpha$ );
12.           $\pi := tail(\pi);$ 
13.          get next percept  $\rho$ ;
14.           $B := brf(B, \rho);$ 
15.          if not sound( $\pi, I, B$ ) then
16.               $\pi := plan(B, I)$ 
17.          end-if
18.      end-while
19.  end-while

```

Open Minded Commitment

- In the previous version, our agent reconsiders its intentions once every time around the outer control loop.
- In this new version, our agent also reconsiders its intentions after every action (lines 15 & 16)
- But this intention reconsideration is **costly!**

```

1.    $B := B_0;$ 
2.    $I := I_0;$ 
3.   while true do
4.       get next percept  $\rho$ ;
5.        $B := brf(B, \rho);$ 
6.        $D := options(B, I);$ 
7.        $I := filter(B, D, I);$ 
8.        $\pi := plan(B, I);$ 
9.       while not (empty( $\pi$ )
           or succeeded( $I, B$ )
           or impossible( $I, B$ )) do
10.           $\alpha := hd(\pi);$ 
11.          execute( $\alpha$ );
12.           $\pi := tail(\pi);$ 
13.          get next percept  $\rho$ ;
14.           $B := brf(B, \rho);$ 
15.           $D := options(B, I);$ 
16.           $I := filter(B, D, I);$ 
17.          if not sound( $\pi, I, B$ ) then
18.               $\pi := plan(B, I)$ 
19.          end-if
20.      end-while
21.  end-while

```

Intention Reconsideration

- A dilemma:
 - an agent that *does not stop* to reconsider its intentions sufficiently often *will continue to attempt* to achieve its intentions even after it is clear that they cannot be achieved, or that there is no longer any reason for achieving them;
 - an agent that *constantly* reconsiders its attentions may *spend insufficient time* actually working to achieve them, and hence runs the risk of never actually achieving them.
- Solution: incorporate an explicit meta-level control component, that decides whether or not to *reconsider*.

```

1.    $B := B_0;$ 
2.    $I := I_0;$ 
3.   while true do
4.     get next percept  $\rho$ ;
5.      $B := brf(B, \rho);$ 
6.      $D := options(B, I);$ 
7.      $I := filter(B, D, I);$ 
8.      $\pi := plan(B, I);$ 
9.     while not(empty( $\pi$ )
          or succeeded( $I, B$ )
          or impossible( $I, B$ )) do
10.        $\alpha := hd(\pi);$ 
11.       execute( $\alpha$ );
12.        $\pi := tail(\pi);$ 
13.       get next percept  $\rho$ ;
14.        $B := brf(B, \rho);$ 
15.       if reconsider( $I, B$ ) then
16.          $D := options(B, I);$ 
17.          $I := filter(B, D, I);$ 
18.       end-if
19.       if not sound( $\pi, I, B$ ) then
20.          $\pi := plan(B, I)$ 
21.       end-if
22.     end-while
23.   end-while

```

Intention Reconsideration

- The possible interactions between meta-level control and deliberation are:

Situation number	Chose to deliberate?	Changed intentions?	Would have changed intentions?	<i>reconsider(...)</i> optimal?
1	No	—	No	Yes
2	No	—	Yes	No
3	Yes	No	—	No
4	Yes	Yes	—	Yes

- An important assumption: cost of *reconsider(...)* is ***much*** less than the cost of the deliberation process itself.

Optimal Intention Reconsideration

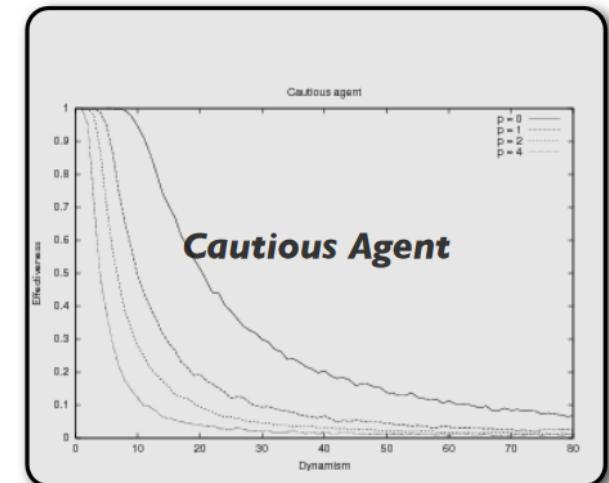
- Two different types of reconsideration strategy considered:
 - bold agents: never pause to reconsider intentions, and
 - cautious agents: stop to reconsider after every action.
- Dynamism in the environment is represented by the rate of world change, γ .

Optimal Intention Reconsideration

- if γ is low (i.e., the environment does not change quickly), then bold agents do well compared to cautious ones.
 - This is because cautious ones waste time reconsidering their commitments while bold agents are busy working towards — and achieving — their intentions.
- If γ is high (i.e., the environment changes frequently), then cautious agents can outperform bold agents.
 - This is because they are able to recognise when intentions are doomed, and also to take advantage of serendipitous situations and new opportunities when they arise.
- When planning costs are high, this advantage can be eroded.

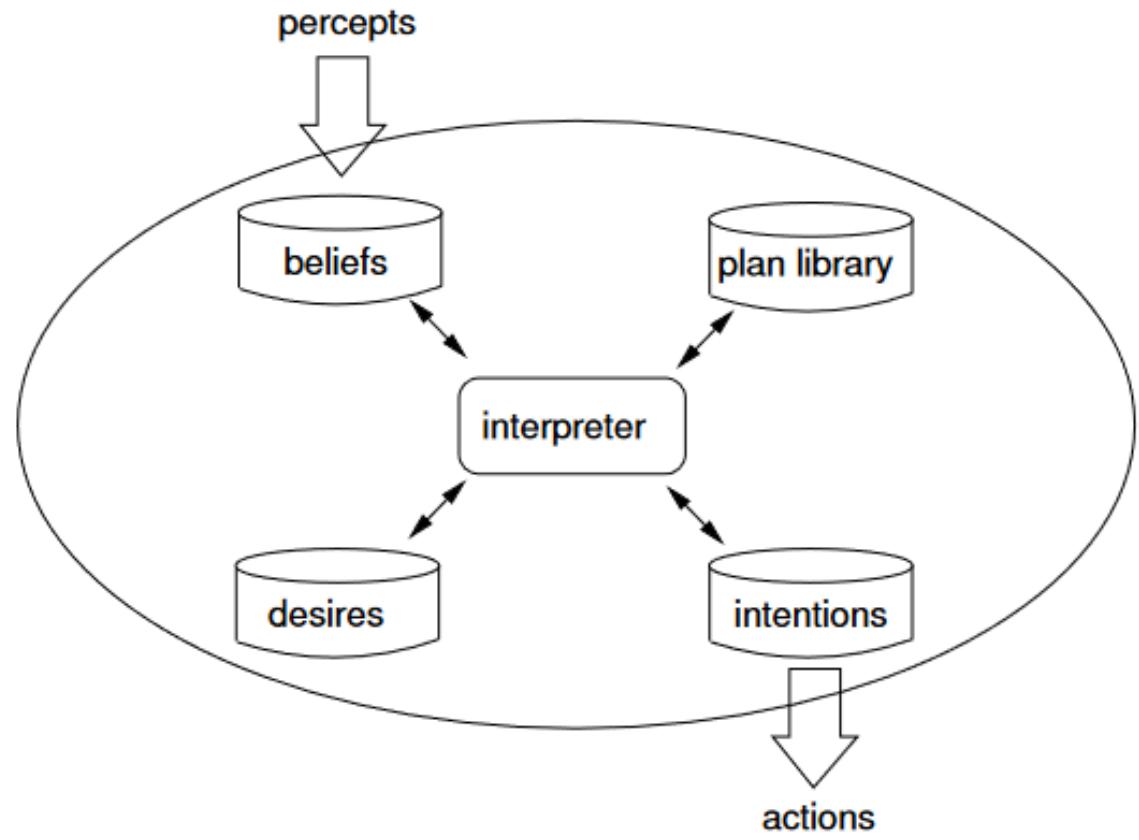


low ← γ → high



Implemented BDI Agents: Procedural Reasoning System (PRS)

- In the PRS, each agent is equipped with a plan library, representing that agent's procedural knowledge: knowledge about the mechanisms that can be used by the agent in order to realise its intentions.
- The options available to an agent are directly determined by the plans an agent has: an agent with no plans has no options.



Readings for this week

- M.Wooldridge: An introduction to MultiAgent Systems – Ch. 4
Practical Reasoning Agents

LECTURE 4: Reactive and Hybrid Agents

Introduction to Multi-Agent Systems (MESIIA, MIA)
URV

Classes of Architecture

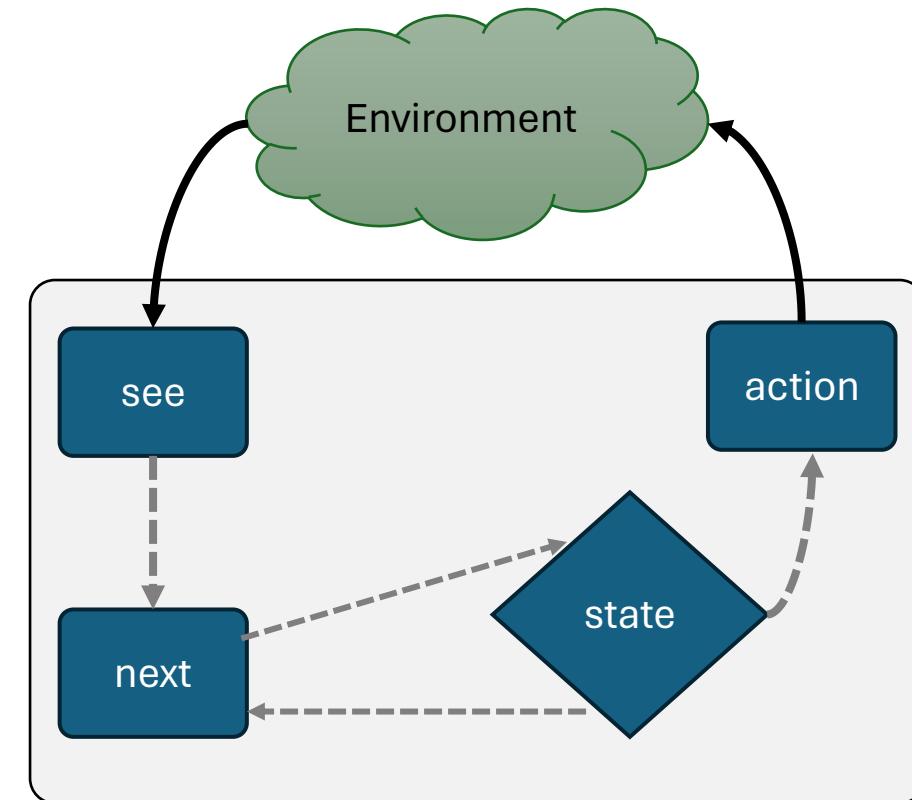
- 1956–present: Symbolic Reasoning Agents
 - Agents make decisions about what to do via symbol manipulation.
 - Its purest expression, proposes that agents use explicit logical reasoning in order to decide what to do.
- 1985–present: Reactive Agents
 - Problems with symbolic reasoning led to a reaction against this
 - led to the reactive agents movement
- 1990–present: Hybrid Agents
 - Hybrid architectures attempt to combine the best of reasoning and reactive architectures

Reactive Architectures

- There are many unsolved (some would say insoluble) problems associated with symbolic AI.
 - These problems have led some researchers to question the viability of the whole paradigm, and to the development of reactive architectures.
 - Although united by a belief that the assumptions underpinning mainstream AI are in some sense wrong, *reactive* agent researchers use many different techniques.
- In this lecture, we look at alternative architectures that better support some classes of agents and robots.
 - At the end, we then examine how *hybrid architectures* exploits the best aspects of deliberative and reactive ones.

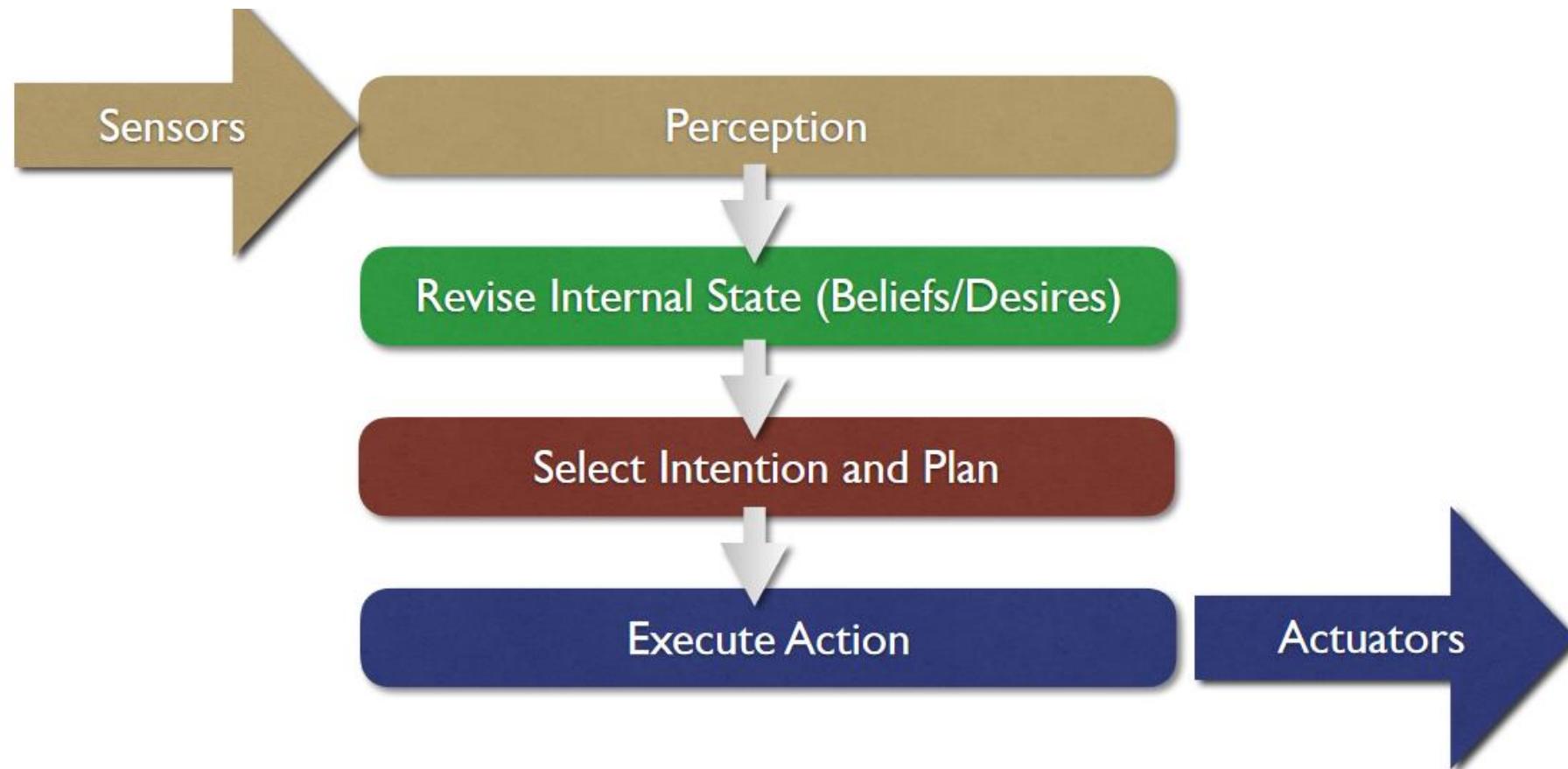
General Control Architecture

- So far, we have viewed the control architecture of an agent as one that:
 - *Perceives* the environment
 - *Revises* its internal state, identifying beliefs and desires
 - *Selects* actions from its intention and plan
 - *Acts*, possibly changing the environment
- Intention Reconsideration is important in highly dynamic environments

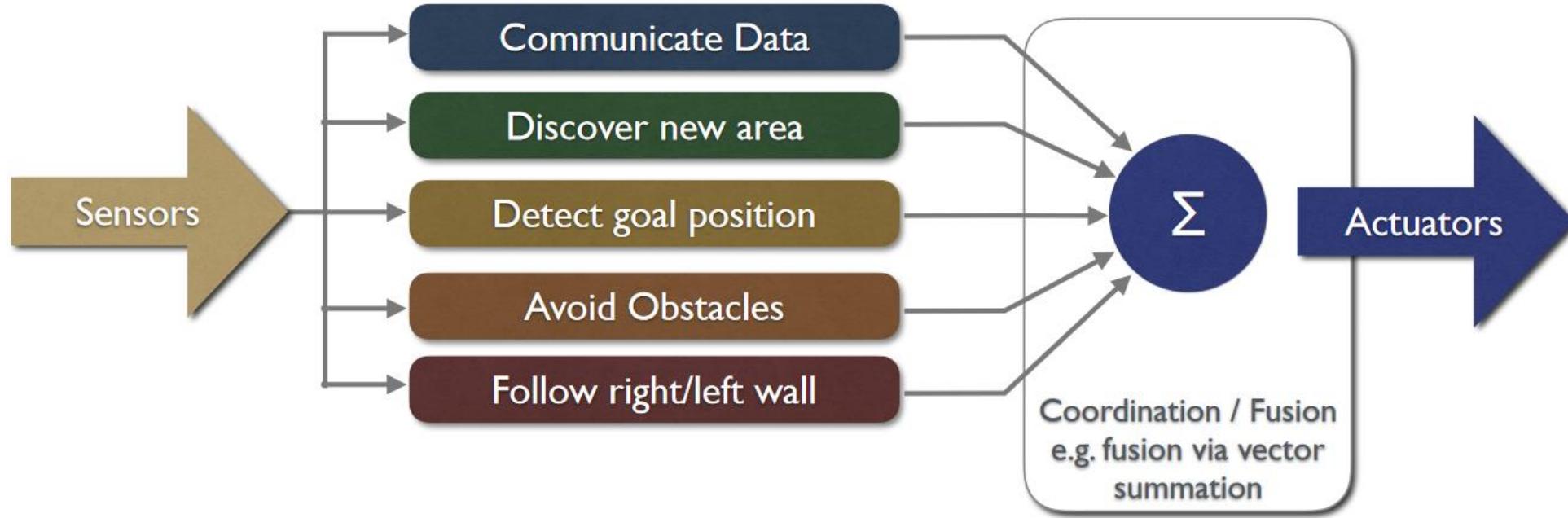


Agent Control Loop as Layers

The classic “Sense/Plan/Act” approach breaks it down serially like this



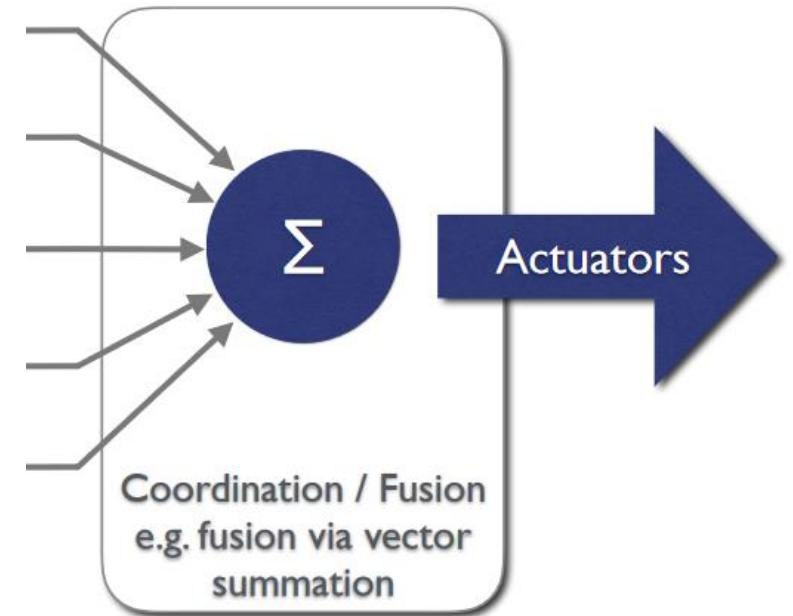
Behaviours



- Behaviour based control sees things differently
 - Behavioural chunks of control each connecting sensors to actuators
 - Implicitly parallel
 - Particularly well suited to Autonomous Robots

Behaviours

- Range of ways of combining behaviours.
- Some examples:
 - Pick the “best”
 - Sum the outputs
 - Use a weighted sum
- Flakey redux used a fuzzy combination which produced a nice integration of outputs.



Subsumption Architecture

- A subsumption architecture is a hierarchy of task-accomplishing behaviours.
 - Each behaviour is a rather simple rule-like structure.
 - Each behaviour ‘competes’ with others to exercise control over the agent.
 - Lower layers represent more primitive kinds of behaviour, (such as avoiding obstacles), and have precedence over layers further up the hierarchy.
- The resulting systems are, in terms of the amount of computation they do, extremely simple.
 - Some of the robots do tasks that would be impressive if they were accomplished by symbolic AI systems.



Rodney Brooks “subsumption architecture” was originally developed open Genghis

Brooks Behavioural Languages

- Brooks proposed the following three theses:
 1. Intelligent behaviour can be generated *without* explicit *representations* of the kind that symbolic AI proposes.
 2. Intelligent behaviour can be generated *without* explicit *abstract reasoning* of the kind that symbolic AI proposes.
 3. Intelligence is an *emergent* property of certain complex systems.

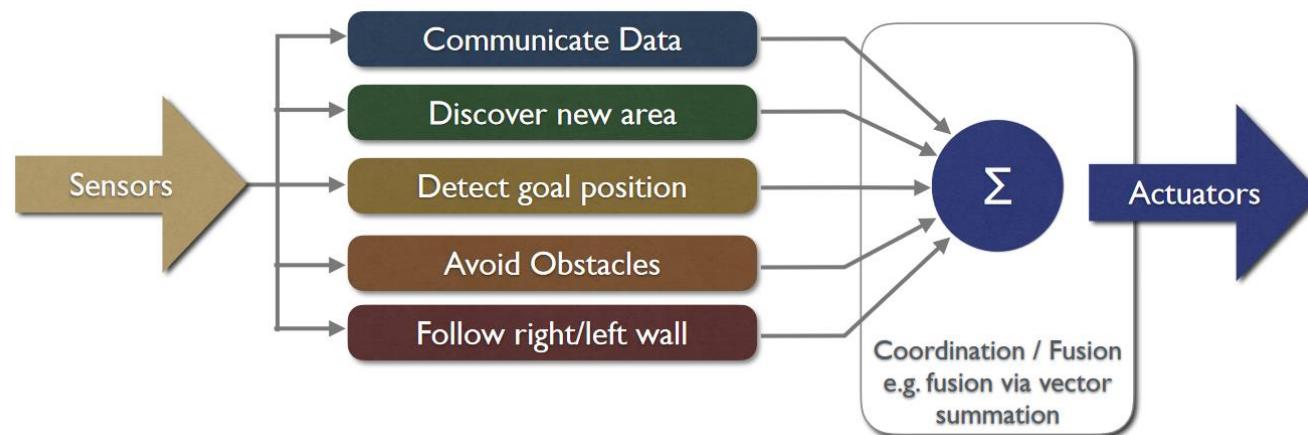


Brooks Behavioural Languages

- He identified two key ideas that have informed his research:
 1. ***Situatedness and embodiment:*** ‘Real’ intelligence is situated in the world, not in disembodied systems such as theorem provers or expert systems.
 2. ***Intelligence and emergence:*** ‘Intelligent’ behaviour arises as a result of an agent’s interaction with its environment. Also, intelligence is ‘in the eye of the beholder’; it is not an innate, isolated property.
- Brooks built several agents (such as Genghis
[https://en.wikipedia.org/wiki/Genghis_\(robot\)](https://en.wikipedia.org/wiki/Genghis_(robot))) based on his subsumption architecture to illustrate his ideas

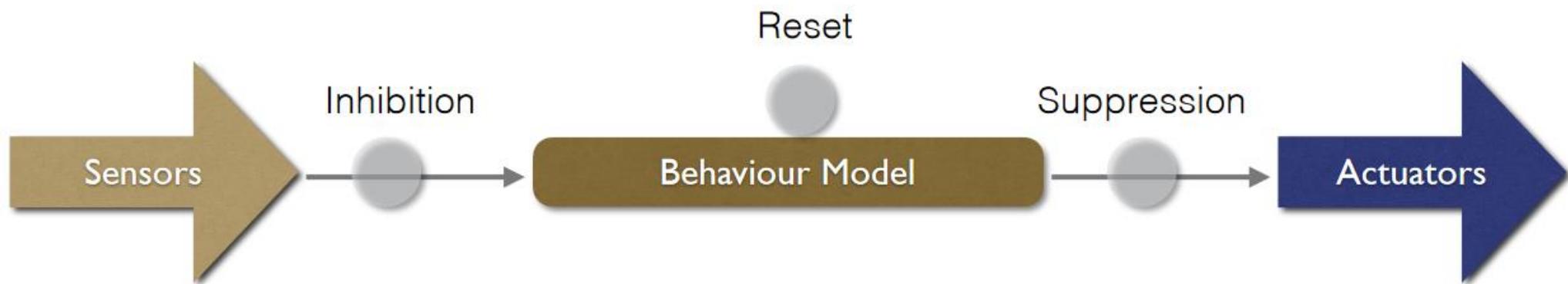
Subsumption Architecture

- It is the piling up of layers that gives the approach of its power.
 - Complex behaviour emerges from simple components
 - Since each layer is independent, each can independently be:
 - Coded / Tested / Debugged
 - Can then assemble them into a complete system



Abstract view of a Subsumption Machine

- Layered approach based on levels of competence
 - Higher level behaviours *inhibit* lower levels
- Augmented finite state machine:



Emergent behaviour

- Important but not well-understood phenomenon
 - Often found in behaviour-based/reactive systems
- Agent behaviours “emerge” from interactions of rules with environment.
 - Sum is greater than the parts.
 - The interaction links rules in ways that weren’t anticipated.
- Coded behaviour: In the programming scheme
- Observed behaviour: In the eyes of the observer
 - There is no one-to-one mapping between the two!
- When observed behaviour “exceeds” programmed behaviour, then we have emergence.

Emergent Flocking

Flocking is a classic example of emergence, e.g. Reynolds “Boids”, or Mataric’s “nerd herd”.



Each agent uses the following three rules:

1. *Don't run into any other robot*
2. *Don't get too far from other robots*
3. *Keep moving if you can*

When run in parallel on many agents, the result is flocking

ToTo

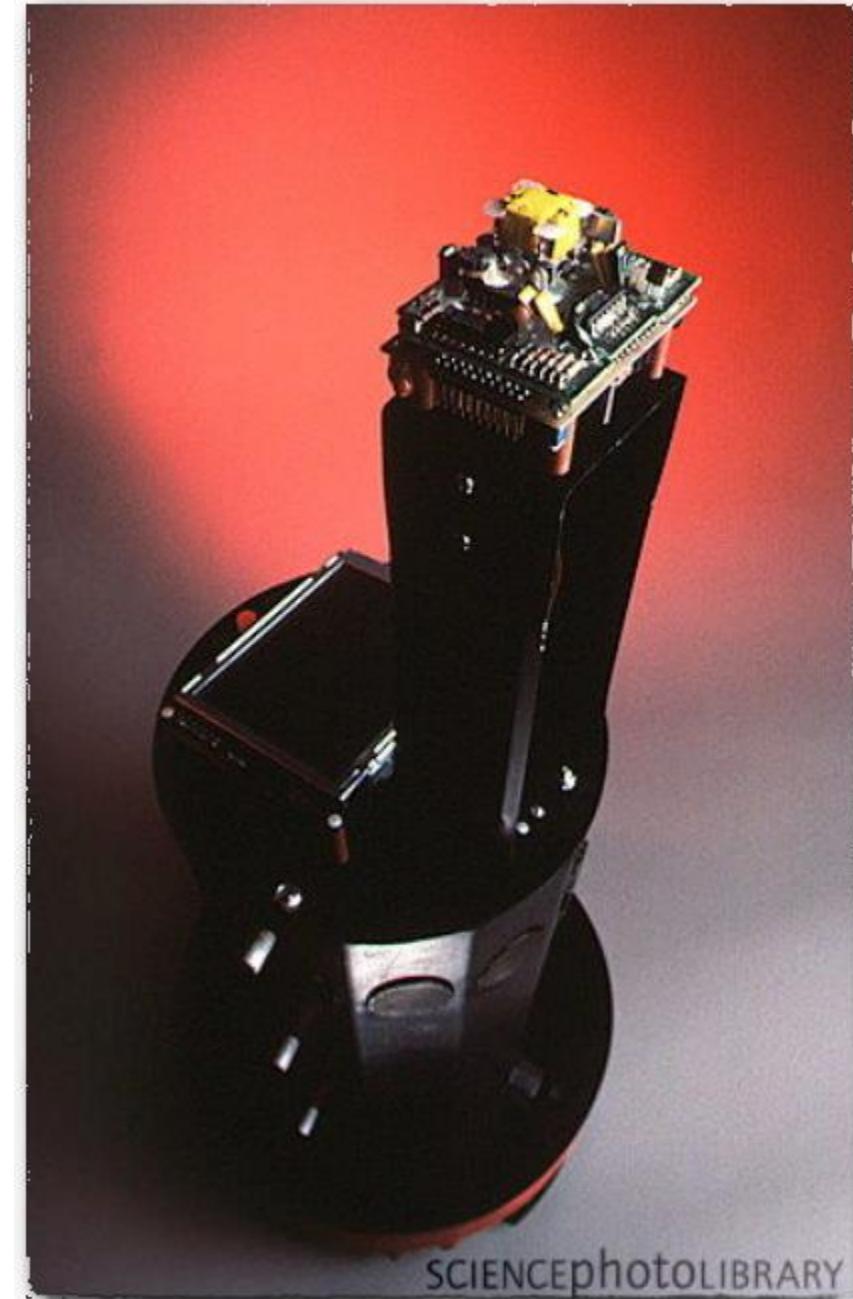
- Maja Mataric 's Toto is based on the subsumption architecture
 - Can map *spaces* and *execute plans* without the need for a *symbolic* representation.
 - Inspired by "...the ability of insects such as bees to identify shortcuts between feeding sites..."
- Each feature/landmark is a set of sensor readings
 - Signature
- Recorded in a behaviour as a triple:
 - Landmark type
 - Compass heading
 - Approximate length/size
- Distributed topological map



SCIENCEphotOLIBRARY

ToTo

- Whenever Toto visited a particular landmark, its associated map behaviour would become activated
 - If no behaviour was activated, then the landmark was new, so a new behaviour was created
 - If an existing behaviour was activated, it inhibited all other behaviours
- Localization was based on which behaviour was active.
 - No map object, but the set of behaviours clearly included map functionality.



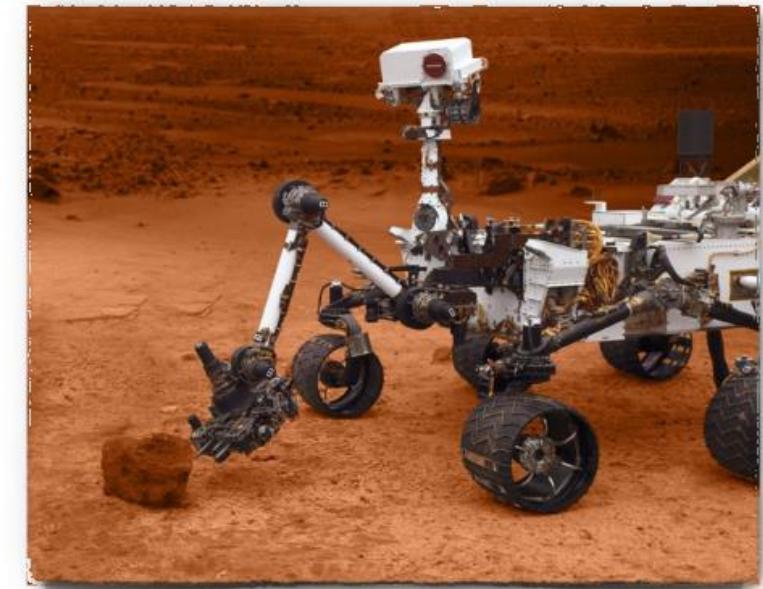
SCIENCEphotOLIBRARY

Steel's Mars Explorer System

- Steels' Mars explorer system
 - Uses the *subsumption* architecture to achieve near-optimal cooperative performance in simulated 'rock gathering on Mars' domain
 - *Individual behaviour* is governed by a set of simple rules.
 - *Coordination between agents* can also be achieved by leaving "markers" in the environment.

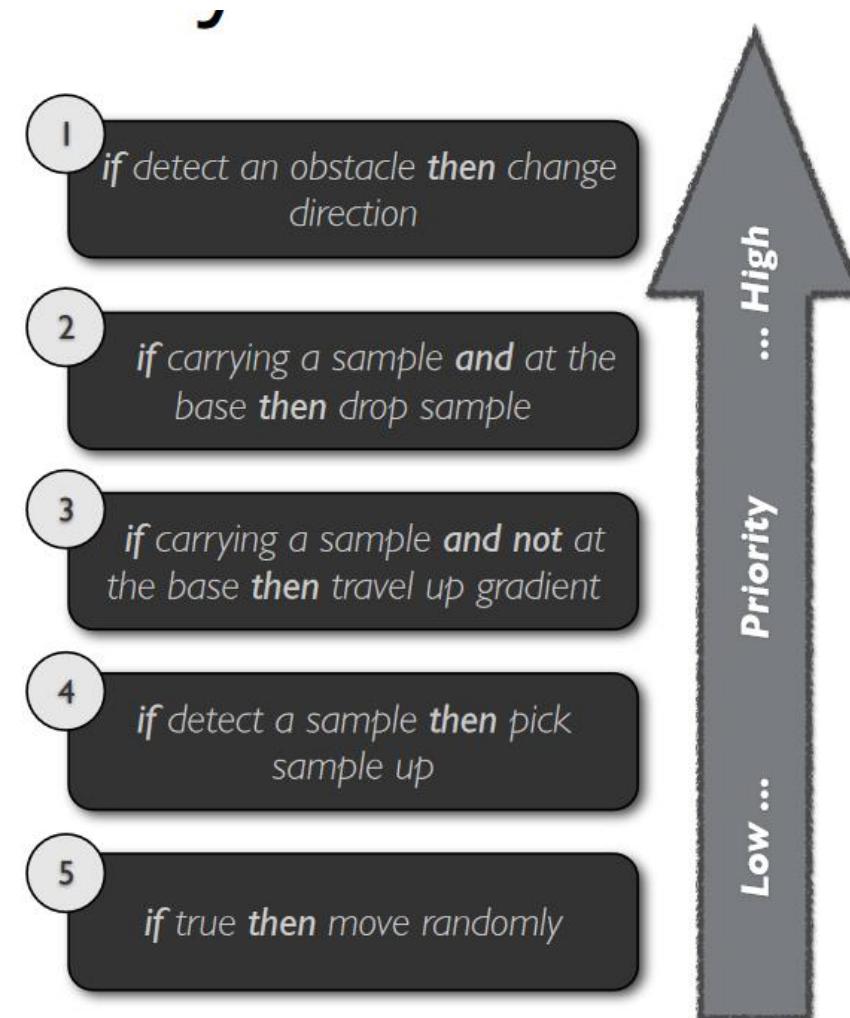
Objective

To explore a distant planet, and in particular, to collect sample of a precious rock. The location of the samples is not known in advance, but it is known that they tend to be clustered.



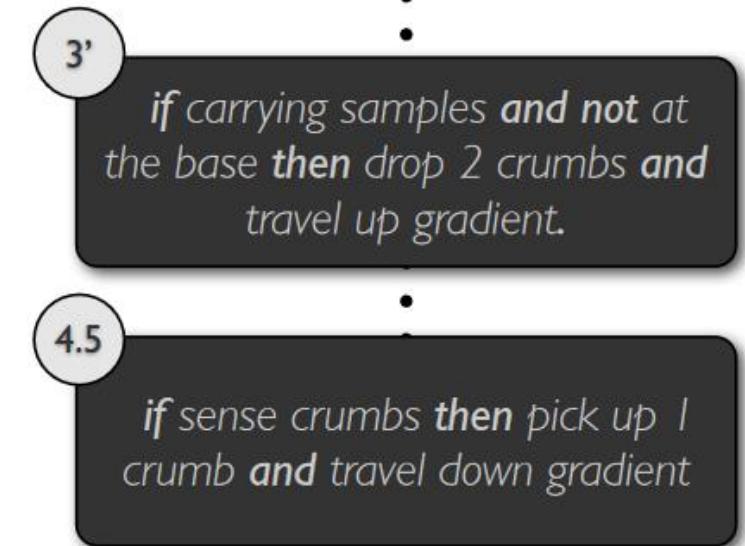
Steel's Mars Explorer System

1. For individual (non-cooperative) agents, the lowest-level behaviour, (and hence the behaviour with the highest “priority”) is obstacle avoidance.
2. Any samples carried by agents are dropped back at the mother-ship.
3. If not at the mother-ship, then navigate back there.
 - The “gradient” in this case refers to a virtual “hill” radio signal that slopes up to the mother ship/base.
4. Agents will collect samples they find.
5. An agent with “nothing better to do” will explore randomly. This is the highest-level behaviour (and hence lowest level “priority”).



Steel's Mars Explorer System

- Existing strategy works well when samples are distributed randomly across the terrain.
 - However, samples are located in clusters
 - Agents should cooperate with each other to locate clusters
- Solution to this is based on foraging ants.
 - Agents leave a “radioactive” trail of crumbs when returning to the mother ship with samples.
 - If another agent senses this trail, it follows the trail back to the source of the samples
 - It also picks up some of the crumbs, making the trail fainter.
 - If there are still samples, the trail is reinforced by the agent returning to the mother ship (leaving more crumbs)
 - If no samples remain, the trail will soon be erased.



Situated Automata

- Approach proposed by Rosenschein and Kaelbling.
 - An agent is specified in a rule-like (declarative) language.
 - Then compiled down to a digital machine, which satisfies the declarative specification
 - This digital machine can operate in a provable time bound.
 - Reasoning is done off line, at compile time, rather than online at run time.
- The theoretical limitations of the approach are not well understood.
 - Compilation (with propositional specifications) is equivalent to an NP-complete problem.
 - The more expressive the agent specification language, the harder it is to compile it.

Situated Automata

- An agent is specified by perception and action
 - Two programs are used to synthesise agents:
 1. RULER specifies the perception component
 - (see opposite)
 2. GAPPS specifies the action component
 - Takes a set of goal reduction rules and a top-level goal
 - (symbolically specified) and generates a non-symbolic program

RULER takes as its input three components...

“...[A] specification of the semantics of the [agent’s] inputs (“whenever bit 1 is on, it is raining”); a set of static facts (“whenever it is raining, the ground is wet”); and a specification of the state transitions of the world (“if the ground is wet, it stays wet until the sun comes out”). The programmer then specifies the desired semantics for the output (“if this bit is on, the ground is wet”), and the compiler ... [synthesises] a circuit whose output will have the correct semantics... All that declarative “knowledge” has been reduced to a very simple circuit...”

Kaelbling, L.P. (1991) A Situated Automata Approach to the Design of Embedded Agents. SIGART Bulletin, 2(4): 85-88

Limitations of Reactive Systems

- Although there are clear advantages of Reactive Systems, there are also limitations!
 - If a model of the environment isn't used, then sufficient information of the local environment is needed for determining actions
 - As actions are based on local information, such agents inherently take a “short-term” view
 - *Emergent behaviour is very hard to engineer* or validate; typically a trial and error approach is ultimately adopted
 - Whilst agents with few layers are straightforward to build, *models using many layers are inherently complex* and difficult to understand.

Hybrid Architectures

- Many researchers have argued that neither a completely deliberative nor completely reactive approach is suitable for building agents.
- They have suggested using *hybrid* systems, which attempt to marry classical and alternative approaches.
- An obvious approach is to build an agent out of two (or more) subsystems:
 - a *deliberative* one, containing a symbolic world model, which develops plans and makes decisions in the way proposed by symbolic AI; and
 - a *reactive* one, which is capable of reacting to events without complex reasoning.

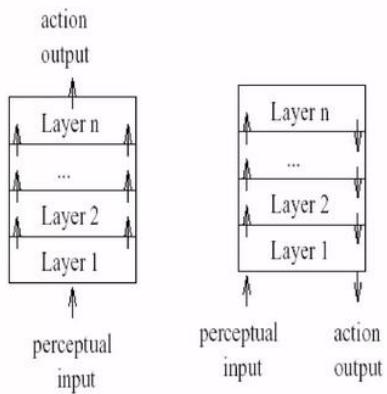
Hybrid Architectures

- Often, the reactive component is given some kind of precedence over the deliberative one.
- This kind of structuring leads naturally to the idea of a layered architecture, of which *InterRap* and *TouringMachines* are examples.
 - In such an architecture, an agent's control subsystems are arranged into a *hierarchy*...
 - ...with higher layers dealing with information at *increasing levels of abstraction*.
- A key problem in such architectures is what kind control framework to embed the agent's subsystems in, to manage the interactions between the various layers.

Hybrid Architectures

Horizontal layering

- Layers are each directly connected to the sensory input and action output.
- In effect, each layer itself acts like an agent, producing suggestions as to what action to perform.

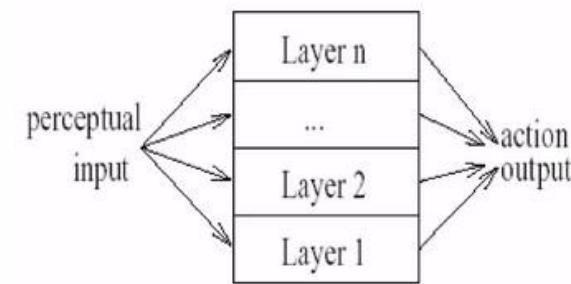


$O(mn)$ interactions
between layers

Not fault tolerant to
layer failure

Vertical layering

- Sensory input and action output are each dealt with by at most one layer each.

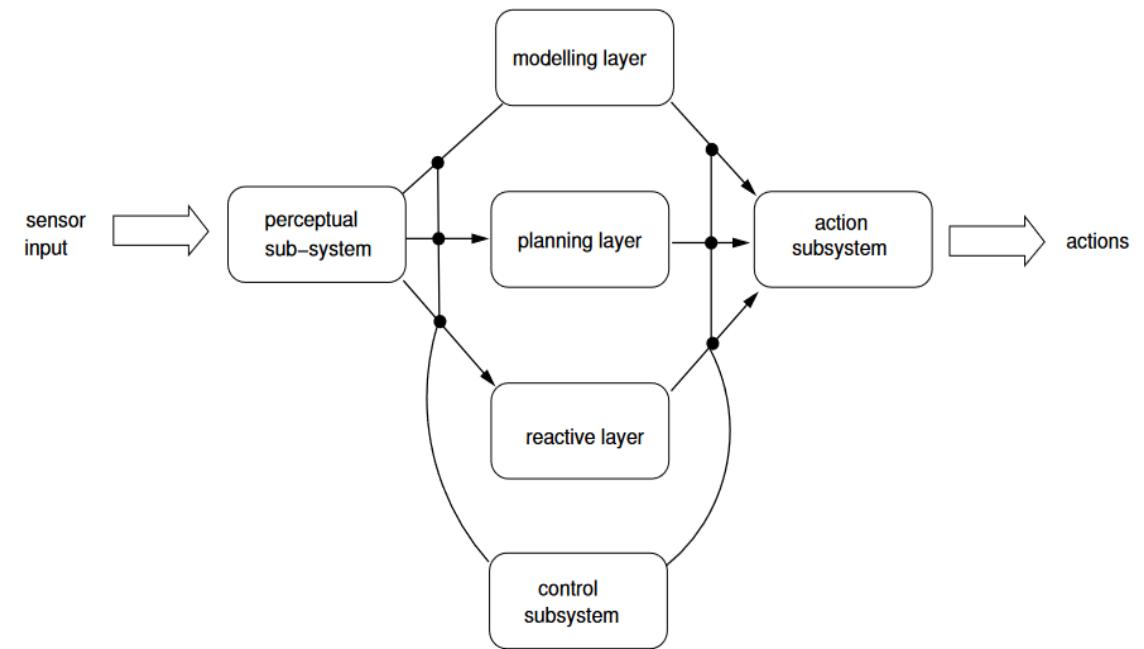


$O(m^n)$ possible
options to be
considered

Introduces bottleneck
in central control system

Ferguson - TouringMachines

- The TouringMachines architecture consists of perception and action subsystems
 - These interface directly with the agent's environment, and three control layers, embedded in a control framework, which mediates between the layers.



Ferguson - TouringMachines

- The reactive layer is implemented as a set of situation-action rules, a` la subsumption architecture.
 - The planning layer constructs plans and selects actions to execute in order to achieve the agent's goals.

```
rule-1: kerb-avoidance
if
    is-in-front(Kerb, Observer) and
    speed(Observer) > 0 and
    separation(Kerb, Observer) < KerbThreshHold
then
    change-orientation(KerbAvoidanceAngle)
```

Ferguson - TouringMachines

- The modelling layer contains symbolic representations of the ‘cognitive state’ of other entities in the agent’s environment.
 - The three layers communicate with each other and are embedded in a control framework, which use control rules.
 - Such control structures have become common in robotics.

```
censor-rule-1:  
    if  
        entity(obstacle-6) in perception-buffer  
    then  
        remove-sensory-record(layer-R, entity(obstacle-6))
```

Real World Example: Stanley

- Won the 2005 DARPA Grand Challenge
 - Used a combination of the subsumption architecture with deliberative planning
 - Consists of 30 different independently operating modules across 6 layers

Global Services Layer

User Interface Layer

Vehicle Interface Layer

Planning and Control layer

Perception layer

Sensor interface layer



The key challenge... was
not one of action, but
one of perception...

Summary

- This lecture has looked at two further kinds of agent:
 - Reactive agents; and
 - Hybrid agents.
- *Reactive agents* build complex behaviour from simple components.
- *Hybrid agents* try to combine the speed of reactive agents with the power of deliberative agents.
- Reading for this week:
 - M.Wooldridge: An introduction to MultiAgent Systems – Ch. 5 Reactive and Hybrid Architectures
 - “A Robust Layered Control System for a Mobile Robot,” Rodney A. Brooks. IEEE Journal of Robotics and Automation, 2(1), March 1986, pp. 14–23. (also MIT AI Memo 864, September 1985)

LECTURE 5: Ontologies & Communication

Introduction to Multi-Agent Systems (MESIIA, MIA)
URV

Social Behaviour

- Previously we looked at:
 - Deductive Agents
 - Practical Reasoning Agents
 - Reactive and Hybrid Agents
- We said:
 - An intelligent agent is a computer system capable of *flexible* autonomous action in some environment.
 - Where by flexible, we mean:
 - Reactive
 - Pro-active
 - Social
- This is where we deal with the “*social*” bit, showing how agents communicate and share information.

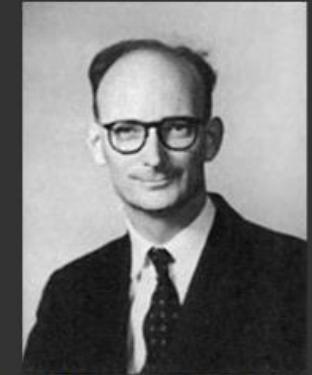
Agent Communication

- In this lecture, we cover macro-aspects of intelligent agent technology, and those issues relating to the agent society, rather than the individual:
 - Communication:
 - speech acts; KQML & KIF; FIPA ACL
 - Ontologies:
 - the role of ontologies in communication
 - aligning ontologies
 - OWL
- There are some limited things that one can do without communication, but they are..., well..., *limited!!!*
 - Most work on multiagent systems assumes communication.

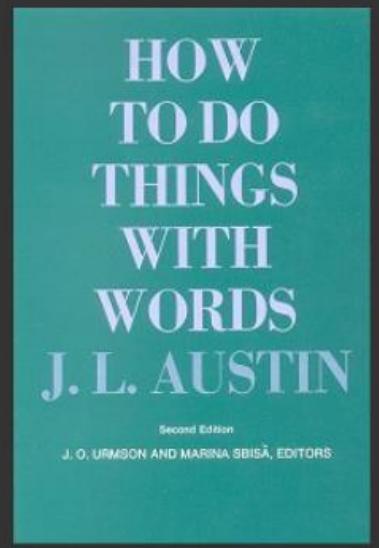
Speech Acts

- Austin's 1962 book “How to Do Things with Words” is usually taken to be the origin of speech acts
- Speech act theories are *pragmatic* theories of language, that is theories of how language *use*:
 - they attempt to account for how language is used by people every day to achieve their goals and intentions.
- Most treatments of communication in (multi-)agent systems borrow their inspiration from speech act theory...
 - ...doubtless because the “action” part can be tied closely to existing ideas about how to model action.
- Austin noticed that some utterances are rather like ‘physical actions’ that appear to *change the state of the world*.

John Langshaw Austin



in 1962, published the book:



Speech Acts

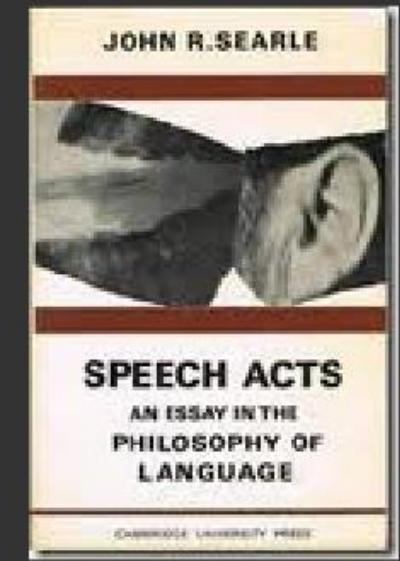
- Paradigm examples are:

- Declaring war ☹;
- Naming a child;
- “I now pronounce you man and wife” ☺

- But more generally, everything we utter is uttered with the intention of satisfying some goal or intention.
- A theory of how utterances are used to achieve intentions is a speech act theory.
 - Proposed by John Searle, 1969.



in 1969, published the book:



Speech Acts: Searle

- In his 1969 book *Speech Acts: an Essay in the Philosophy of Language* he identified:
 - Representatives:
 - such as informing, e.g., ‘it is raining’
 - Directives:
 - attempts to get the hearer to do something e.g., ‘please make the tea’
 - Commisives:
 - which commit the speaker to doing something, e.g., ‘I promise to...’
 - Expressives:
 - whereby a speaker expresses a mental state, e.g., ‘thank you!’
 - Declarations:
 - such as declaring war or naming.

Speech Acts: Searle

- There is some debate about whether this (or any!) typology of speech acts is appropriate.
- In general, a speech act can be seen to have two components:
 - A performative verb:
 - E.g., request, inform, ...etc
 - Propositional content:
 - E.g., ‘the door is closed’

The door is closed...

request

speech act = “*please close the door*”

inform

speech act = “*the door is closed!*”

inquire

speech act = “*is the door closed?*”



Each of the above speech acts result from the same propositional content (“...*the door is closed...*”), but with different performatives

Plan Based Semantics

- How does one define the semantics of speech acts?
 - When can one say someone has uttered, e.g., a request or an inform?
 - Cohen & Perrault (1979) defined semantics of speech acts using the precondition-delete-add list formalism of planning research.
 - Just like STRIPS planner
 - Note that a speaker cannot (generally) force a hearer to accept some desired mental state.

The semantics for “request”

request(s, h, φ)

precondition:

- *s believes h can do φ*
(you don't ask someone to do something unless you think they can do it)
- *s believe h believe h can do φ*
(you don't ask someone unless they believe they can do it)
- *s believe s want φ*
(you don't ask someone unless you want it!)

post-condition:

- *h believe s believe s want φ*
(the effect is to make them aware of your desire)

KQML and KIF

- We now consider agent communication languages (ACLs)
 - ACLs are standard formats for the exchange of messages

“... [developing] protocols for the exchange of represented knowledge between autonomous information systems...”

Tim Finin, 1993

- One well known ACL is KQML, developed by the DARPA-funded Knowledge Sharing Effort (KSE).
 - The ACL proposed by KSE was comprised of two parts:
 - the message itself: the Knowledge Query and Manipulation Language (KQML); and
 - the body of the message: the Knowledge interchange format (KIF).

KQML and KIF

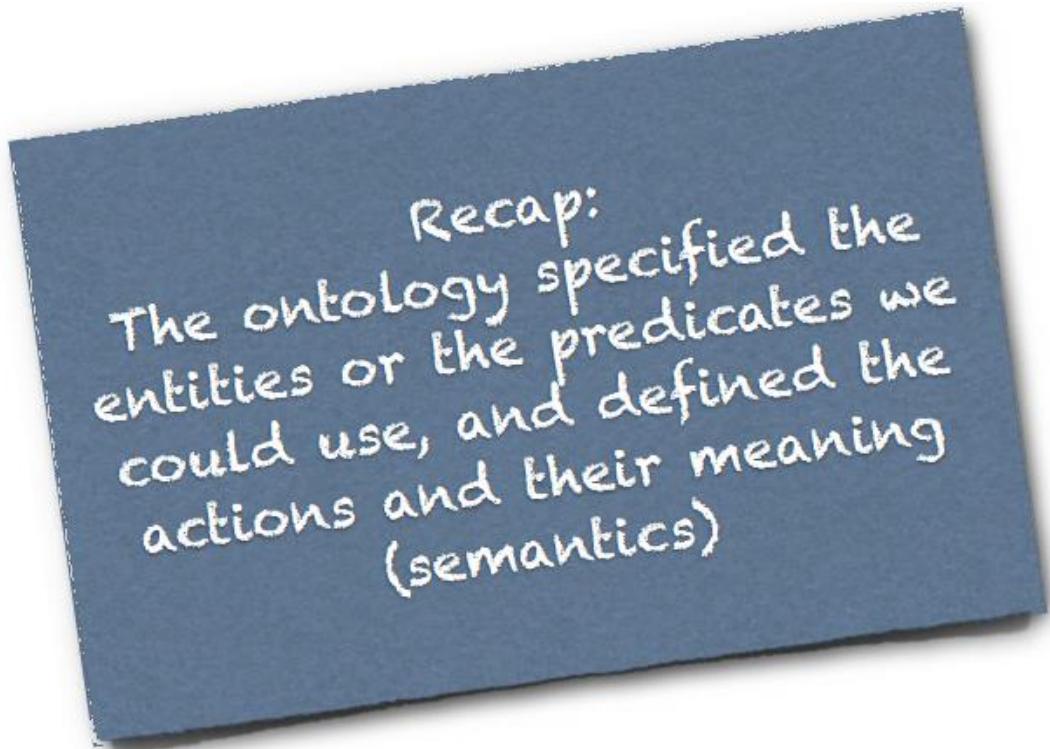
- KQML is an ‘outer’ language, that defines various acceptable ‘communicative verbs’, or performatives.
 - Example performatives:
 - ask-if (‘is it true that. . . ’)
 - perform (‘please perform the following action. . . ’)
 - tell (‘it is true that. . . ’)
 - reply (‘the answer is . . . ’)
- KIF is a language for expressing message content, or domain knowledge.
 - It can be used to writing down ontologies.
 - KIF is based on first-order logic.

KQML & Ontologies

- In order to be able to communicate, agents need to agree on the words (terms) they use to describe a domain.
 - Always a problem where multiple languages are concerned.
- A formal specification of a set of terms is known as a ontology.
 - The DARPA Knowledge Sharing Effort project has associated with it a large effort at defining common ontologies
 - software tools like ontolingua, etc, for this purpose.
- We've previously discussed the use of ontologies and semantics...

Blocksworld

- The environment is represented by an ontology.



Blocksworld Ontology

$On(x,y)$	object x on top of object y
$OnTable(x)$	object x is on the table
$Clear(x)$	nothing is on top of object x
$Holding(x)$	arm is holding x

$Stack(x, y)$

pre	$Clear(y) \wedge Holding(x)$
del	$Clear(y) \wedge Holding(x)$
add	$ArmEmpty \wedge On(x, y)$

$UnStack(x, y)$

pre	$On(x, y) \wedge Clear(x) \wedge ArmEmpty$
del	$On(x, y) \wedge ArmEmpty$
add	$Holding(x) \wedge Clear(y)$

Ontologies

- The role of an ontology is to fix the meaning of the terms used by agents.

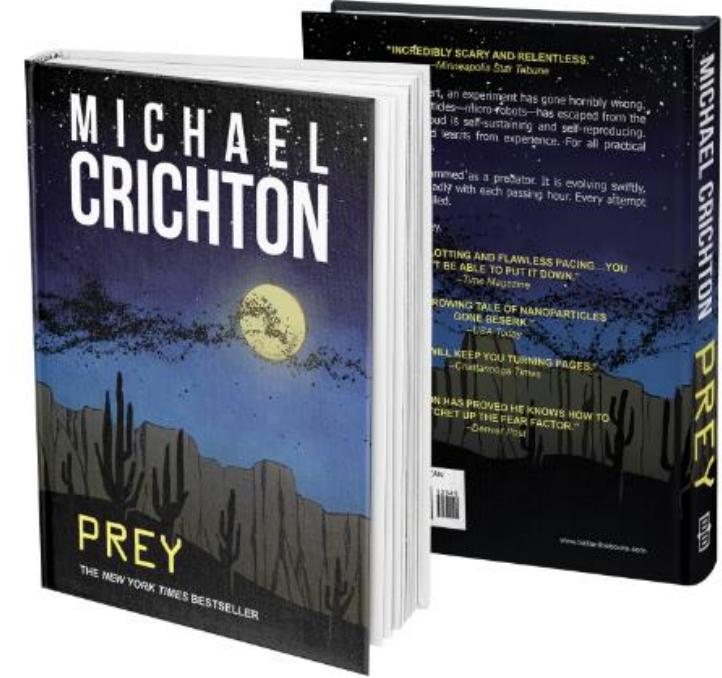
“... An ontology is a formal definition of a body of knowledge. The most typical type of ontology used in building agents involves a structural component. Essentially a taxonomy of class and subclass relations coupled with definitions of the relationships between these things ...”

Jim Hendler

- How do we do this? Typically by defining new terms in terms of old ones.
 - Let’s consider an example.

Ontologies

- Alice:
 - Did you read “Prey”?
- Bob:
 - No, what is it?
- Alice:
 - A science fiction novel. Well, it is also a bit of a horror novel. It is about multiagent systems going haywire.



Ontologies

- What is being conveyed about “Prey” here?
 1. It is a novel
 2. It is a science fiction novel
 3. It is a horror novel
 4. It is about multiagent systems
- Alice assumes that Bob knows what a “novel” is, what “science fiction” is and what “horror” is.
- She thus defines a new term “Pery” in terms of ones that Bob already knows.

Types of objects

Classes

collections of things with similar properties

Instances

specific examples of classes

Relations

Describe the properties of objects and connect them together

Note that we also have these types of objects in languages such as Java, or modelling frameworks such as ER Diagrams. Such languages and frameworks also support inheritance.

Ontologies

- Part of the reason this interaction works is that Bob has knowledge that is relevant.
 - Bob knows that novels are fiction books
 - “novel” is a subclass of “fiction book”
 - Bob knows things about novels: they have
 - authors,
 - publishers,
 - publication dates, and so on.
- Because “Prey” is a novel, it *inherits* the properties of novels. It has an author, a publisher, a publication date.
 - Instances inherit attributes from their classes.

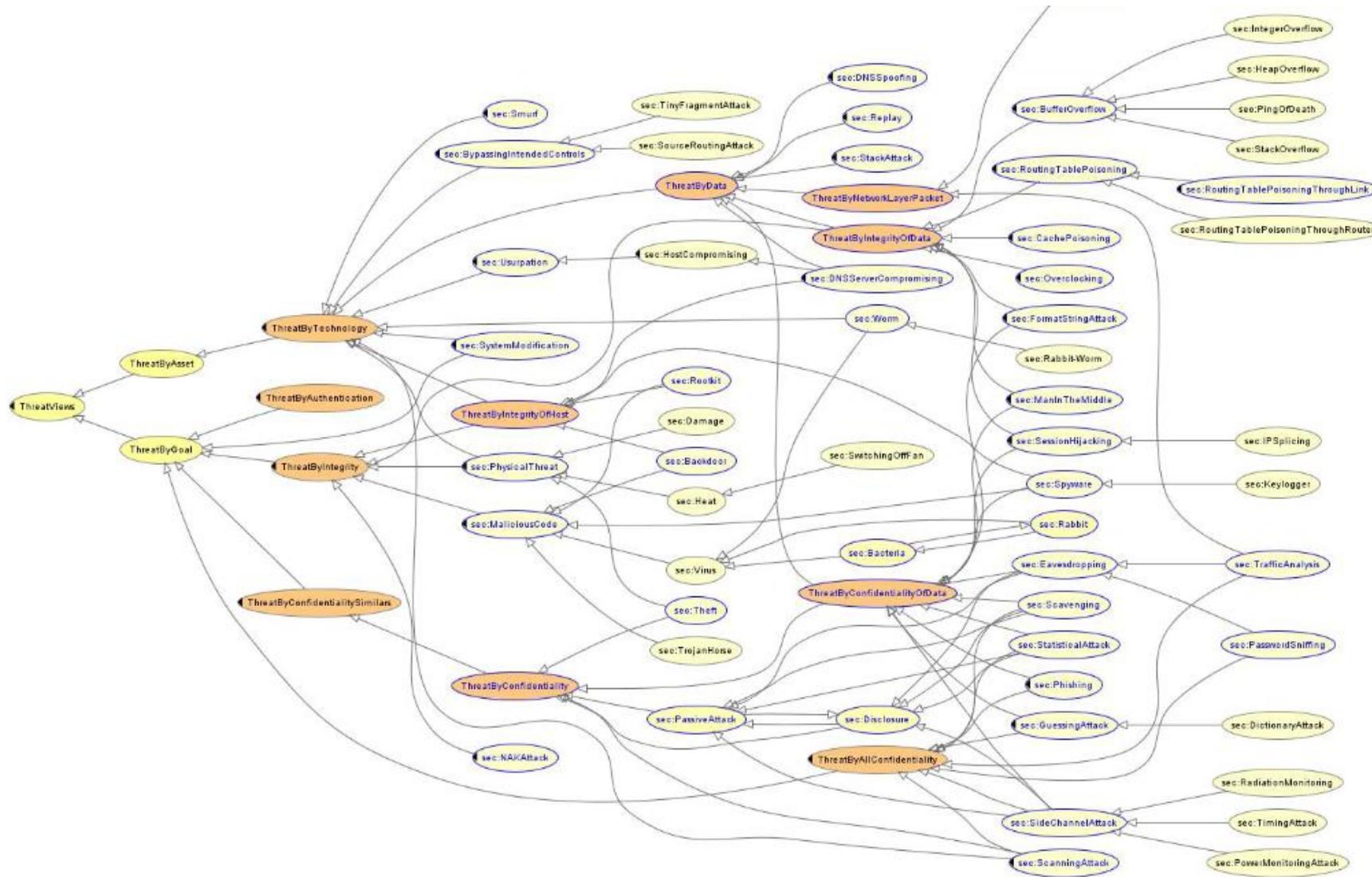
Ontology Inheritance

- Classes also inherit.
 - Classes inherit attributes from their super-classes
 - If “novel” is a subclass of “fiction book”, then “fiction book” is a superclass of “novel”
- Fiction books are books.
 - Books are sold in bookstores.
 - Thus fiction books are sold in bookstores.

Ontologies

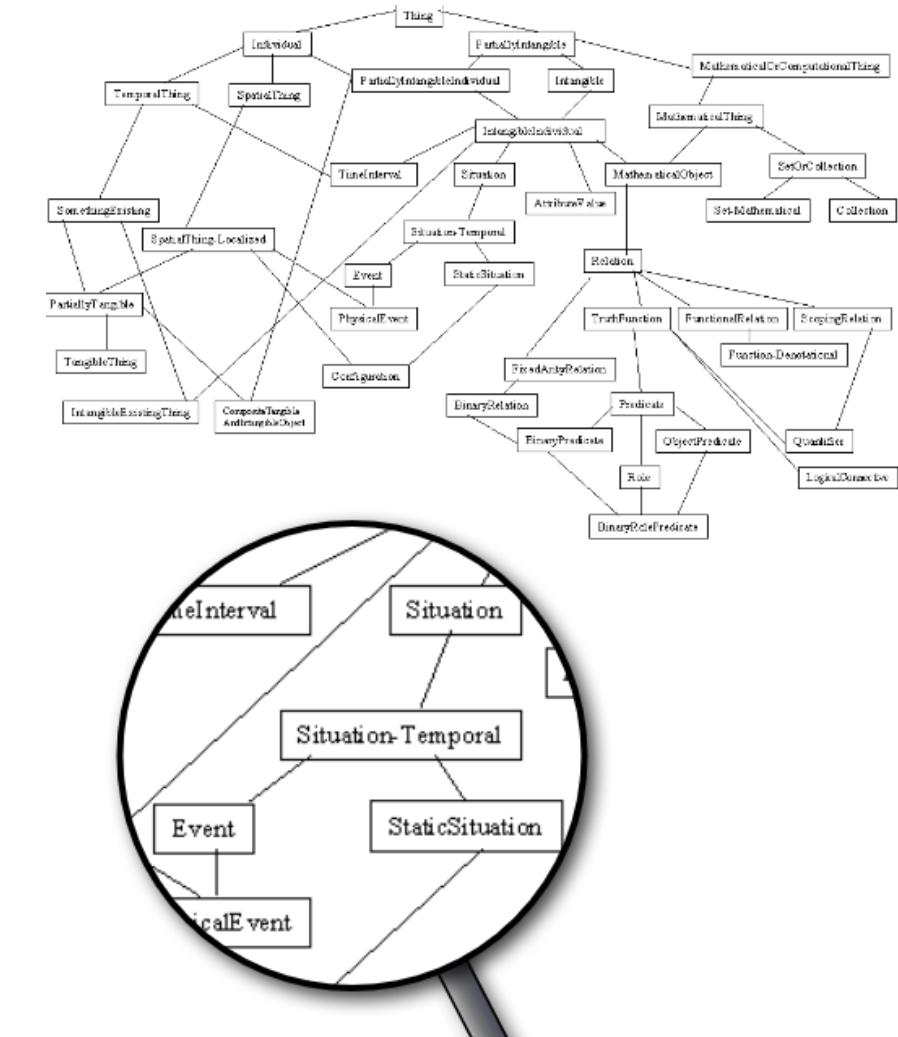
- A lot of knowledge can be captured using these notions.
 - We specify which class “is-a” sub-class of which other class.
 - We specify which classes have which attributes.
 - An axiomatic theory can also be included to support inference
 - if socrates is [an instance of a] man, and all men are mortal...
 - ... we can infer that socrates is mortal!
- This structure over knowledge is called an ontology.
 - A knowledge base is an ontology with a set of instances.
- A huge number of ontologies have been constructed.

An ontology of threats



Ontologies

- In general, there are multiple ontologies at different levels of detail.
 - Application ontology
 - Like the threat ontology (see opposite)
 - Domain ontology
 - Upper ontology
 - Contains very general information about the world.
- The more specific an ontology, the less reusable it is.



Multiple Ontologies

- Application and domain ontologies will typically overlap
 - Illustrated by the challenges of facilitating interoperability between similar ontologies.
 - Different knowledge systems can be integrated to form merged knowledge bases.
- But in many cases, all that is needed is to be understood!

Ontologies as perspectives on a domain

A single domain may have an **intended representation** in the real world, that is not perfectly represented by any single formal ontology. Many separate ontologies then emerge, based on different contexts...

Ontology Alignment Evaluation Initiative

A test suite of similar ontologies used to test out alignment systems that link different ontologies representing the same domain.

The conference test suite consists of 21 ontology pairs.

Modelling and Context

- The problem with modelling ontologies is that different designers have different contexts, and requirements

The Architect

*When modelling a bridge,
important characteristics
include:*

*tensile strength
weight
load
etc*



*Pat Hayes, 2001 in
conversation*

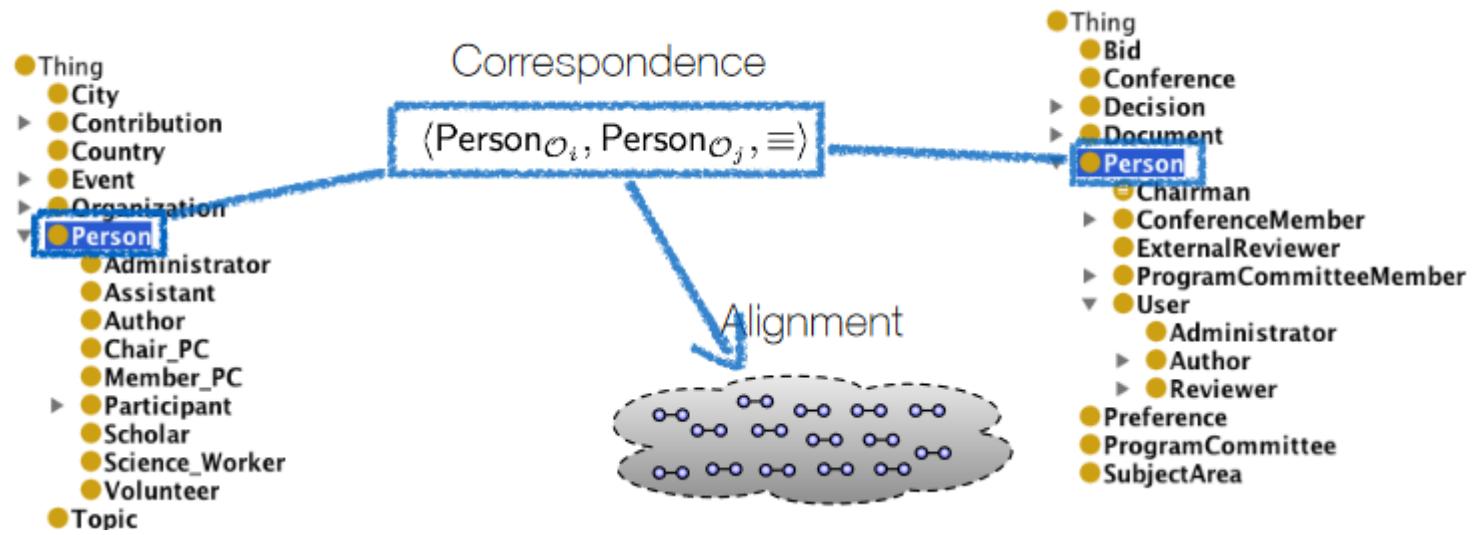
The Military

*When modelling a bridge,
important characteristics
include:*

*what munitions are
required to destroy it!*

Ontologies, Alignments and Correspondences

- Different ontological models exist for overlapping domains
 - Modelled implicitly, or explicitly by defining entities (classes, roles etc), typically using some logical theory, i.e. an Ontology
- Alignment Systems align similar ontologies



Aligning Agents' Ontologies

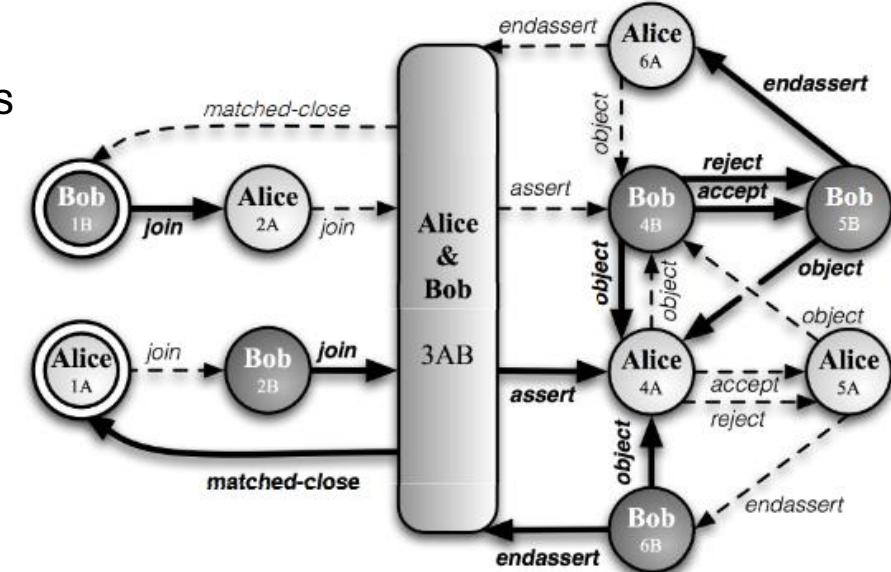
“... as agents can differ in the ontologies they assume, the resulting semantic heterogeneity can impede meaningful communication. One solution is to align the ontologies; i.e. find correspondences between the ontological entities to resolve this semantic heterogeneity. However, this raises the question: how can agents align ontologies that they do not want to disclose?...”

Terry Payne & Valentina Tamma, 2014

- Agent alignment is inherently *decentralised!*

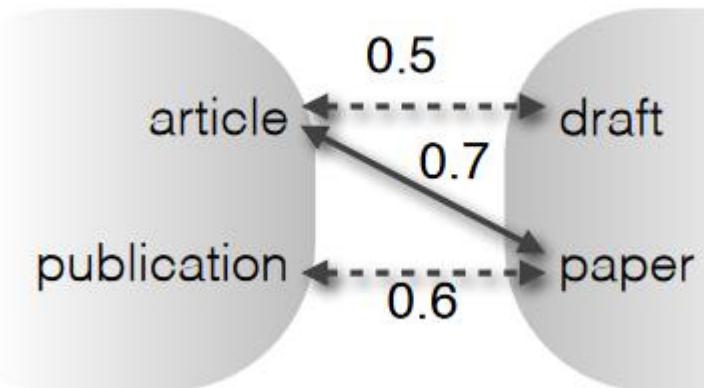
Correspondence Inclusion Dialogue

- Correspondence Inclusion Dialogue (CID)
 - Allows two agents to exchange knowledge about correspondences to agree upon a mutually acceptable final alignment AL.
 - This alignment aligns only those entities in each agents' working ontologies, without disclosing the ontologies, or all of the known correspondences.
- Assumptions
 1. Each agent knows about different correspondences from different sources
 2. This knowledge is partial, and possibly ambiguous; i.e. more than one correspondence exists for a given entity
 3. Agents associate a utility (Degree of Belief) κ_c to each unique correspondence



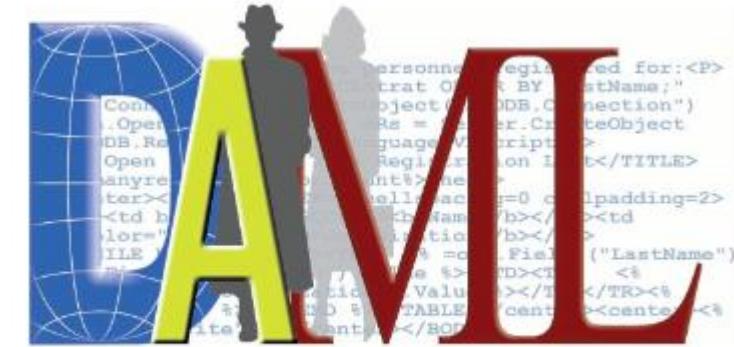
Picking the right correspondences

- Quality vs Quantity
 - Do we maximise coverage
 - Preferable when merging the whole ontology
 - Do we find the “best” mappings
 - Preferable when aligning specific signatures



OWL - Web Ontology Language

- A general purpose family of ontology languages for describing knowledge
 - Originated from the DARPA Agent Markup Language Program
 - Followup to DARPA Control of Agent Based Systems (CoABS)
- Based on description logics
 - Various flavours with different expressivity / computability
 - Different syntaxes: XML, Turtle, Manchester Syntax...
 - Underpins the Semantic Web

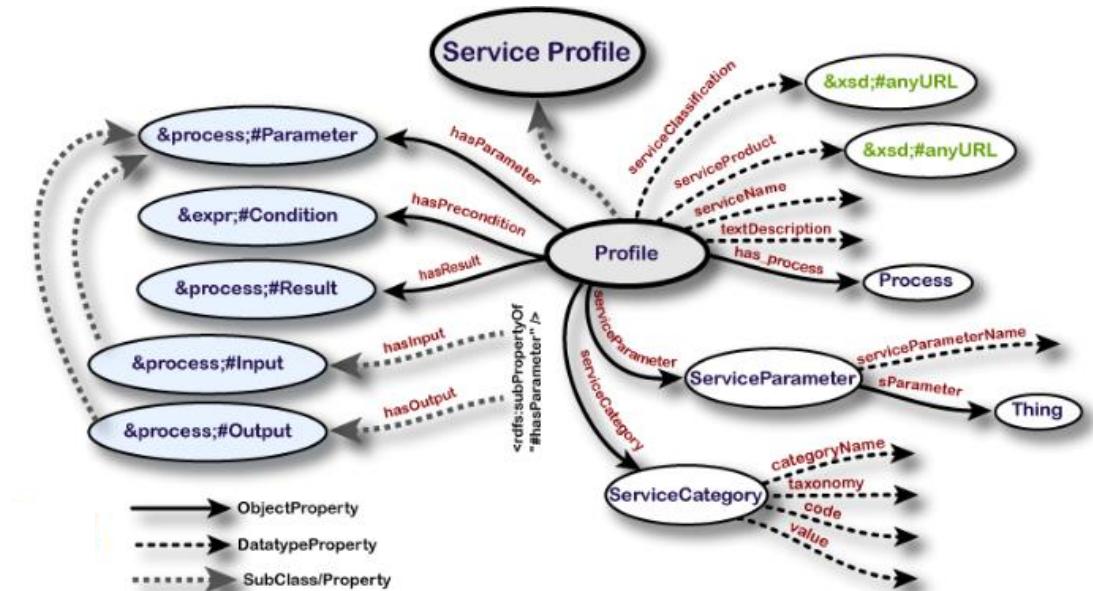
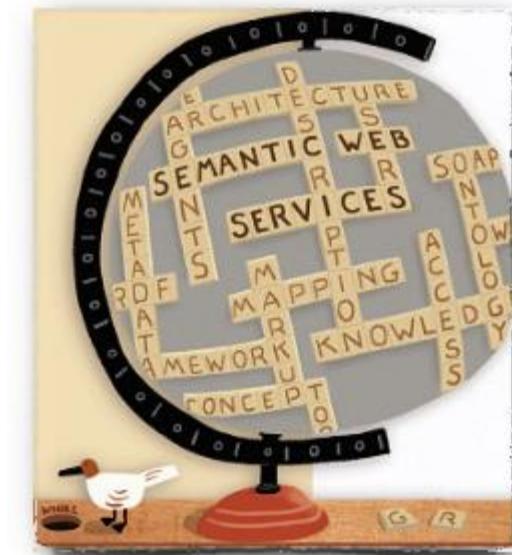


OWL Example

```
NS1:geographicCoordinates rdf:nodetID='A179'>
<NS1:mapReferences>North America</NS1:mapReferences>
<NS1:totalArea>9629091</NS1:totalArea>
<NS1:landArea>9158960</NS1:landArea>
<NS1:waterArea>470131</NS1:waterArea>
<NS1:comparativeArea>about half the size of Russia; about three-tenths the size of Africa; about half the size of South America (or slightly larger than Brazil); slightly larger than China; about two and a half times the size of Western Europe
</NS1:comparativeArea>
<NS1:landBoundaries>12034</NS1:landBoundaries>
<NS1:coastline>19924</NS1:coastline>
<NS1:contiguousZone>24</NS1:contiguousZone>
<NS1:exclusiveEconomicZone>200</NS1:exclusiveEconomicZone>
<NS1:territorialSea>12</NS1:territorialSea>
<NS1:climate>mostly temperate, but tropical in Hawaii and Florida, arctic in Alaska, semiarid in the great plains west of the Mississippi River, and arid in the Great Basin of the southwest; low winter temperatures in the northwest are ameliorated occasionally in January and February by warm chinook winds from the eastern slopes of the Rocky Mountains
</NS1:climate>
<NS1:terrain>vast central plain, mountains in west, hills and low mountains in east; rugged mountains and broad river valleys in Alaska; rugged, volcanic topography in Hawaii
</NS1:terrain>
```

OWL and Services

- OWL-S is an *upper level service ontology* developed to describe agent based (or semantic web) services
 - Profile used for discovery
 - Input/outputs, etc
 - Process model provided a planning formalism
 - Grounding linked to the syntactic messaging



KQML / KIF

- After that digression, we can return to the KQML/KIF show.
 - KQML is an agent communication language. It provides a set of performatives for communication.
- KIF is a language for representing domain knowledge.
 - It can be used to writing down ontologies.
 - KIF is based on first-order logic.
- Given that, let's look at some examples.

KQML/KIF Example

```
(stream-about
  :sender      A
  :receiver    B
  :language    KIF
  :ontology    motors
  :reply-with  q1
  :content     m1
)
```

A asks B for information about motor 1, using the ontology (represented in KIF) about motors.

```
(tell
  :sender      B
  :receiver    A
  :in-reply-to q1
  :content
    (= (torque m1) (scalar 12 kgf))
)
(tell
  :sender      B
  :receiver    A
  :in-reply-to q1
  :content
    (= (status m1) normal)
)
(eos
  :sender      B
  :receiver    A
  :in-reply-to q1
)
```

B responds to A's query q1.
Two facts are sent:
1) that the torque of motor 1
is 12kgf; and
2) that the status of the
motor is normal.

The ask stream is terminated
using the eos performatve.

Problems with KQML

- The basic KQML performative set was fluid
 - Different implementations were not *interoperable*
- Transport mechanisms for messages were not precisely defined
 - Again – *interoperability*
- Semantics of KQML were not rigorously defined
 - Ambiguity resulted in impairing *interoperability*
- There were no commissives in the language
 - Without the ability to *commit* to a task, how could agents *coordinate* behaviour
- The performative set was arguably ad-hoc and overly large

FIPA ACL

- More recently, the Foundation for Intelligent Physical Agents (FIPA) started work on a program of agent standards
 - the centrepiece is an ACL.
- Basic structure is quite similar to KQML:
 - The number of performatives was reduced to 20.
 - A formal semantics has been defined for the language, using the language SL
 - SL can represent beliefs, desires and uncertain beliefs, as well as actions

FIPA ACL example

```
(inform
  :sender    agent1
  :receiver  agent5
  :content   (price good200 150)
  :language  sl
  :ontology  hpl-auction
)
```



FIPA ACL Performatives

performative	passing info	requesting info	negotiation	performing actions	error handling
accept-proposal			x		
agree				x	
cancel		x		x	
cfp			x		
confirm	x				
disconfirm	x				
failure					x
inform	x				
inform-if	x				
inform-ref	x				
not-understood					x
propose			x		
query-if		x			
query-ref		x			
refuse				x	
reject-proposal			x		
request				x	
request-when				x	
request-whenever				x	
subscribe		x			

“Inform” and “Request”

- This lecture has looked at two further kinds of agent:
 - Reactive agents; and
 - Hybrid agents.
- *Reactive agents* build complex behaviour from simple components.
- *Hybrid agents* try to combine the speed of reactive agents with the power of deliberative agents.
- Reading for this week:
 - M.Wooldridge: An introduction to MultiAgent Systems – Ch. 5 Reactive and Hybrid Architectures
 - A Robust Layered Control System for a Mobile Robot,” Rodney A. Brooks. IEEE Journal of Robotics and Automation, 2(1), March 1986, pp. 14–23. (also MIT AI Memo 864, September 1985)

“Inform” and “Request”

- “Inform” and “Request” are the two basic performatives in FIPA. Others are macro definitions, defined in terms of these.
- The meaning of inform and request is defined in two parts:
 - pre-condition: what must be true in order for the speech act to succeed.
 - “rational effect”: what the sender of the message hopes to bring about.

FIPA “Inform” Performative

The content is a **statement**. The pre-condition is that the sender:

- holds that the content is true;
- intends that the recipient believe the content;
- does not already believe that the recipient is aware of whether content is true or not

The speaker only has to believe that what he says is true.

FIPA “Request” Performative

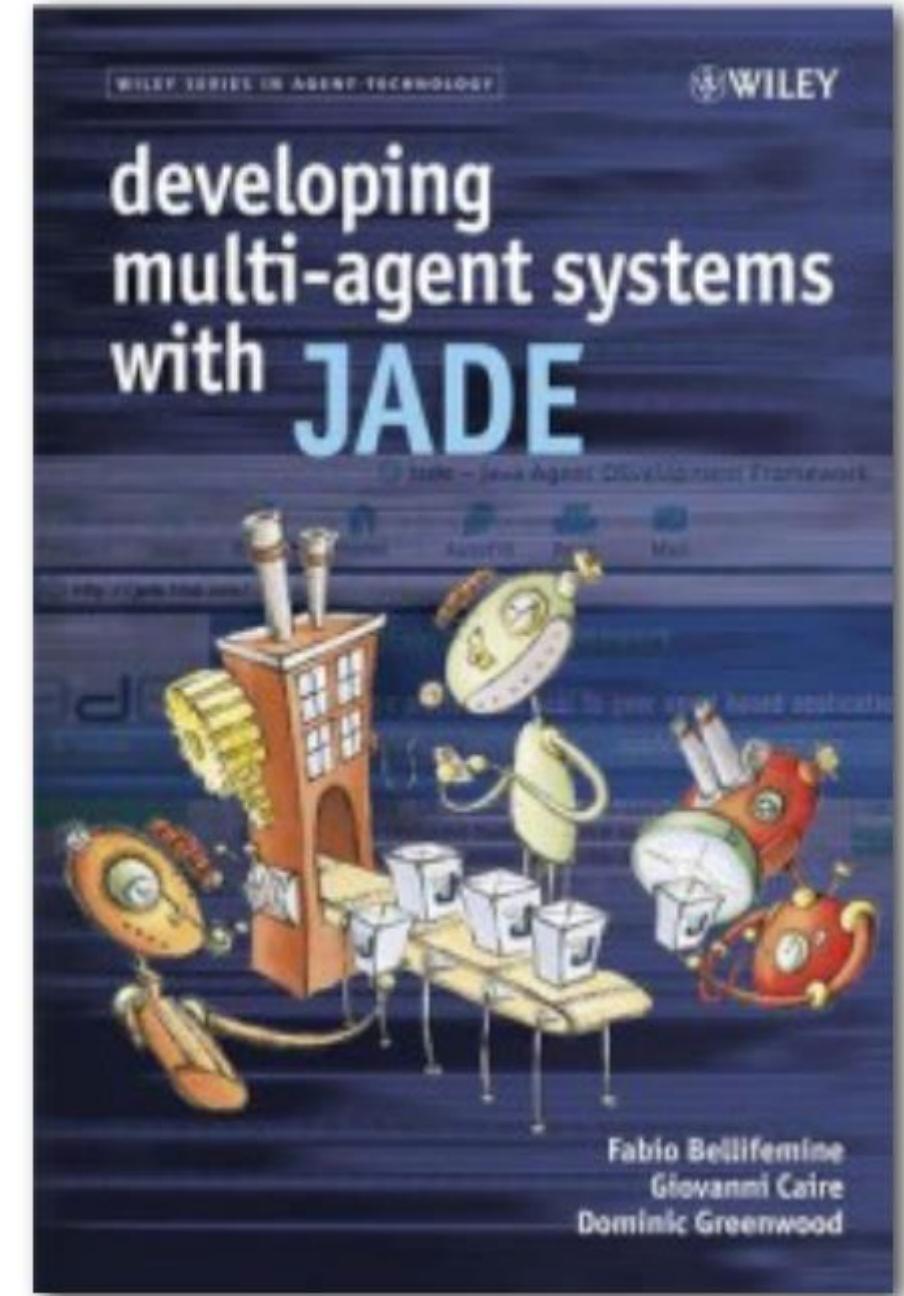
The content is an **action**. The pre-condition is that the sender:

- intends action content to be performed;
- believes recipient is capable of performing this action;
- does not believe that recipient already intends to perform action.

The last of these conditions captures the fact that you don't speak if you don't need to.

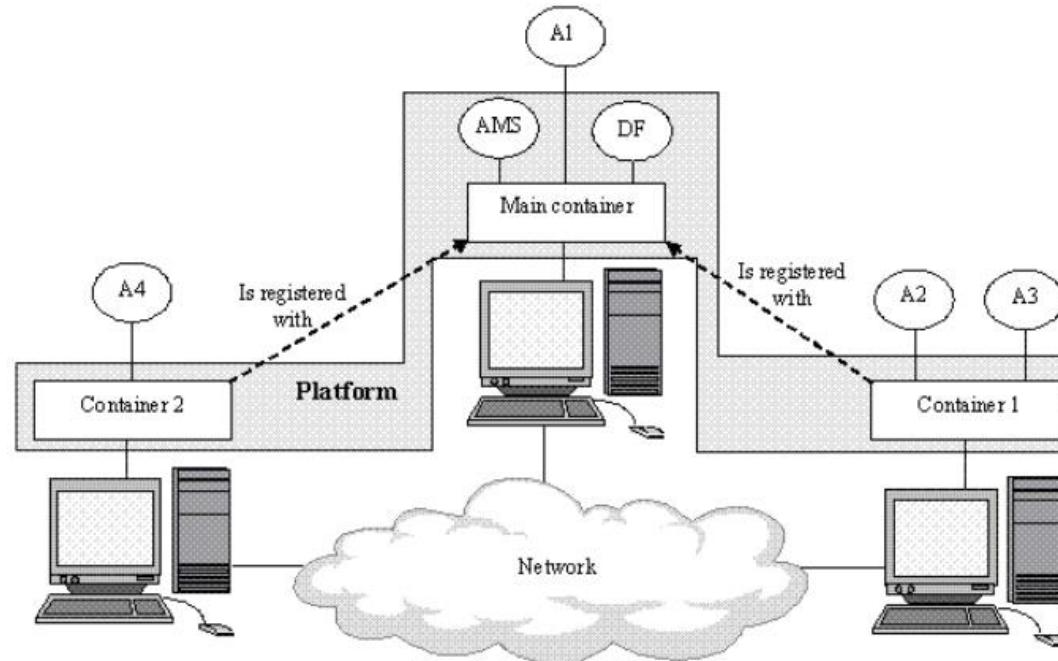
Jade

- The FIPA ACL provides a language for writing messages down.
 - It says nothing about how they are passed between agents.
- Several software platforms have been developed to support ACL-based communication.
 - One of the most widely used is JADE.
- Provides transparent (from the perspective of the agent designer) transport of ACL messages



Jade

- In JADE, agents are Java threads running in a “container”.



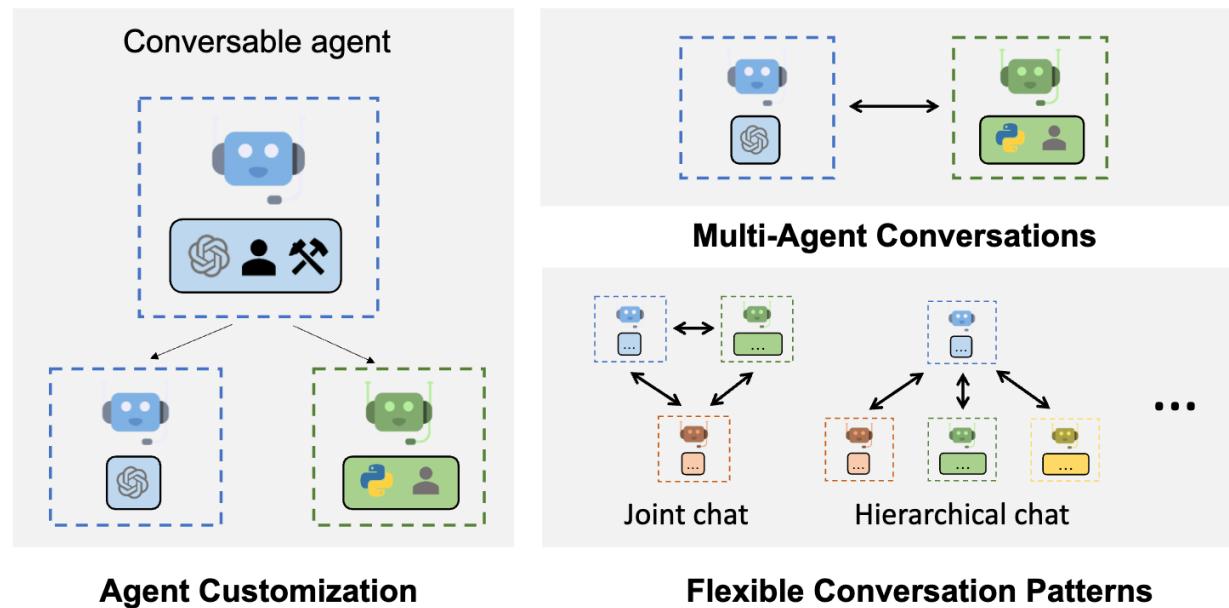
- All containers register with the main container

JADE Main Container

- The main container does the following:
 - Maintains the container table which lists all the containers and their contact information.
 - Maintains a list of all the agents in the system (including location and status).
 - Hosts the agent management system (AMS) which names agents as well as creating and destroying them.
 - Hosts the directory facilitator which provides a yellow pages allowing agents to be identified by the services they provide.
- See <http://jade.tilab.com/> for more details.

AutoGen

- AutoGen is a framework that enables the development of LLM applications using multiple agents that can converse with each other to solve tasks. AutoGen agents are customizable, conversable, and seamlessly allow human participation. They can operate in various modes that employ combinations of LLMs, human inputs, and tools.



- <https://github.com/microsoft/autogen>

Class Reading

- M.Wooldridge: An introduction to MultiAgent Systems – Ch. 6
Understanding Each Other
- M.Wooldridge: An introduction to MultiAgent Systems – Ch. 7
Communicating

LECTURE 6: Working Together

Introduction to Multi-Agent Systems (MESIIA, MIA)
URV

Working Together

- The term `cooperation` is frequently used in the concurrent systems.
- Two main distinctions between MAS and `traditional` distributed systems (DSs)
 - Agents in MAS may have been designed and implemented by different individuals with different goals. Agents act strategically to achieve the outcome they *most prefer*.
 - Ability to *dynamically* cooperate.
 - Agents are assumed to be acting autonomously
 - Coordination and cooperation in DS are typically hardwired in at design time



Agent Motivations

- Important to make a distinction between:
 - benevolent agents; and
 - self-interested agents

Agent Motivations

- Benevolent Agents
 - If we “own” the whole system, we can design agents to help each other whenever asked
 - In this case, we can assume agents are *benevolent*: our best interest is their best interest
 - Problem-solving in benevolent systems is *Cooperative Distributed Problem Solving* (CDPS)
 - Benevolence simplifies the system design task enormously!
 - We will talk about CDSP *in this lecture*
- Self Interested Agents
 - If agents represent the interests of individuals or organisations, (the more general case), then we cannot make the benevolence assumption
 - Agents will be assumed to act to *further their own interests*, possibly at the expense of others.
 - Potential for *conflict*
 - May complicate the design task enormously.
 - Strategic behaviour may be required — we will cover some of these aspects in *later lectures*

Cooperative Distributed Problem Solving

“... CDPS studies how a loosely coupled network of problem solvers can work together to solve problems that are beyond their individual capabilities. Each problem solving node in the network is capable of sophisticated problem-solving, and can work independently, but the problems faced by the nodes cannot be completed without cooperation. Cooperation is necessary because no single node has sufficient expertise, resources, and information to solve a problem, and different nodes might have expertise solving different parts of the problem....”

(Durfee et. al. 1989).

Coherence and Coordination

- Coherence:

- We can measure coherence in terms of solution quality, how efficiently resources are used, conceptual clarity and so on.

“... how well the [multiagent] system behaves as a unit along some dimension of evaluation...”

(Bond and Gasser, 1988).

- Coordination:

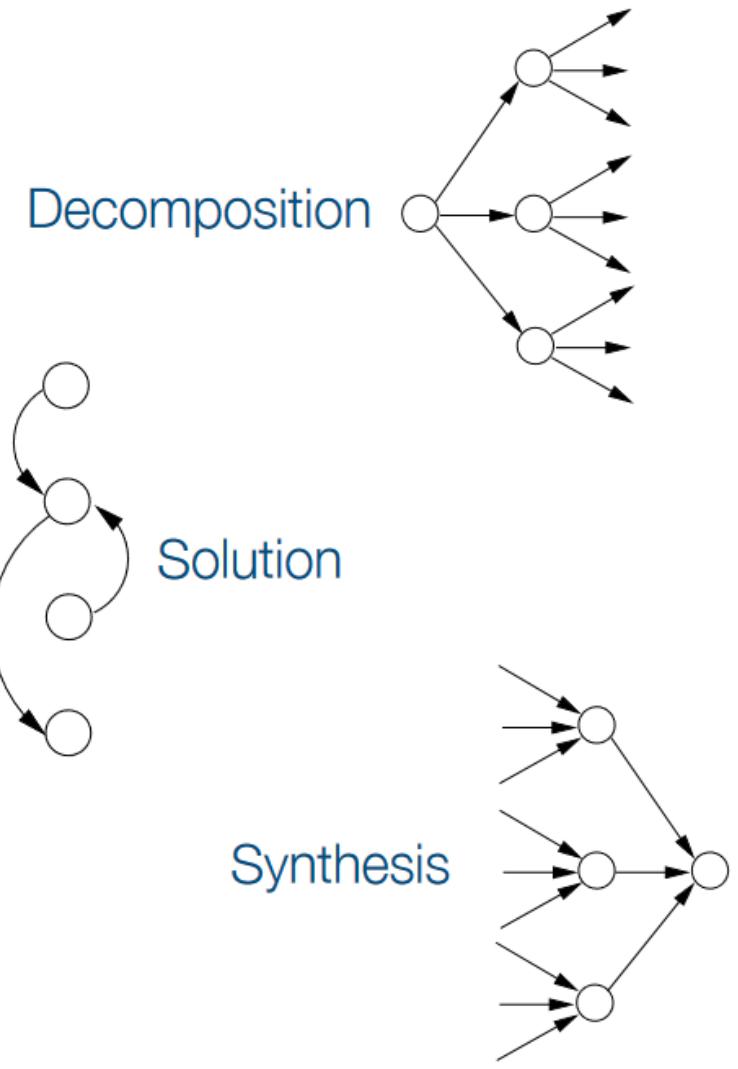
- If the system is perfectly coordinated, agents will not get in each others' way, in a physical or a metaphorical sense.

“... the degree. . . to which [the agents]. . . can avoid ‘extraneous’ activity [such as] . . . synchronizing and aligning their activities...”

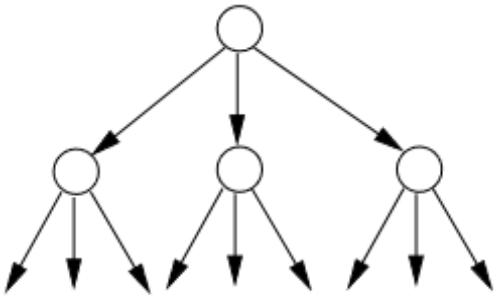
(Bond and Gasser, 1988).

Task Sharing and Result Sharing

- How does a group of agents work together to solve problems?
- CPDS addresses the following:
 - Problem decomposition
 - How can a problem be *divided into smaller tasks* for distribution amongst agents?
 - Sub-problem solution
 - How can the overall problem-solving activities of the agents be optimised so as to produce a solution that *maximises the coherence metric*?
 - What techniques can be used to coordinate the activity of the agents, *thus avoiding destructive interactions*?
 - Answer synthesis
 - How can a problem solution be effectively *synthesised from subproblem results*?



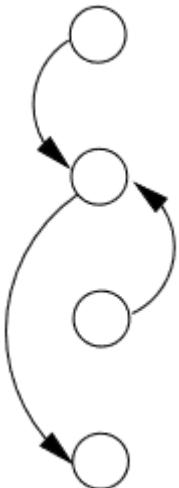
Problem Decomposition



- The overall problem is divided into smaller sub-problems.
 - *This is typically a recursive/hierarchical process.*
 - *Subproblems get divided up also.*
 - *The granularity of the subproblems is important*
 - *At one extreme, the problem is decomposed to atomic actions*
 - *In ACTORS, this is done until we are at the level of individual program instructions.*
- Clearly there is some processing to do the division.
 - How this is done is one design choice.
- Another choice is **who** does the division.
 - Is it centralised?
 - Which agents have knowledge of task structure?
 - Who is going to solve the sub-problems?

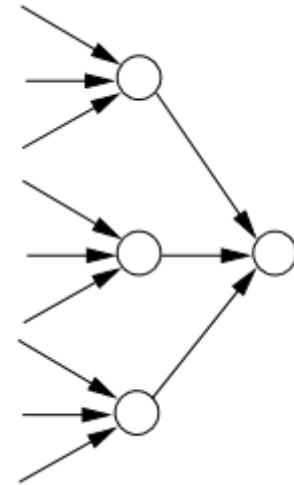
Sub-problem Solution

- The sub-problems derived in the previous stage are solved.
 - *Agents typically share some information during this process.*
- *A given step may involve two agents synchronising their actions.*
 - eg. *box pushing, safe opening*



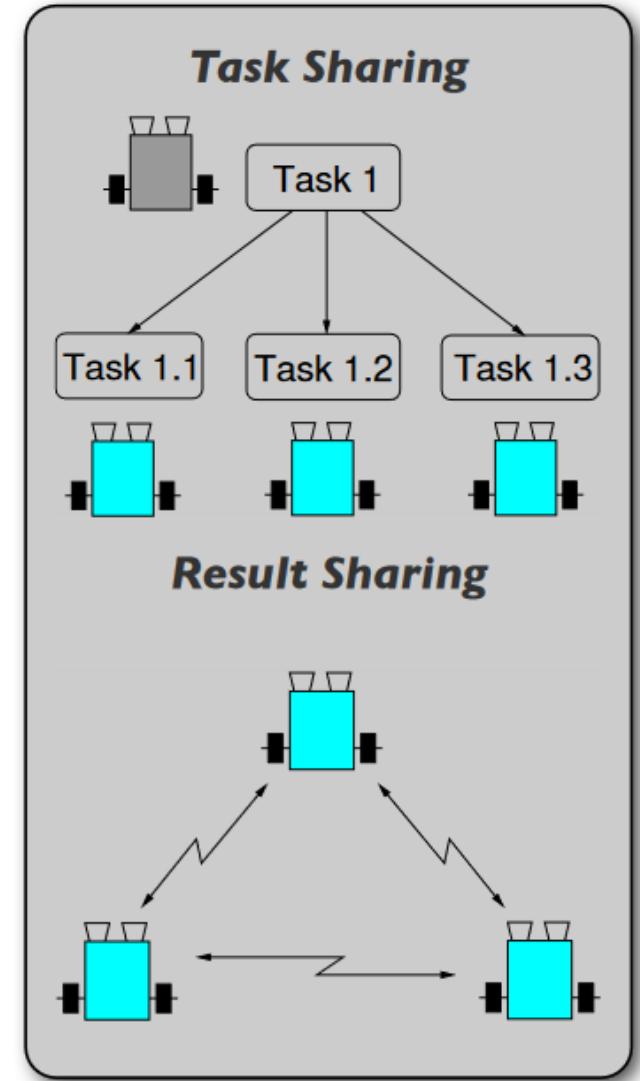
Solution Synthesis

- In this stage solutions to sub-problems are integrated.
 - Again this may be hierarchical
- Different solutions at different levels of abstraction.



Solution Synthesis

- Given this model of cooperative problem solving, we have two activities that are likely to be present:
 - Task Sharing:
 - components of a task are distributed to component agents;
 - how do we decide how to allocate tasks to agents?
 - Result Sharing:
 - information (partial results etc) is distributed.
 - how do we assemble a complete solution from the parts?
- An agent may well need a solution to both these problems in order to be able to function in a CDPS environment.



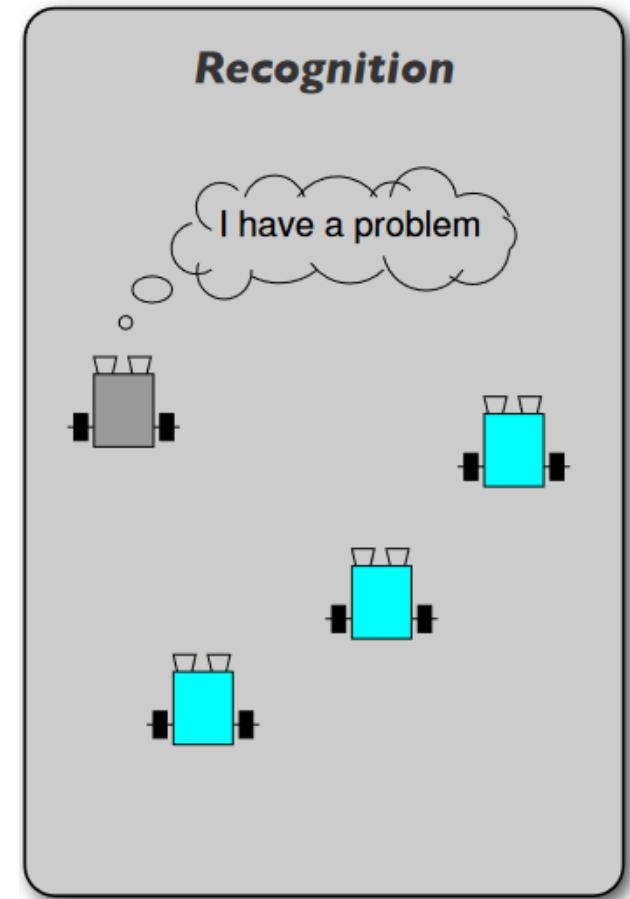
Task Sharing & the Contract Net

- Well known task-sharing protocol for task allocation is the ***contract net***.
- The contract net includes five stages:
 1. Recognition;
 2. Announcement;
 3. Bidding;
 4. Awarding;
 5. Expediting.



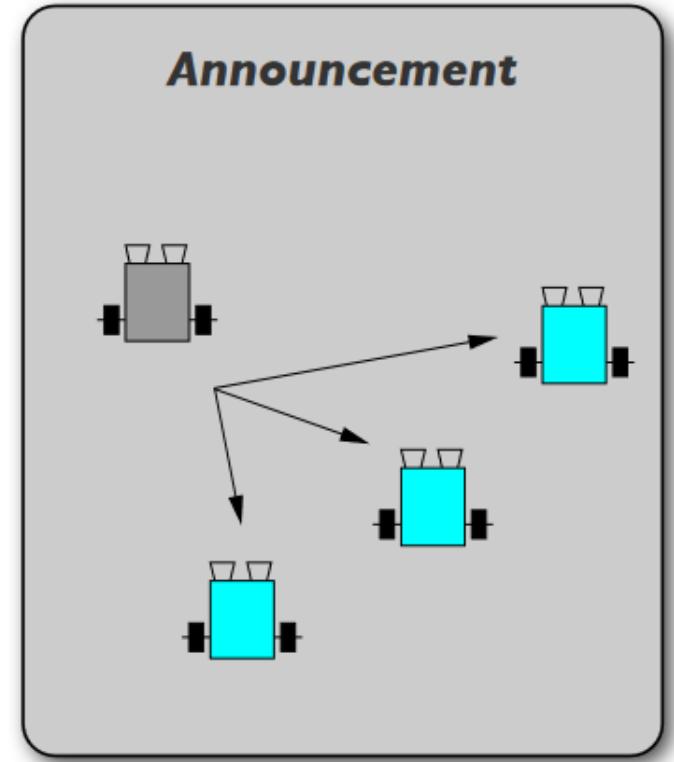
Recognition

- In this stage, an agent recognises it has a problem it wants help with.
- Agent has a goal, and either . . .
 - realises it cannot achieve the goal in isolation
 - i.e. it does not have capability;
 - realises it would prefer not to achieve the goal in isolation (typically because of solution quality, deadline, etc)
- As a result, it needs to involve other agents.



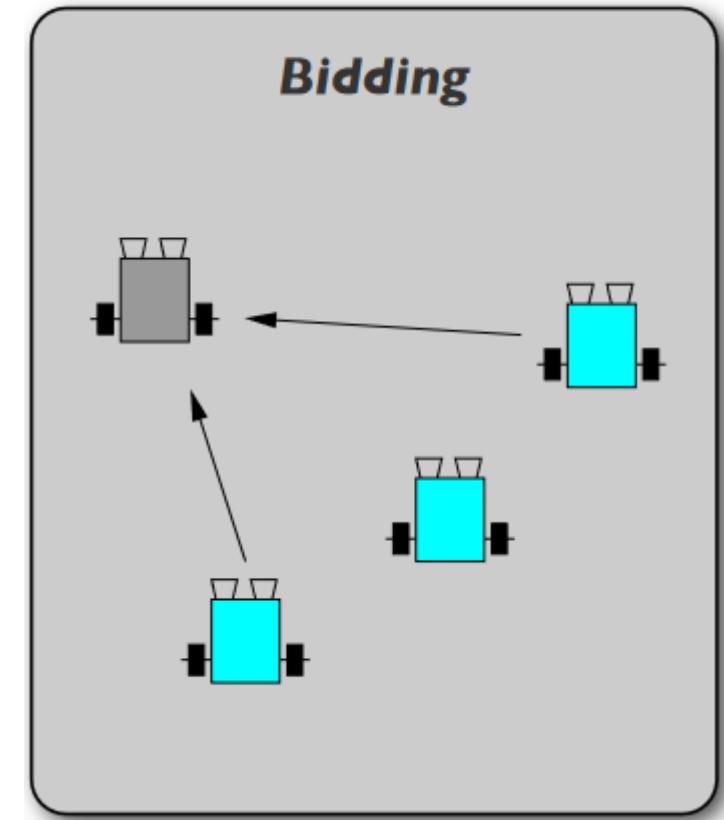
Announcement

- In this stage, the agent with the task sends out an announcement of the task which includes a specification of the task to be achieved.
- Specification must encode:
 - description of task itself (maybe executable)
 - any constraints (e.g., deadlines, quality constraints)
 - meta-task information (e.g., “ . . . bids must be submitted by . . . ”)
- The announcement is then broadcast



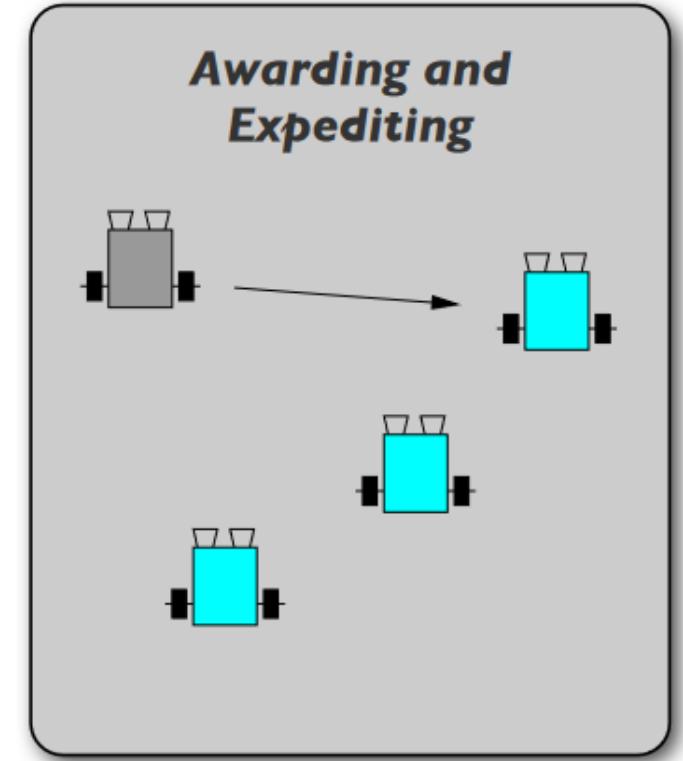
Bidding

- Agents that receive the announcement decide for themselves whether they wish to bid for the task.
- Factors:
 - agent must decide whether it is capable of expediting task;
 - agent must determine quality constraints & price information (if relevant).
- If they do choose to bid, then they **submit a tender**.



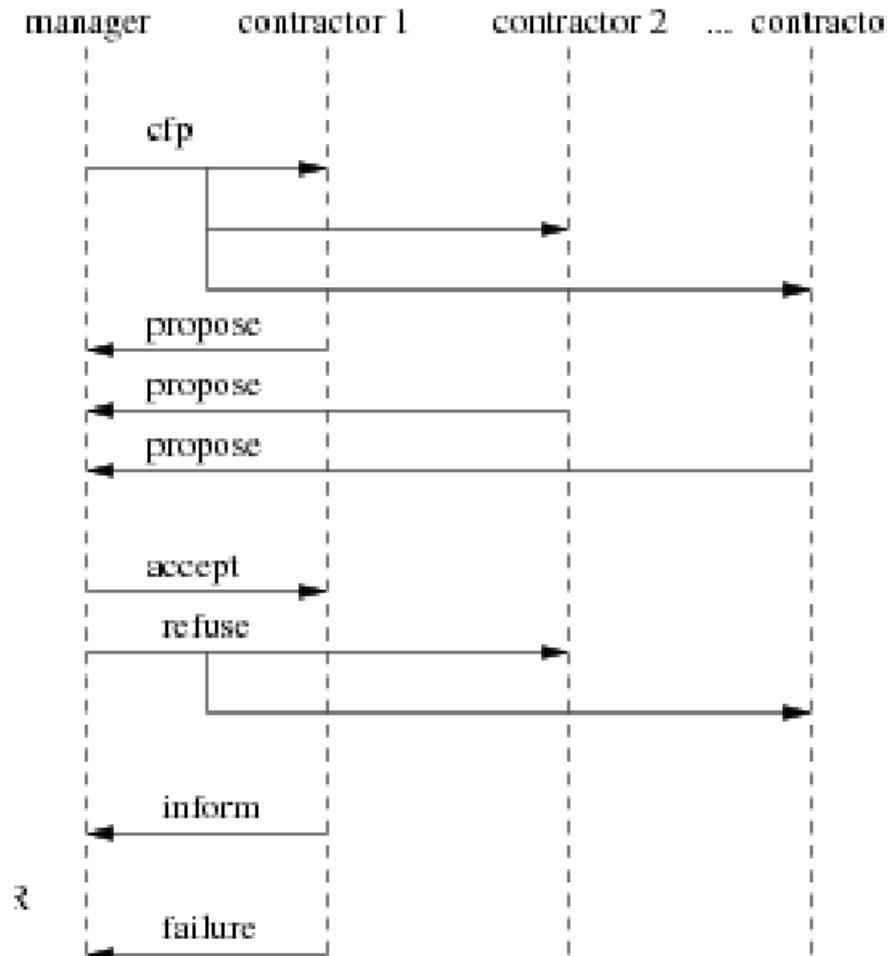
Awarding & Expediting

- Agent that sent task announcement must choose between bids & decide who to “award the contract” to.
 - The result of this process is communicated to agents that submitted a bid.
 - The successful contractor then expedites the task.
- May involve generating further manager-contractor relationships:
 - sub-contracting
 - May involve another contract net.



The Contract Net via FIPA Performatives

- The FIPA ACL was designed to be able to capture the contract net.
 - *cfp (call for proposals):*
 - Used for announcing a task;
 - *propose, refuse:*
 - Used for making a proposal, or declining to make a proposal.
 - *accept, reject:*
 - Used to indicate acceptance or rejection of a proposal.
 - *inform, failure:*
 - Used to indicate completion of a task (with the result) or failure to do so



Issues for Implementing Contract Net

- How to...
 - ... specify **tasks**?
 - ... specify **quality of service**?
 - ... decide how to **bid**?
 - ... select between competing offers?
 - ... differentiate between offers based on multiple criteria?



Deciding how to bid

- At some time t a contractor i is scheduled to carry out τ_i^t .
 - Contractor i also has resources e_i .
 - Then i receives an announcement of task specification ts , which is for a set of tasks $\tau(ts)$.
 - The cost to i to carry these out is: $c_i^t(\tau)$
- The **marginal cost** of carrying out τ will be

$$\mu_i(\tau(ts) \mid \tau^t_i) = c_i(\tau(ts) \cup \tau^t_i) - c_i(\tau^t_i)$$

- that is the difference between carrying out what it has already agreed to do and what it has already agreed plus the new tasks.

Deciding how to bid

- Due to synergies, this is often not just $c_i^t(\tau(ts))$
 - in fact, it can be zero — the additional tasks can be done for free.
- Think of the cost of giving another person a ride to work.
 - As long as $\mu_i(\tau(ts)|\tau_i^t) < e$ then the agent can afford to do the new work, then it is rational for the agent to bid for the work.
 - Otherwise not.
- You can extend the analysis to the case where the agent gets paid for completing a task.
 - And for considering the duration of tasks.

Results Sharing

- In results sharing, agents provide each other with information as they work towards a solution.
- It is generally accepted that results sharing improves problem solving by:
 - Independent pieces of a solution can be cross-checked.
 - Combining local views can achieve a better overall view.
 - Shared results can improve the accuracy of results.
 - Sharing results allows the use of parallel resources on a problem.
- The following are examples of results sharing.



Results Sharing in Blackboard Systems



- The first scheme for cooperative problem solving: was the blackboard system.
 - Results shared via shared data structure (BB).
 - Multiple agents (KSs/KAs) can read and write to BB.
 - Agents write partial solutions to BB.
- Blackboards may be structured as a hierarchy.
 - Mutual exclusion over BB required \Rightarrow bottleneck.
 - Not concurrent activity.

Result Sharing in Subscribe/Notify Pattern

- Common design pattern in OO systems: subscribe/notify.
 - An object *subscribes* to another object, saying “*tell me when event e happens*”
 - When event *e* happens, original object is notified.
- Information pro-actively *shared* between objects.
- Objects required to know about the *interests* of other objects ⇒ inform objects when relevant information arises.



The Centibots robots collaborate to map a space and find objects.



Handling Inconsistency

- A group of agents may have inconsistencies in their:
 - Beliefs
 - Goals or intentions
- Inconsistent beliefs arise because agents have different views of the world.
 - May be due to sensor faults or noise or just because they can't see everything.
- Inconsistent goals may arise because agents are built by different people with different objectives.

Handling Inconsistency

- Three ways to handle inconsistency (Durfee at al.)
 - Do not allow it!
 - For example, in the contract net the only view that matters is that of the manager agent.
 - Resolve inconsistency
 - Agents discuss the inconsistent information/goals until the inconsistency goes away.
- Build systems that degrade gracefully in the face of inconsistency.

Coordination

- Coordination is managing dependencies between agents.
 - *Any thoughts in resolving the following?*

1. We both want to leave the room through the same door. We are walking such that we will arrive at the door at the same time. What do we do to ensure we can both get through the door?
2. We both arrive at the copy room with a stack of paper to photocopy. Who gets to use the machine first?

Coordination

- Von Martialis suggested that positive coordination is:
 - Requested (explicit)
 - Non-requested (implicit)
- Non-requested coordination relationships can be as follows.
 - Action equality:
 - We both plan to do something, and by recognising this one of us can be saved the effort.
 - Consequence:
 - What I plan to do will have the side-effect of achieving something you want to do.
 - Favor:
 - What I plan to do will make it easier for you to do what you want to do.
- Now let's look at some approaches to coordination.

Social Norms

- Societies are often regulated by (often unwritten) rules of behaviour.
- Example:
 - A group of people is waiting at the bus stop. The bus arrives. Who gets on the bus first?
 - Another example:
 - On 34th Street, which side of the sidewalk do you walk along?
- In an agent system, we can design the norms and program agents to follow them, or let norms evolve.



Offline Design

- Recall how we described agents before:
 - As a function which, given a run ending in a state, gives us an action.
- A *constraint* is then a pair:
 - where $\acute{E} \subseteq E$ is a set of states, and $\alpha \in Ac$ is an action.
 - This constraint says that α cannot be done in any state in \acute{E} .
- A *social law* is then *a set of these constraints*.

$$Ag: R^E \rightarrow Ac$$

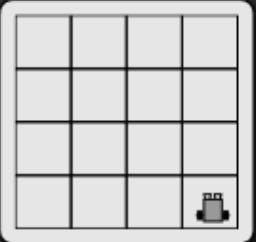
$$\langle E', \alpha \rangle$$

Offline Design

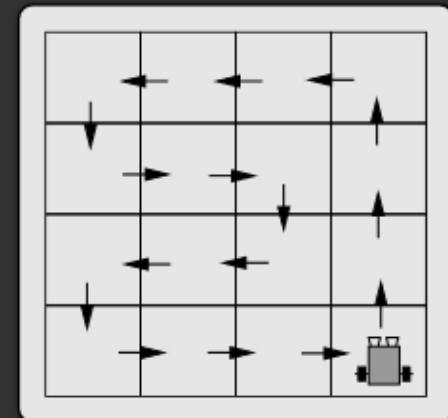
- We can refine our view of an environment.
 - Focal states, $F \subseteq E$ are the states we want our agent to be able to get to.
 - From any focal state $e \in F$ it should be possible to get to any other focal state $e' \in F$ (though not necessarily right away).
- A useful *social* law is then one that does not prevent agents from getting from one focal state to another.

A useful social law that prevents collisions

(Wooldridge p177, from Shoham and Tennenholz):



1. On even rows the robots move left while in odd rows the robots move right.
2. Robots move up when in the rightmost column.
3. Robots move down when in the leftmost column of even rows or the second rightmost column of odd rows.



Not necessarily efficient (On² steps to get to a specific square).

Emergence

- We can also design systems in which social laws emerge.

“... Agents have both a red t-shirt and a blue t-shirt and wear one. Goal is for everyone to end up with the same color on. In each round, each agent meets one other agent, and decides whether or not to change their shirt. During the round they only see the shirt their pair is wearing — they don’t get any other information...”

T-shirt Game (Shoham and Tennenholtz)

- What strategy update function should they use?

Simple majority:

Agents pick the shirt they have seen the most.

Simple majority with types:

Agents come in two types. When they meet an agent of the same type, agents pass their memories. Otherwise they act as simple majority.

Highest cumulative reward:

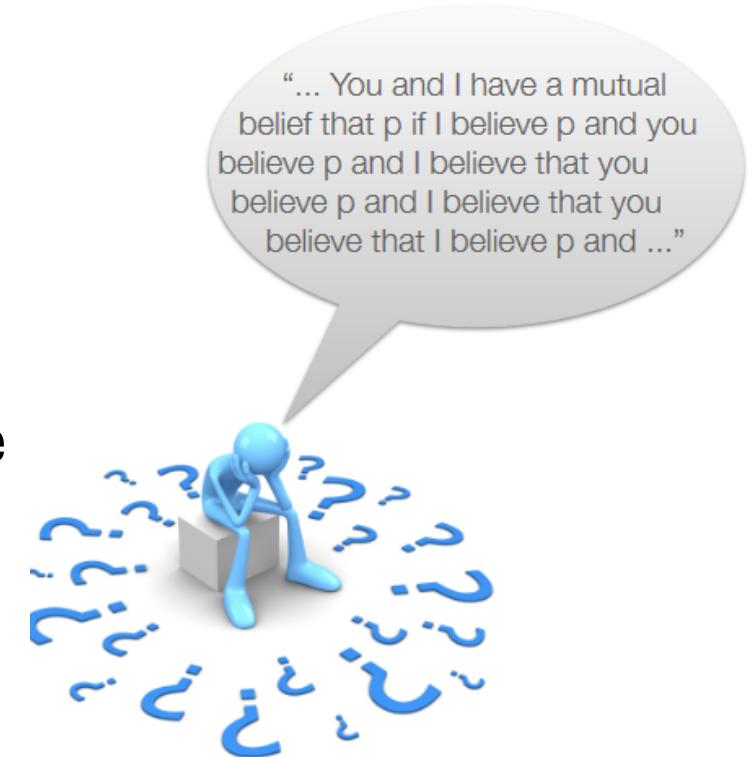
Agents can “see” how often other agents (some subset of all the agents) have matched their pair. They pick the shirt with the largest number of matches.

Joint Intentions

- Just as we have individual intentions, we can have joint intentions for a team of agents.
- Levesque defined the idea of *a joint persistent goal* (JPG).
 - A group of agents have a collective commitment to bring about some goal ϕ , “move the couch”.
 - Also have motivation ψ , “Simon wants the couch moved”.
- The mental states of agents mirror those in BDI agents.
 - Agents don’t believe that ϕ is satisfied, but believe it is possible.
 - Agents maintain the goal ϕ until a termination condition is reached.

Joint Intentions

- The terminations condition is that it is mutually believed that:
 - goal ϕ is satisfied; or
 - goal ϕ is impossible; or
 - the motivation ψ is no longer present
- The termination condition is achieved when an agent realizes that, the goal is satisfied, impossible and so on.
- But it doesn't drop the goal right away.
 - Instead it adopts a new goal — to make this new knowledge mutually believed.
 - This ensures that the agents are coordinated.
- They don't stop working towards the goal until they are all appraised of the situation.
 - Mutual belief is achieved by communication.



Multiagent Planning

- Another approach to coordinate is to explicitly plan what all the agents do.
 - For example, come up with a large STRIPS plan for all the agents in a system.
- Could have:
 - Centralised planning for distributed plans.
 - One agent comes up with a plan for everybody
 - Distributed planning
 - A group of agents come up with a centralised plan for another group of agents
 - Distributed planning for distributed plans
 - Agents build up plans for themselves, but take into account the actions of others.

Multiagent Planning

- In general, the more decentralized it is, the harder it is.
- Georgeff proposed a distributed version of STRIPS.
 - New list: during
 - Specifies what must be true while the action is carried out.
 - This places constraints on when other agents can do things.
- Different agents plan to achieve their goals using these operators and then do:
 - Interaction analysis: do different plans affect one another?
 - Safety analysis: which interactions are problematic?
 - Interaction resolution: treat the problematic interactions as critical sections and enforce mutual exclusion.

Readings for this week

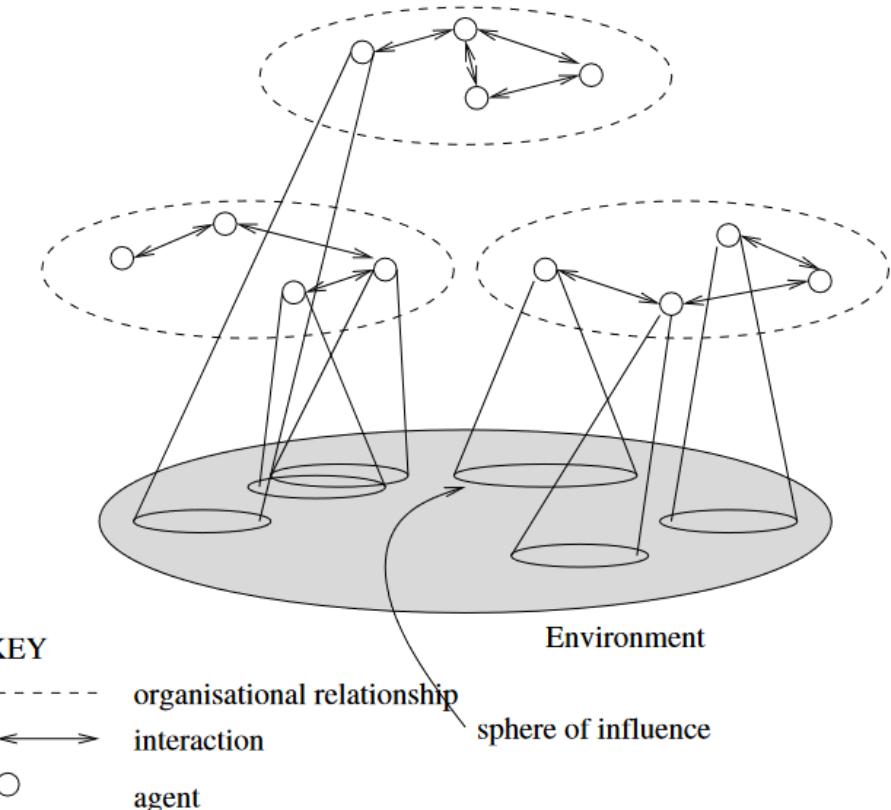
- M.Wooldridge: An introduction to MultiAgent Systems – Ch. 8 Working Together
- Distributed Problem Solving and Planning”, E.H. Durfee. In Weiss, G. ed.: Multiagent Systems, 1999, pp121-164.

LECTURE 7: Multiagent Decision Making (I)

Introduction to Multi-Agent Systems (MESIIA, MIA)
URV

What are Multi-Agent Systems?

- A multiagent system contains a number of agents that:
 - interact through communication;
 - are able to act in an environment;
 - have different “spheres of influence” (which may coincide); and
 - will be linked by other (organisational) relationships.



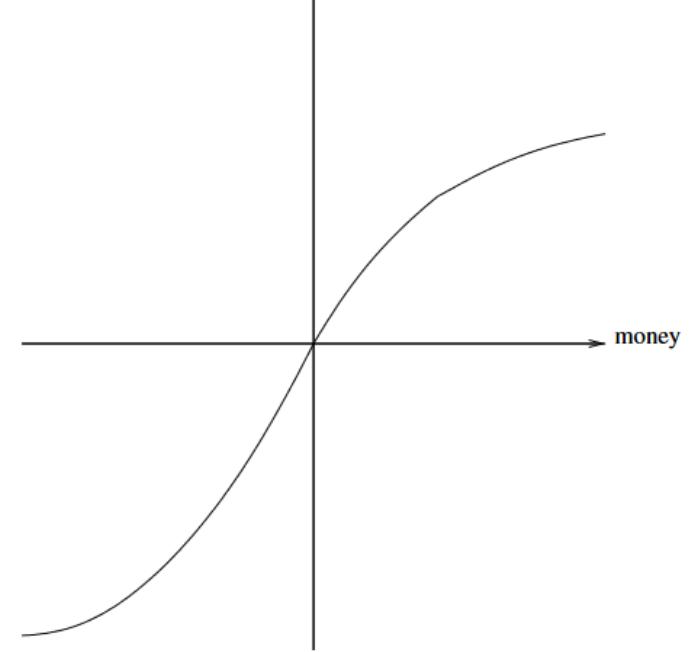
Types of Agreement

- Multiagent encounters (game-like character)
- Voting.
- Coalition forming.
- Allocating resources (Auctions)



Utilities and Preferences

- Our Assumptions:
 - Assume we have just two agents:
 - Agents are assumed to be self-interested i.e. they have preferences over how the environment is.
 - Assume $\Omega = \{\omega_1, \omega_2, \dots\}$ is the set of “outcomes” that agents have preferences over.
- We capture preferences by utility functions, represented as real numbers (\mathbb{R}):
- Utility functions lead to preference orderings over outcomes, e.g.:



Utility is not money. Just a way to encode preferences.

Multiagent Encounters

- We need a model of the environment in which these agents will act...
 - *agents simultaneously choose an action to perform, and as a result of the actions they select, an outcome in Ω will result*
 - *the actual outcome depends on the combination of actions*
 - *assume each agent has just two possible actions that it can perform:*
 - i.e. $A_c = \{C, D\}$, where
 - C (“cooperate”) and
 - D (“defect”)
- *Environment behaviour given by state transformer function τ*

$$\tau : \underbrace{A_c}_{\text{agent } i\text{'s action}} \times \underbrace{A_c}_{\text{agent } j\text{'s action}} \rightarrow \Omega$$

Multiagent Encounters

- Here is a state transformer function
 - This environment is sensitive to actions of both agents.
- With this state transformer, neither agent has any influence in this environment.
- With this one, the environment is controlled by j

$$\begin{array}{ll} \tau(D,D) = \omega_1 & \tau(D,C) = \omega_2 \\ \tau(C,D) = \omega_3 & \tau(C,C) = \omega_4 \end{array}$$

$$\begin{array}{ll} \tau(D,D) = \omega_1 & \tau(D,C) = \omega_1 \\ \tau(C,D) = \omega_1 & \tau(C,C) = \omega_1 \end{array}$$

$$\begin{array}{ll} \tau(D,D) = \omega_1 & \tau(D,C) = \omega_2 \\ \tau(C,D) = \omega_1 & \tau(C,C) = \omega_2 \end{array}$$

Rational Action

- Suppose we have the case where both agents can influence the outcome, and they have the following utility functions:

$$u_i(\omega_1) = 1 \quad u_i(\omega_2) = 1 \quad u_i(\omega_3) = 4 \quad u_i(\omega_4) = 4$$

$$u_j(\omega_1) = 1 \quad u_j(\omega_2) = 4 \quad u_j(\omega_3) = 1 \quad u_j(\omega_4) = 4$$

- With a bit of abuse of notation:

$$u_i(D,D) = 1 \quad u_i(D,C) = 1 \quad u_i(C,D) = 4 \quad u_i(C,C) = 4$$

$$u_j(D,D) = 1 \quad u_j(D,C) = 4 \quad u_j(C,D) = 1 \quad u_j(C,C) = 4$$

- Then agent i 's preferences are $(C, C) \geq_i (C, D) >_i (D, C) \geq_i (D, D)$
- In this case, what should i do?
- i prefers all outcomes that arise through C over all outcomes that arise through D .
 - Thus, C is the *rational choice* for i .

Payoff Matrices

- We can characterise the previous scenario in a payoff matrix shown opposite
 - Agent i is the column player and gets the upper reward in a cell.
 - Agent j is the row player and gets the lower reward in a cell.
- Actually there are two matrices here, one (call it A) that specifies the payoff to i and another B that specifies the payoff to j.
- Sometimes we'll write the game as (A, B) in recognition of this.

In this case, *i* cooperates and gains a utility of 4; whereas *j* defects and gains a utility of only 1.

		i	
		defect	coop
j	defect	1	4
	coop	1	4

$$(C, C) \geq_i (C, D) >_i (D, C) \geq_i (D, D)$$

Solution Concepts

- How will a rational agent will behave in any given scenario?
- *Play*. . .
 - *Dominant strategy*;
 - *Nash equilibrium strategy*;
 - *Pareto optimal strategies*;
 - *Strategies that maximise social welfare*.

Dominant Strategies

- Given any particular strategy s (either C or D) that agent i can play, there will be a number of possible outcomes.
 - We say s_1 dominates s_2 if every outcome possible by i playing s_1 is preferred over every outcome possible by i playing s_2 .
- Thus in the game opposite, C dominates D for both players.

		i	
		defect	coop
j	defect	1	4
	coop	1	4
		4	4

Dominant Strategies

- A rational agent will never play a *dominated strategy*.
 - i.e, a strategy that is dominated (and thus inferior) by another.
- So in deciding what to do, we can delete dominated strategies.
 - *Unfortunately*, there isn't always a unique undominated strategy.

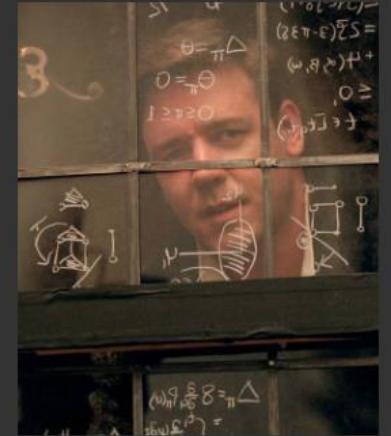
		<i>i</i>	
		defect	coop
<i>j</i>	defect	1	4
	coop	1	4
		4	4

Nash Equilibrium

- In general, we will say that two strategies s_1 and s_2 are in Nash equilibrium (NE) if:
 - under the assumption that agent i plays s_1 , agent j can do no better than play s_2 ;
 - I.e. if **I** drive on the right side of the road, **you** can do no better than also driving on the right!
 - under the assumption that agent j plays s_2 , agent I can do no better than play s_1 .
 - I.e. if **you** drive on the right side of the road, **I** can do no better than also driving on the right!
- Neither agent has any incentive to deviate from a Nash Equilibrium (NE).

John Forbes Nash

(Nobel Laureate in Economics)



Portrayed by Russel Crowe in the film
“A Beautiful Mind”

Nash Equilibrium

- Consider the payoff matrix opposite:
 - Here the Nash equilibrium (NE) is (D, D).
 - In a game like this you can find the NE by cycling through the outcomes, asking if either agent can improve its payoff by switching its strategy.
- Thus, for example, (C, D) is not a NE because i can switch its payoff from 1 to 5 by switching from C to D.

		<i>i</i>	
		defect	coop
<i>j</i>	defect	5	1
	coop	3	0

		<i>i</i>	
		defect	coop
<i>j</i>	defect	5	1
	coop	3	0

Nash Equilibrium

- Not every interaction scenario has a pure strategy Nash Equilibrium (NE).
 - The game opposite (upper) has two pure strategy NEs, (C, C) and (D, D)
- Some interaction scenarios have more than one pure strategy Nash Equilibrium (NE).
 - The game opposite has no pure strategy NE
 - For every outcome, one of the agents will improve its utility by switching its strategy.

		<i>i</i>	defect	coop
		<i>j</i>	defect	coop
defect	defect	5	1	
	coop	3	2	3
coop	defect	0	3	
	coop	2	3	3

		<i>i</i>	defect	coop
		<i>j</i>	defect	coop
defect	defect	2	1	
	coop	1	2	
coop	defect	0	1	
	coop	2	1	

Mixed Strategy Nash equilibrium

- Matching Pennies
 - Players i and j simultaneously choose the face of a coin, either “heads” or “tails”.
 - If they show the same face, then i wins, while if they show different faces, then j wins.
- NO pair of strategies forms a pure strategy NE:
 - whatever pair of strategies is chosen, somebody will wish they had done something else
- The solution is to allow mixed strategies:
 - play “heads” with probability 0.5
 - play “tails” with probability 0.5.
- This is a Mixed Nash Equilibrium strategy.

		<i>i</i>	
		heads	tails
<i>j</i>	heads	1	-1
	tails	-1	1
	heads	-1	1
	tails	1	-1



Mixed Strategy Nash equilibrium

- Consider the Game Rock/Paper/Scissors
 - Paper covers rock
 - Scissors cut paper
 - Rock blunts scissors
- This has the following payoff matrix

		<i>i</i>		
		rock	paper	scissors
<i>j</i>	rock	0	1	0
	paper	0	0	1
<i>j</i>	paper	0	0	1
	scissors	1	0	0
		0	1	0

- What should you do?
 - Choose a strategy at random!



Mixed Strategies

- A mixed strategy has the form
 - play a_1 with probability p_1
 - play a_2 with probability p_2
 - ...
 - play a_k with probability p_k .
 - such that $p_1 + p_2 + \dots + p_k = 1$.

Nash's Theorem

Nash proved that every finite game has a Nash equilibrium in mixed strategies. (Unlike the case for pure strategies.)

So this result overcomes the lack of solutions; but there still may be more than one Nash equilibrium. . .

Pareto Optimality

- An outcome is said to be Pareto optimal (or Pareto efficient) if:
 - there is no other outcome that makes one agent better off without making another agent worse off.*
 - If an outcome is Pareto optimal, then at least one agent will be reluctant to move away from it (because this agent will be worse off).*
- “Reasonable” agents would agree to move to ω' in this case.
 - Even if I don't directly benefit from ω' , you can benefit without me suffering.*

This game has one Pareto efficient outcome: (D, D)

	<i>i</i>	defect	coop
<i>j</i>	defect	5 3	1 2
	coop	0 2	0 1

There is no solution in which either agent does better

Social Welfare

- The social welfare of an outcome ω is the sum of the utilities that each agent gets from ω :

$$\sum_{i \in Ag} u_i(\omega)$$

- Think of it as the “total amount of money in the system”.
- As a solution concept:
 - may be appropriate when the whole system (all agents) has a single owner (then overall benefit of the system is important, not individuals)
 - It doesn’t consider the benefits to individuals.
 - A very skewed outcome can maximise social welfare.

In both these games, (C, C) maximises social welfare

		<i>i</i>	
		defect	coop
<i>j</i>	defect	2	1
	coop	3	4

		<i>i</i>	
		defect	coop
<i>j</i>	defect	2	1
	coop	3	7

The Prisoner's Dilemma

- Payoff matrix for prisoner's dilemma:

		<i>i</i>	
		defect	coop
<i>j</i>	defect	2	1
	coop	4	3

- Top left: If both defect, then both get punishment for mutual defection.
- Top right: If i cooperates and j defects, i gets sucker's payoff of 1, while j gets 4.
- Bottom left: If j cooperates and i defects, j gets sucker's payoff of 1, while i gets 4.
- Bottom right: Reward for mutual cooperation (i.e. neither confess)

The Prisoner's Dilemma

Two men are collectively charged with a crime and held in separate cells, with no way of meeting or communicating. They are told that:

- if one confesses and the other does not (C,D) or (D,C), the confessor will be freed, and the other will be jailed for three years;
- if both confess (D,D), then each will be jailed for two years.

Both prisoners know that if neither confesses (C,C), then they will each be jailed for one year.

What should you do?

- The *individual rational action* is defect.
 - This guarantees a payoff of no worse than 2, whereas cooperating guarantees a payoff of at most 1.
 - So defection is the best response to all possible strategies: both agents defect, and get payoff = 2.
- But intuition says this is not the best outcome:
 - Surely they should both cooperate and each get payoff of 3!
- This is why the Prisoners Dilemma game is Interesting
 - The analysis seems to give us a *contradictory* answer.

Solution Concepts

- The dominant strategy here is to defect.
- (D, D) is the only Nash equilibrium.
- All outcomes **except** (D, D) are Pareto optimal.
- (C, C) maximises social welfare.

		<i>i</i>	
		defect	coop
<i>j</i>	defect	2	1
	coop	4	3
		1	3

The Prisoner's Dilemma

- This apparent contradiction is the fundamental problem of multi-agent interactions.
 - It appears to imply that cooperation will not occur in societies of self-interested agents.
- The prisoner's dilemma is ubiquitous.
 - Can we recover cooperation?

Solution Concepts

- The dominant strategy here is to defect.
- (D, D) is the only Nash equilibrium.
- All outcomes **except** (D, D) are Pareto optimal.
- (C, C) maximises social welfare.

		<i>i</i>	
		defect	coop
<i>j</i>	defect	2	1
	coop	4	3
		1	3

Arguments for Recovering Cooperation

- Conclusions that some have drawn from this analysis:
 - the game theory notion of rational action is wrong!
 - somehow the dilemma is being formulated wrongly
- Arguments to recover cooperation:
 - We are not all machiavelli!
 - The other prisoner is my twin!
 - ***Program equilibria and mediators***
 - The shadow of the future. . .

Program Equilibria

- The strategy you really want to play in the prisoner's dilemma is: *I'll cooperate if he will*
 - Program equilibria provide one way of enabling this.
- Each agent submits a *program strategy* to a *mediator* which *jointly executes* the strategies.
 - Crucially, strategies can be *conditioned on the strategies of the others*.
- The best response to this program:
 - *submit the same program*, giving an outcome of (C, C)!

Player 1 (P1)

```
If (P1 == P2) {  
    do(C)  
} else {  
    do(D)  
}  
stop
```

Player 2 (P2)

```
If (P1 == P2) {  
    do(C)  
} else {  
    do(D)  
}  
stop
```

Mediator

P1:C P2:C

Player 1 (P1)

```
If (P1 == P2) {  
    do(C)  
} else {  
    do(D)  
}  
stop
```

Player 2 (P2)

```
do(D)  
stop
```

Mediator

P1:D P2:D

Social Choice

- *Social choice theory* is concerned with group decision making.
 - Agents make decisions based on their preferences, but they are aware of other agents' preferences as well.
- Classic example of social choice theory: *voting*
 - Formally, the issue is combining preferences to *derive a social outcome*.

Components of a Social Choice Model

- Assume a set $A_g = \{1, \dots, n\}$ of *voters*.
 - These are entities who express preferences.
 - Voters make group decisions with respect to a set $\Omega = \{\omega_1, \omega_2, \dots\}$ of *outcomes*.
 - Think of these as the *candidates*.
 - If $|\Omega| = 2$, we have a pairwise election.
- Each voter has preferences over Ω
 - An ordering over the set of possible outcomes Ω .
 - Sometimes we will want to pick one, most preferred candidate.
 - More generally, we may want to rank, or order these candidates.

Preference Order Example

Suppose

$$\Omega = \{\text{pear, plum, banana, orange}\}$$

then we might have agent i with preference order:

$$(\text{banana, plum, pear, orange})$$

meaning

$$\text{banana} >_i \text{plum} >_i \text{pear} >_i \text{orange}$$

Preference Aggregation

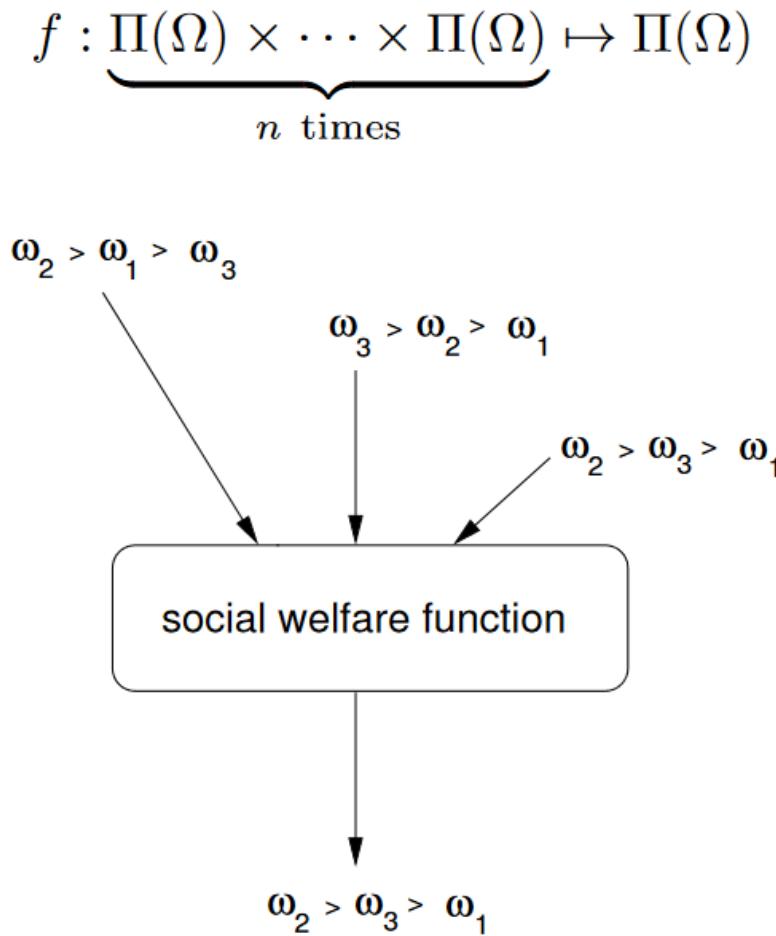
- The fundamental problem of social choice theory is that...
 - ***...different voters typically have different preference orders!***

“... given a collection of preference orders, one for each voter, how do we combine these to derive a group decision, that reflects as ***closely as possible*** the preferences of voters? ...”

- We need a way to combine these opinions into one overall decision.
 - What social choice theory is about is finding a way to do this.
 - Two variants of preference aggregation:
 - social welfare functions
 - social choice functions

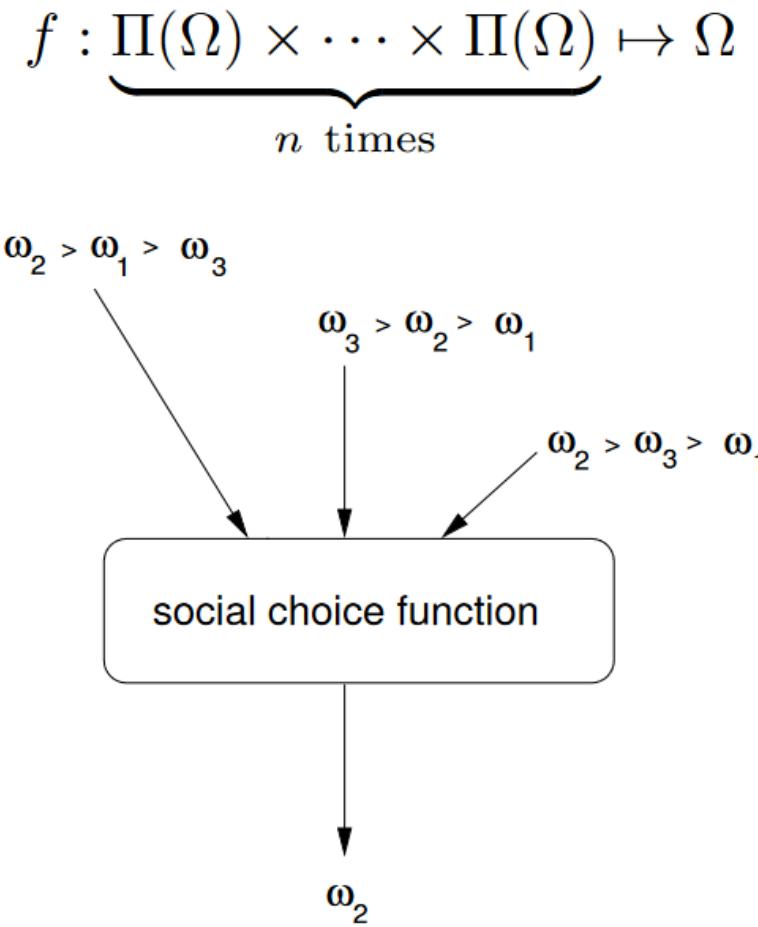
Social Welfare Function

- Let $\Pi(\Omega)$ be a set of preference orderings over Ω
 - A *social welfare function* takes voter preferences and produces a *social preference order*.
 - That is, it merges voter opinions and comes up with an order over the candidates.
- We let $>*$ denote the outcome of a social welfare function: $\omega >* \omega'$
 - which indicates that ω is ranked above ω' in the social ordering
 - Example: combining search engine results, collaborative filtering, collaborative planning, etc.



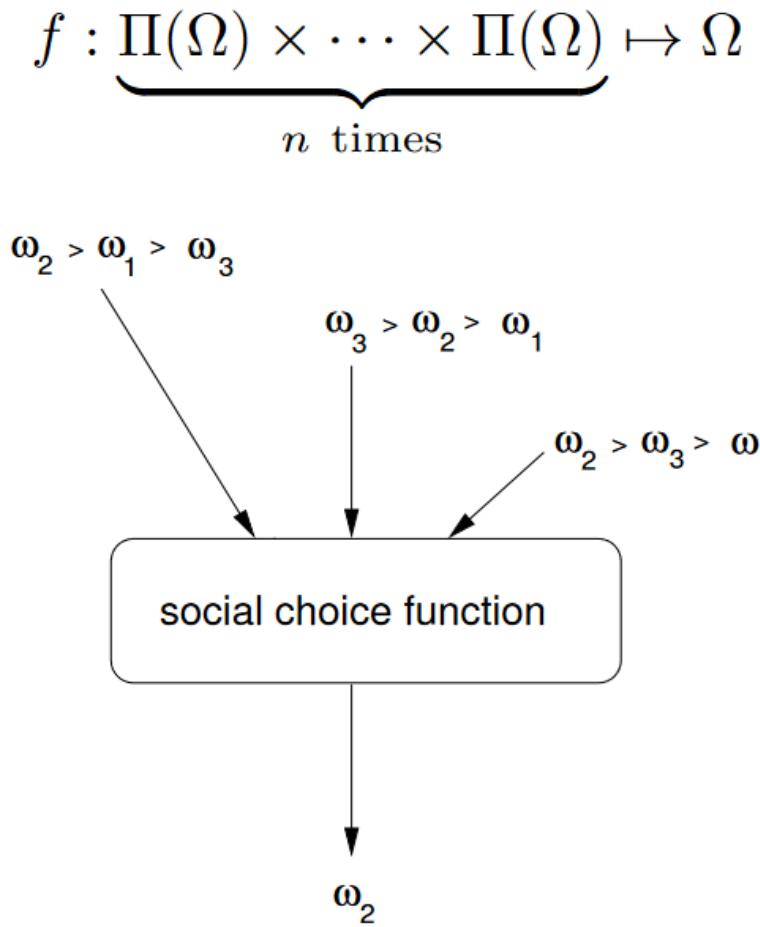
Social Choice Function

- Sometimes, we just one to select **one** of the possible candidates, rather than a social order.
 - This gives a *social choice function* (see opposite)
- In other words, we don't get an ordering out of a social choice function but, as its name suggests, we get a *single choice*.
 - Of course, if we have a social welfare function, we also have a social choice function.
- For the rest of this lecture...
 - ...we'll refer to both social choice and social welfare functions as *voting procedures*.



Desirable Properties of the Social Choice Function

- **Calculability**
 - A social preference ordering $<^*$ should exist for all possible inputs.
- **Completeness**
 - $<^*$ should be defined for every pair of alternatives $(\omega, \omega') \in \Omega$
- **Linearity**
 - $<^*$ should be antisymmetric and transitive over Ω
- **Anonymity / No dictatorship**
 - The outcome of the social choice rule depends on the set of opinions, but not on which agents have these opinions.



Desirable Properties of the Social Choice Function

- Unanimity / Pareto efficiency

- Do not misorder the options if all agents agree.
- If everybody thinks that A is better than B, A should be preferred to B in the aggregated order.

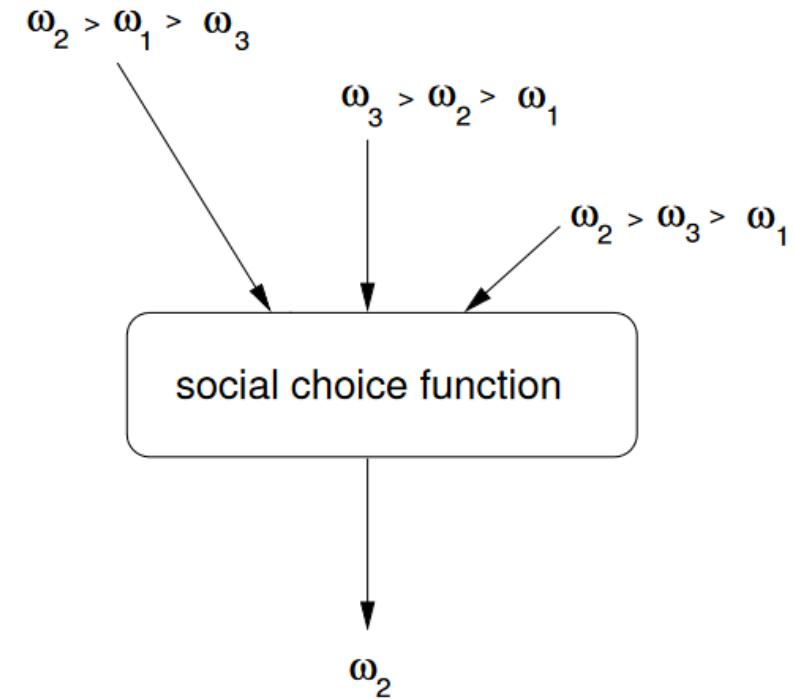
- Neutrality

- The outcome of the social choice function should not depend on how alternatives are named or ordered.

- Independence of irrelevant alternatives

- Removing / Adding an irrelevant alternative should not affect the winner of the vote

$$f : \underbrace{\Pi(\Omega) \times \cdots \times \Pi(\Omega)}_{n \text{ times}} \mapsto \Omega$$



Simple Voting Procedures

- Some basic voting mechanisms
 - Plurality / Anti-plurality / Best-Worst / Approval
- Protocols based on total orders
 - Binary protocol (series of votes of 2 options each)
 - Borda protocol (sum of all the preferences of the agents)
 - Condorcet protocol (pairwise comparison of options, given full preference ordering of each agent)
- All the procedures are problematic in one sense or another.

Plurality Voting Procedure

- Social choice function: selects a single outcome.
 - Each agent (i.e., voter) can give **1 vote to 1 of the alternatives**
 - The alternative with the **highest number of votes** wins
- If we have only two candidates, then plurality is a ***simple majority election***

Vote for one option.

<input type="checkbox"/>	Joe Smith
<input checked="" type="checkbox"/>	John Citizen
<input type="checkbox"/>	Jane Doe
<input type="checkbox"/>	Fred Rubble
<input type="checkbox"/>	Mary Hill

Anomalies with Plurality

- Suppose:
 - $|A_g| = 100$ and $\Omega = \{\omega_1, \omega_2, \omega_3\}$
- with:
 - 40% voters voting for ω_1
 - 30% of voters voting for ω_2
 - 30% of voters voting for ω_3
- With plurality, ω_1 gets elected even though a clear majority (60%) prefer another candidate

Advantages of plurality voting

- Most simple voting mechanism
- Very efficient from the computational point of view
- Equality principle, as it preserves the idea of *1 agent = 1 vote*

Another version: anti-plurality

- Each voter gives a *negative* vote to the alternative they consider the worst
- The option with *less* votes *wins*
- Example:
 - 30% CBDA
 - 30% CADB
 - 20% ABDC
 - 20% BADC
 - C gets 40% negative votes but also first option for 60%
 - A and B get 30% negative votes
 - D is the winner with 0 negative votes – but it was not the first or second option for anyone

Best-worst voting systems

- Each agent gives a *positive* vote to his best alternative and a *negative* vote to his worst alternative
- Each alternative receives $\alpha > 0$ points for each positive vote and $-\delta < 0$ points for each negative vote
- The option with more points wins

	1	2	3	4	5	6	7
C	C	C	B	B	A	A	A
B	B	B	C	C	B	C	C
A	A	A	A	A	C	B	B

$$A \rightarrow 3\alpha - 4\delta \text{ points} \quad B \rightarrow 2\alpha - 2\delta \text{ points} \quad C \rightarrow 2\alpha - \delta \text{ points}$$

Plurality ($\delta = 0$) $A \succ B \sim C$

Antiplurality ($\alpha = 0$) $C \succ B \succ A$

$\alpha = \delta = 1$ $C \succ B \succ A$

$\alpha = 2$ and $\delta = 1$ $C \succ A \sim B$

$\alpha = 4$ and $\delta = 1$ $A \succ C \succ B$

Approval voting

- Each voter selects a *subset* of the candidates
- The candidate with *most votes wins*
- *k-approval* voting
 - Each voter selects a subset of k candidates
 - k=1: plurality
 - k= n-1: anti-plurality

Voting Procedures based on Linear Orders

- Each voter gives a full list of the options, ordered according to his preferences (from best to worst)
- A voter prefers option i to option j if option i appears before option j in his list

Binary Procedure

- All the options are ordered and then evaluated in pairs (options 1 and 2, the winner with option 3, the winner with option 4, etc.)
- Simple majority: option A is better than option B if and only if the number of voters that prefer A to B is greater than the number of voters that prefer B to A
- The option that wins the last evaluation is the overall winner
 - $\text{win}(\text{a5}, \text{win}(\text{a4}, \text{win}(\text{a3}, \text{win}(\text{a2}, \text{a1}))))$

The Ordering Problem

- Example:
 - $x > z > y$ (35%)
 - $y > x > z$ (33%)
 - $z > y > x$ (32%)
- Note that y is preferred to x (65-35), x is preferred to z (68-32), and z is preferred to y (67-33)
 - $\text{win}(x, \text{win}(y,z))=x$
 - $\text{win}(y,\text{win}(x,z))=y$
 - $\text{win}(z, \text{win}(x,y))=z$
- The order of the pairings affects the outcome !
 - The voter organiser may influence the result
 - The last options have more chances of winning
 - *No Neutrality*

Another problematic example

35% of agents have preferences $c > d > b > a$

33% of agents have preferences $a > c > d > b$

32% of agents have preferences $b > a > c > d$

- Evaluation in the order abcd:
 - $\text{Win}(a,c)=a$ $\text{Win}(a,b)=b$ $\text{Win}(b,d)=d \Rightarrow d$ Wins
 - d was the worst alternative for 32%
 - d was not the best alternative for anyone
 - Everybody prefers c to d (!) – No Unanimity

Summary of problems of binary voting

- Decisive role of the ordering of the alternatives
- An alternative x may win even if there is another alternative x' which is preferred to x by all agents
 - Alternatives may be misordered
- Temporal cost of the voting process
 - sequence of pairwise eliminative votes

Borda Procedure

- One reason plurality has so many anomalies is that it ignores most of a voter's preference orders: it only looks at the top ranked candidate.
 - The Borda count takes whole preference order into account.
- Suppose we have k candidates - i.e. $k = |\Omega|$
 - For each candidate, we have a variable, counting the strength of opinion in favour of this candidate.
 - If ω_i appears first in a preference order, then we increment the count for ω_i by $k - 1$;
 - we then increment the count for the next outcome in the preference order by $k - 2$,
 - . . . , until the final candidate in the preference order has its total incremented by 0.
- After we have done this for all voters, then the totals give the ranking.

Example of Borda Count

Assume we have three voters with preferences:

$$\begin{array}{l} \omega_2 >_1 \omega_1 >_1 \omega_3 \\ \omega_3 >_2 \omega_2 >_2 \omega_1 \\ \omega_1 >_3 \omega_2 >_3 \omega_3 \end{array}$$

The Borda count of ω_2 is 4:

2 from the first place vote of voter 1.

1 each from the second place votes of voters 2 and 3.

What are the Borda counts of the other candidates?

Borda Inconsistency

- $a > b > c > d$
- $b > c > d > a$
- $c > d > a > b$
- $a > b > c > d$
- $b > c > d > a$
- $c > d > a > b$
- $a > b > c > d$
- $a = 11, b = 12, \mathbf{c=13}, \mathbf{d=6}$
- If the worst alternative -d- is removed
 - $a > b > c$
 - $b > c > a$
 - $c > a > b$
 - $a > b > c$
 - $b > c > a$
 - $c > a > b$
 - $a > b > c$
 - $\mathbf{a = 8, b = 7, c=6}$

Even if we keep the relative preferences between a, b and c, the final result changes completely

Problems of the Borda Procedure

- Most computationally expensive
- Eliminating (or adding) one irrelevant alternative may totally change the outcome of the voting
 - Winner => Last
 - Second worst => Winner
- Total order changes if options are removed one by one

Borda Procedure with Weak Orders

- The Borda protocol has been extended to manage weak orders in different ways
- A simple one: an option o receives from a voter v as many points as the number of options that are considered worst than o by v .

R_1	R_2	R_3
x_1	$x_2 \quad x_4$	x_3
$x_2 \quad x_3$	x_3	x_1
x_4	x_1	x_2
		x_4

$x_1 \rightarrow 3 + 0 + 2 = 5$ $x_2 \rightarrow 1 + 2 + 1 = 4$
 $x_3 \rightarrow 1 + 1 + 3 = 5$ $x_4 \rightarrow 0 + 2 + 0 = 2$

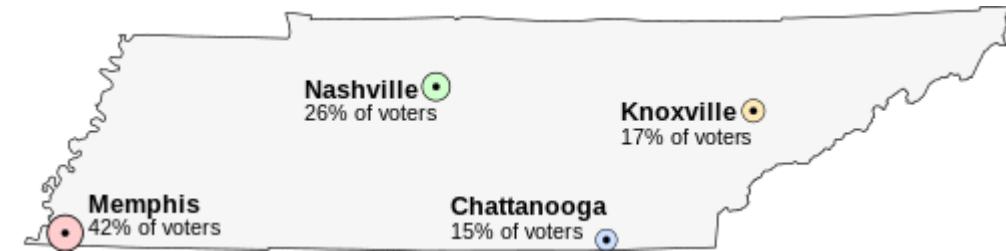
$x_1 \sim x_3 \succ x_2 \succ x_4$

Condorcet Procedure

- Each voter ranks the candidates in order of preference
- Each candidate is compared to each other
- If a candidate wins all the comparisons, it is the winner of the election
- In the event of a tie, use another resolution method (e.g. Borda count)

Condorcet Procedure

- Example: Voting on the location of Tennessee's capital
 - Election of the capital city of Tennessee
 - Everybody prefers to have the capital as close as possible
 - The candidates for the capital are:
 - Memphis, the state's largest city, with 42% of the voters, but located far from the other cities
 - Nashville, with 26% of the voters, near the center of the state
 - Knoxville, with 17% of the voters
 - Chattanooga, with 15% of the voters



42% of voters (close to Memphis)	26% of voters (close to Nashville)	15% of voters (close to Chattanooga)	17% of voters (close to Knoxville)
1. Memphis 2. Nashville 3. Chattanooga 4. Knoxville	1. Nashville 2. Chattanooga 3. Knoxville 4. Memphis	1. Chattanooga 2. Knoxville 3. Nashville 4. Memphis	1. Knoxville 2. Chattanooga 3. Nashville 4. Memphis

The preferences of the voters

Condorcet Procedure

- Example: Voting on the location of Tennessee's capital
 - Election of the capital city of Tennessee
 - Everybody prefers to have the capital as close as possible
 - The candidates for the capital are:
 - Memphis, the state's largest city, with 42% of the voters, but located far from the other cities
 - Nashville, with 26% of the voters, near the center of the state
 - Knoxville, with 17% of the voters
 - Chattanooga, with 15% of the voters

Pair	Winner
Memphis (42%) vs. Nashville (58%)	Nashville
Memphis (42%) vs. Chattanooga (58%)	Chattanooga
Memphis (42%) vs. Knoxville (58%)	Knoxville
Nashville (68%) vs. Chattanooga (32%)	Nashville
Nashville (68%) vs. Knoxville (32%)	Nashville
Chattanooga (83%) vs. Knoxville (17%)	Chattanooga

1st	Nashville [N]			3 Wins ↓
2nd	Chattanooga [C]		1 Loss → ↓ 2 Wins	[N] 68% [C] 32%
3rd	Knoxville [K]	2 Losses → ↓ 1 Win	[C] 83% [K] 17%	[N] 68% [K] 32%
4th	Memphis [M]	3 Losses →	[K] 58% [M] 42%	[C] 58% [M] 42% [N] 58% [M] 42%

Nashville wins

Problem of Condorcet method

- Possibility of circular ambiguities
 - No alternative wins to all the other alternatives
 - There are many ways to resolve them
 - Keep the candidate that wins more matches (Copeland)
 - Take into account the relative strengths of defeats (Minimax, Ranked Pairs, Schulze, ...)
 - You can look at the winning votes or at the winning margin

Use in practical exercise

- The collector agents could vote the order in which to pick up the discovered treasures.
- The vote of each collector could depend on its current position, the position of the treasures, its current state (idle, already moving towards a treasure, with a list of assigned treasures pending to be collected, etc.).
- The collector assigned to a treasure could then be somehow chosen between its voters.

Readings for this week

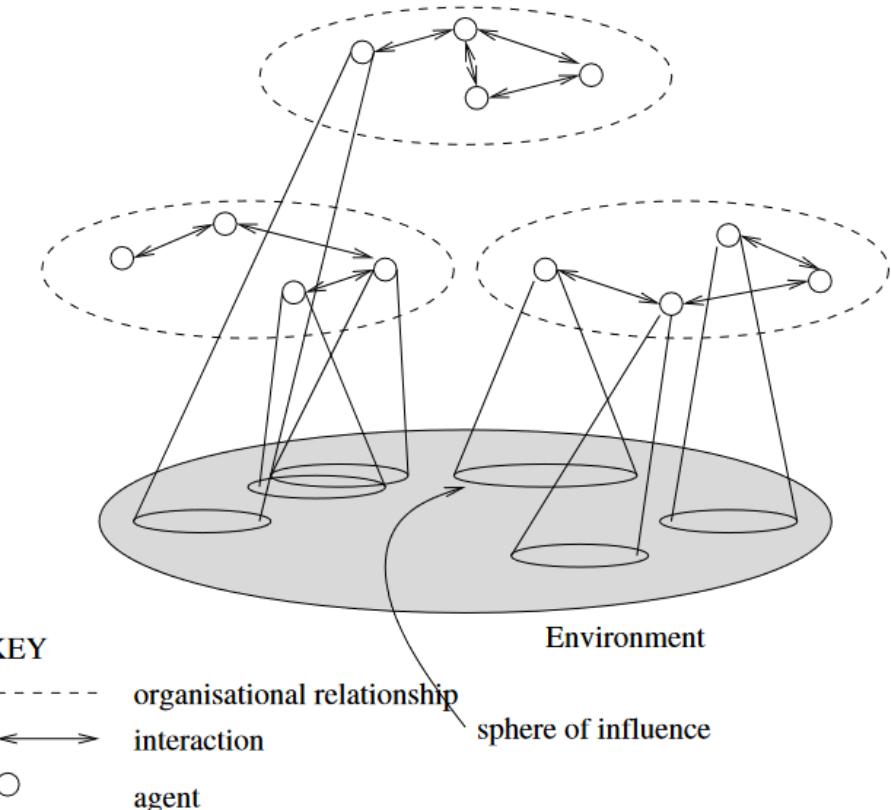
- Chapters 11, 12 of the book by M.Wooldridge “An introduction to Multi-Agent Systems” (2nd edition).

LECTURE 8: Multiagent Decision Making (II)

Introduction to Multi-Agent Systems (MESIIA, MIA)
URV

What are Multi-Agent Systems?

- A multiagent system contains a number of agents that:
 - interact through communication;
 - are able to act in an environment;
 - have different “spheres of influence” (which may coincide); and
 - will be linked by other (organisational) relationships.



Types of Agreement

- Multiagent encounters (game-like character)
- Voting.
- Coalition forming.
- Auctions



Cooperative Game Theory

- So far, we have taken a game theoretic view of multi-agent interactions.
 - Non-cooperative games
 - Competitive (Zero-Sum games)
 - **Non- Competitive**
 - Agents/players are self-interested agents.
 - No cooperation due to the following two reasons
 - Binding agreements are not possible (how can we recover the cooperation in this case?)
 - Utility is given to individuals based on individual action.
 - An agent plays strategies that maximize its **utility** and expect each other to play likewise.
- These constraints do not necessarily hold in the real world
 - Contracts, or collective payments can facilitate cooperation, leading to *Coalition Games* and *Cooperative Game Theory*

Example

Three kids, Adam, Bill, and Carmen want to buy ice cream. Adam has \$6, Bill has \$4, and Carmen has \$3. They found that the ice cream shop offers 3 different sizes of ice cream: small (250g) with cost of \$7, medium(375g) with cost of \$9 and large(500g) with cost of \$11.

They can share the ice cream among themselves. How should they cooperate to buy and share the ice cream?

Cooperative Games

- Coalitional games model scenarios where agents can benefit by cooperating.
- Three stages of cooperative action:

Coalitional Structure Generation

Deciding in principle who will work together. It asks the basic question:

Which coalition should I join?

The result: partitions agents into disjoint coalitions. The overall partition is a coalition structure.

Solving the optimization problem of each coalition

Deciding how to work together, and how to solve the “joint problem” of a coalition. It also involves finding how to maximise the utility of the coalition itself, and typically involves joint planning etc.

Dividing the benefits

Deciding “who gets what” in the payoff. Coalition members cannot ignore each other’s preferences,

...if you try to give me a bad payoff, I can always walk away...

We might want to consider issues such as fairness of the distribution.

Formalising Cooperative Games

- Let $Ag = \{1, \dots, n\}$ to be the set of agents/players
- $C, C^*, C_1 \subseteq Ag$ are called coalitions
- $C = Ag$ is the ***grand coalition***
- $\{i\}$ where $i \in Ag$ is a ***singleton coalition***
- Let $v: 2^{Ag} \rightarrow \mathbb{R}$ be the characteristic function of the game. It assigns to every possible coalition a numeric value representing the pay-off that may be distributed between the members of the coalition.
- Then, we can formally define a ***cooperative game*** (or coalitional game) as a pair $G = \langle Ag, v \rangle$.

A possible function of the Ice-Cream example

Coalition (C)	$v(C)$
\emptyset	0
$\{A\}$	0
$\{B\}$	0
$\{C\}$	0
$\{A, B\}$	375
$\{A, C\}$	375
$\{B, C\}$	250
$\{A, B, C\}$	500

Outcomes

- An outcome of a coalitional game $G = \langle Ag, v \rangle$ is a pair $\langle \mathbb{C}, X \rangle$ where:
 - (1) $\mathbb{C} = \{C_1, C_2, \dots, C_k\}$ is the **coalition structure**, i.e., partition of Ag , s.t.
 - a. $\bigcup_{i=1}^k C_i = Ag$ and
 - b. $C_i \cap C_j = \emptyset, \forall i \neq j$
 - (2) $X = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ is a payoff vector indicating the value of each agent/player, s.t.
 - a. $x_i \geq v(\{i\}) \forall i \in Ag$ **[Individual Rationality]**
 - b. $\sum_{i \in C} x_i = v(C) \forall C \in \mathbb{C}$ **[Efficiency]**
- Collation and coalition structure
 - If we have 3 agents A, B, C, how many possible coalitions and coalition structures? List them.
 - **Outcome examples (Ice-Cream)?**

Outcomes

- An outcome of a coalitional game $G = \langle Ag, v \rangle$ is a pair $\langle \mathbb{C}, X \rangle$ where:
 - (1) $\mathbb{C} = \{C_1, C_2, \dots, C_k\}$ is the **coalition structure**, i.e., partition of Ag , s.t.
 - a. $\bigcup_{i=1}^k C_i = Ag$ and
 - b. $C_i \cap C_j = \emptyset, \forall i \neq j$
 - (2) $X = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ is a payoff vector indicating the value of each agent/player, s.t.
 - a. $x_i \geq v(\{i\}) \forall i \in Ag$ **[Individual Rationality]**
 - b. $\sum_{i \in C} x_i = v(C) \forall C \in \mathbb{C}$ **[Efficiency]**
- **Outcome examples (Ice-Cream)**
 - $((\{A, B\}, \{C\}), (200, 175, 0))$, An outcome
 - $((\{A, B, C\}), (250, 150, 100))$, An outcome
 - $((\{A, B\}, \{C\}), (250, 150, 100))$ not an outcome, why?

Super-Additive Games

A coalitional game $G = \langle Ag, \nu \rangle$ is super-additive if

- $\nu(C \cup C^\circ) \geq \nu(C) + \nu(C^\circ)$ for all $C, C^\circ \subseteq Ag$ s.t. $C \cap C^\circ = \emptyset$

- In that case, the coalition that maximises social welfare is the *Grand Coalition*
- So, we can simply reduce the definition of outcome to be only the payoff vector $X = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$
- Is the Ice-Cream example super-additive?

Sub-Additive Games

A coalitional game $G = \langle Ag, \nu \rangle$ is Sub-additive if

- $\nu(C \cup C^\circ) < \nu(C) + \nu(C^\circ)$ for all $C, C^\circ \subseteq Ag$ s.t. $C \cap C^\circ = \emptyset$

- In that case, The coalitions that maximis social welfare are *singletons*
- Some games are neither subadditive or superadditive:
 - the characteristic function value calculations need to be determined for each of the possible coalitions!
 - This is exponentially complex.

Which Coalition Should I Join?

- The agent should only join a coalition C which is:
 - *Feasible*: the coalition C really could obtain some payoff than an agent could not object to; and
 - *Efficient*: all of the payoff is allocated
- However, there may be many coalitions
 - Each has a different characteristic function
 - Agents prefer coalitions that are as productive as possible
 - Therefore a coalition will only form if all the members prefer to be in it (I.e. they don't defect to a more preferable coalition)
- Therefore:
 - “*which coalition should I join?*” can be reduced to “*is the coalition stable?*”
 - Is it rational for all members of coalition C to stay with C , or could they benefit by defecting from it?
 - There's no point in me joining a coalition with you, unless you want to form one with me, and vice versa.

Stability

- Let's consider the ice cream example
 - Players and resources : Adam has \$6, Bill has \$4, and Carmen has \$3
 - Outcomes (ice cream sizes): Small (250), Medium (375), and Large (500)
- As this game is super additive, then the best coalition is the *ground coalition* $\{A, B, C\}$, so:
 - The resources will be in total \$13, and
 - They can buy the large ice cream (500)g
- Which of the following outcomes is stable?
 - $(500/3, 500/3, 500/3)$, $(187.5, 187.5, 125)$, $(250, 125, 125)$, $(250, 244, 1)$
 - To answer this, we need to define what we mean by *stable outcome*.

Stability

- Let's first introduce the objection term
 - We say that a coalition C **objects** to an outcome for the ground coalition if there is some outcome for C that makes all members of C strictly better off.
 - Formally, $C \subseteq Ag$ objects to an outcome (x_1, x_2, \dots, x_n) for the grand coalition if there is some outcome $(x'_1, x'_2, \dots, x'_n)$ for C s.t.:
$$x_i > x'_i, i \in C$$
- Hence, we say that an outcome is **stable** if there is no objection, or no agent has incentive to **defect**.
- Stability is a **necessary** but not **sufficient** condition for coalitions to form
 - i.e. Unstable coalitions will never form, but a stable coalition isn't guaranteed to form

Stability

- Which of the following outcomes is stable?
 - $(500/3, 500/3, 500/3), (187.5, 187.5, 125), (250, 125, 125), (250, 244, 1)$
 - Remember the characteristic function values of the game:

Coalition (C)	$\nu(C)$
ϕ	0
$\{A\}$	0
$\{B\}$	0
$\{C\}$	0
$\{A, B\}$	375
$\{A, C\}$	375
$\{B, C\}$	250
$\{A, B, C\}$	500

The Core

- Stability can be reduced to the notion of the *core*
- The core of a coalitional game is the set of feasible distributions of payoff to members of a coalition that no sub-coalition can reasonably *object* to.
- The idea is that an outcome is not going to happen if somebody objects to it!
 - i.e. if the core is *empty*, then no coalition can form
- So, the question “Is the grand coalition stable?” is same as asking: “Is the core non-empty?”

The Core and Fair Payoffs

- Sometimes the core is non-empty but is it “*fair*”?
 - Suppose we have $Ag = \{1, 2\}$, with the following characteristic function, v :
 - $v(\{1\}) = 5$
 - $v(\{2\}) = 5$
 - $v(\{1, 2\}) = 20$
 - The outcome $\langle 20, 0 \rangle$ (i.e., agent 1 gets everything) *will not be in the core*, since agent 2 can object; by *working on its own it can do better*, because $v(\{2\}) = 5$
 - However, outcome $\langle 14, 6 \rangle$ is in the core, as *agent 2 gets more than working on its own*, and thus has no objection.
- But is it “*fair*” on agent 2 to get only a payoff of 6, if agent 1 gets 14???

Marginal Contribution

- Let $\delta_i(C)$ to be the amount that agent i adds by joining a coalition $C \subseteq Ag$
 - i.e. the *marginal contribution* of i to C is defined as $\mu_i(C) = v(C \cup \{i\}) - v(C)$
 - Note that if $\mu_i(C) = v\{i\}$ then there is *no added value* from i joining C as the amount i adds is the same as if i earns on its own

Sharing the Benefits of Cooperation

- The Shapley value is best known attempt to define how to divide benefits of cooperation fairly.
 - It does this by taking into account how much an agent contributes.
 - The Shapley value of agent i is the average amount that i is expected to contribute to a coalition.
 - The Shapley value is one that satisfies the axioms opposite!
- The Shapley Value for i , denoted sh_i , is the value that agent $i \in Ag$ is given in the game $\langle Ag, v \rangle$

Symmetry

Agents that make the same contribution should get the same payoff. I.E. the amount an agent gets should only depend on their contribution.

Dummy Player

These are agents that never have any synergy with any coalition, and thus only get what they can earn on their own.

Additivity

If two games are combined, the value an agent gets should be the sum of the values it gets in the individual games.

Shapley Axioms: Symmetry

- Agents that make the same contribution should get the same payoff
 - The amount an agent gets should only depend on their contribution
 - Agents i and j are interchangeable if $\mu_i(C) = \mu_j(C) \forall C \subseteq Ag - \{i, j\}$
- The symmetry axiom states:
 - If i and j are interchangeable, then $sh_i = sh_j$



Shapley Axioms: Dummy Player

- Agents that never have any synergy with any coalition, and thus only get what they can earn on their own.
- The amount an agent gets should only depend on their contribution
 - An agent is a dummy player if:
$$\mu_i(C) = v(\{i\}) \quad \forall C = Ag - \{i\}$$
 - i.e. an agent only adds to a coalition what it could get on its own
- The dummy player axiom states:
 - If i is a dummy player, then $sh_i = v(\{i\})$



Shapley Axioms: Additivity

- If two games are combined, the value an agent gets should be the sum of the values it gets in the individual games.
 - I.e. an agent doesn't gain or loose by playing more than once
 - Let $G^1 = \langle Ag, v^1 \rangle, G^2 = \langle Ag, v^2 \rangle$ be games with the same agents
 - Let $i \in Ag$ be one of the agents
 - Let sh_i^1, sh_i^2 be the values agent i gets in games G^1 and G^2 respectively
 - Let $G^{1+2} = \langle Ag, v^{1+2} \rangle$ be the game such that $v^{1+2}(C) = v^1(C) + v^2(C)$
- The additivity axiom states:
 - The value sh_i^{1+2} of agent i in game G^{1+2} should be $sh_i^1 + sh_i^2$



Shapley value

- Let $\Pi(Ag)$ denote the set of all possible orderings of the agents Ag .
- If $o \in \Pi(Ag)$, then let $C_i(o)$ be the agents that appear before i in o , so, we define the Shapley value for agent i , denoted sh_i , as:

$$sh_i = \frac{1}{|Ag|!} \sum_{o \in \Pi(Ag)} \mu_i(C_i(o))$$

- For example, if $Ag = \{1, 2, 3\}$ then:
 - $\Pi(Ag) = \{(1,2,3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$
 - For $o = (3, 1, 2)$, find
 - $C_3(o) = ?$
 - $C_1(o) = ?$
 - $C_2(o) = ?$

Shapley Example

- Suppose we have $Ag = \{1, 2\}$, with the opposite characteristic function.
- We first calculate the marginal contribution $\mu_i(C)$ of each agent $i \in C$, for each coalition $C \subseteq Ag$

Coalition (C)	Agent i	$\mu_i(C)$
ϕ	1	$v(\phi \cup \{1\}) - v(\phi) = 5 - 0 = 5$
	2	$v(\phi \cup \{2\}) - v(\phi) = 10 - 0 = 10$
$\{1\}$	2	$v(\{1\} \cup \{2\}) - v(\{1\}) = 20 - 5 = 15$
$\{2\}$	1	$v(\{1\} \cup \{2\}) - v(\{2\}) = 20 - 10 = 10$

Coalition (C)	$v(C)$
ϕ	0
$\{1\}$	5
$\{2\}$	10
$\{1, 2\}$	20

- Then, we can calculate the individual Shapley values for each agent i :

$$\bullet \quad sh_1 = \frac{\mu_1(\phi) + \mu_1(\{2\})}{|Ag|!} = \frac{5+10}{2} = 7.5$$

$$\bullet \quad sh_2 = \frac{\mu_2(\phi) + \mu_2(\{1\})}{|Ag|!} = \frac{10+15}{2} = 12.5$$

Representing Coalitional Games

- It is important for an agent to know if the core of a coalition is non-empty
 - Problem: a naive, obvious representation of a coalitional game is *exponential* in the size of Ag .
 - For n agents we need an input file of $2^n + 1$ lines.
 - e.g. a 100-player game would require 1.2×10^{30} lines
- Now such a representation is:
 - utterly infeasible in practice; and
 - so large that it renders comparisons to this input size meaningless

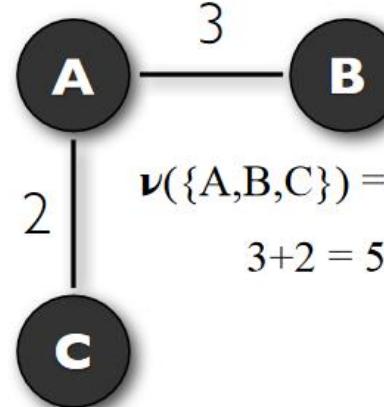
```
% Representation of a Simple  
% Characteristic Function Game  
  
% List of Agents  
1,2,3  
% Characteristic Function  
1 = 5  
2 = 5  
3 = 5  
1,2 = 10  
1,3 = 10  
2,3 = 10  
1,2,3 = 25
```

Representing Characteristic Functions?

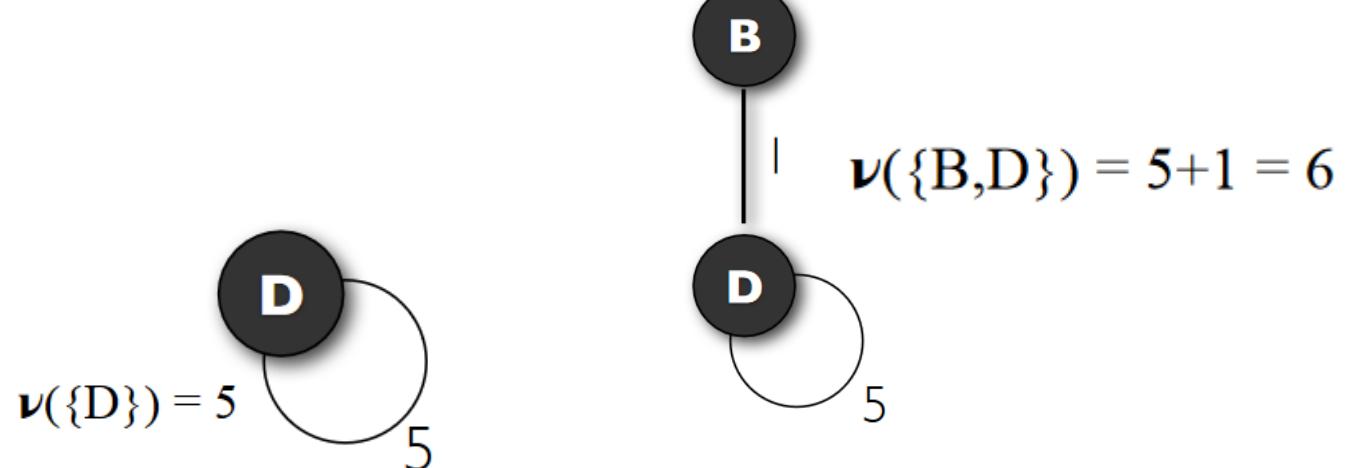
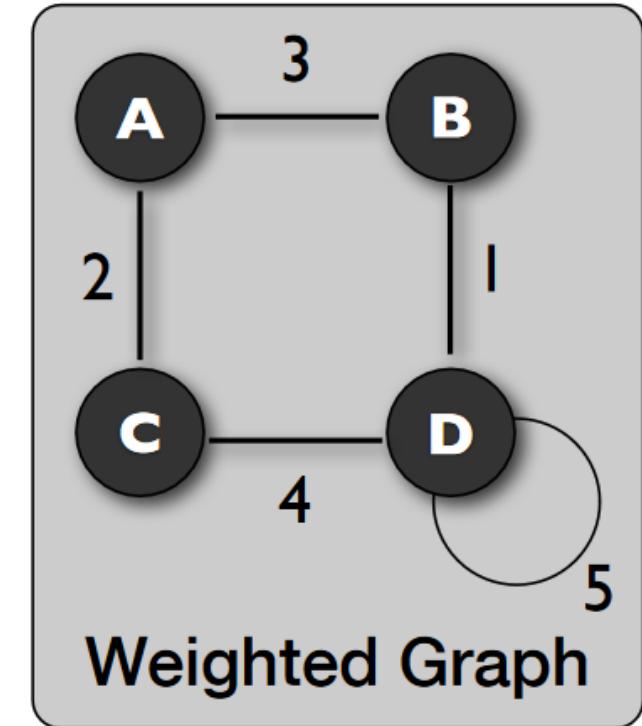
- Two approaches to this problem:
 - try to find a complete representation that is succinct in “most” cases
 - try to find a representation that is not complete but is always succinct
- A common approach:
 - interpret characteristic function over a combinatorial structure.
- We look at two possible approaches:
 - Induced Subgraph
 - Marginal Contribution Networks

Induced Subgraph

- Represent v as an undirected graph on Ag , with weights $w_{i,j}$ between $i, j \in Ag$.
- Value of coalition C is then:
 - $v(C) = \sum_{i,j \in Ag} w_{i,j}$
- .e., the value of a coalition $C \subseteq Ag$ is *the weight of the subgraph* induced by C



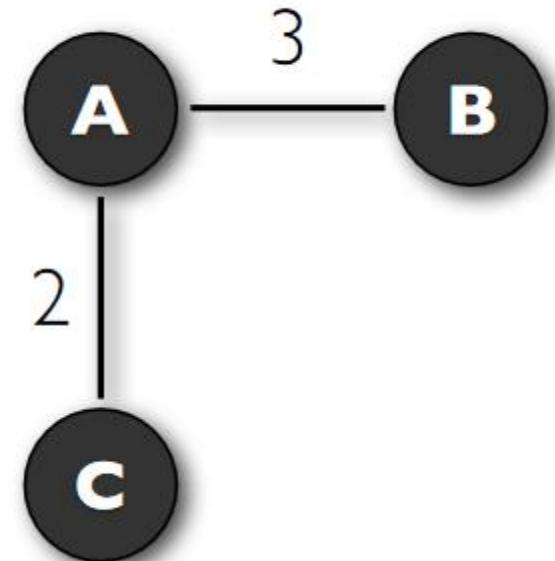
$$v(\{A,B,C\}) = 3+2=5$$



$$v(\{D\}) = 5$$

Induced Subgraph

- Representation is succinct, but not complete
 - there are characteristic functions that cannot be captured using this representation
- Determining emptiness of the core is NP-complete
 - Checking whether a specific distribution is in the core is co-NP-complete
- Shapley value can be calculated in polynomial time
 - $sh_i = \frac{1}{2} \sum_{i \neq j} w_{i,j}$
 - i.e. an agent gets *half the income from the edges in the graph to which it is attached.*



Marginal Contribution Nets

- Characteristic function ν represented as rules:

pattern \rightarrow value

- Pattern is conjunction of agents, a rule applies to a group of agents C if C is a superset of the agents in the pattern.
- Value of a coalition is then sum over the values of all the rules that apply to the coalition.
- We can also allow negations in rules (i.e. for when an agent is not present).

Rule set (rs) 2:

$$a \wedge b \rightarrow 5$$

$$b \rightarrow 2$$

$$c \rightarrow 4$$

$$b \wedge \neg c \rightarrow -2$$

$\nu_{rs2}(\{a\}) = 0$	no rules apply
$\nu_{rs2}(\{b\}) = 2 + -2 = 0$	2 nd and 4 th rules
$\nu_{rs2}(\{c\}) = 4$	3 rd rule
$\nu_{rs2}(\{a, b\}) = 5 + 2 + -2 = 5$	1 st , 2 nd and 4 th rules
$\nu_{rs2}(\{a, c\}) = 4$	3 rd rule
$\nu_{rs2}(\{b, c\}) = 2 + 4 = 6$	2 nd and 3 rd rules
$\nu_{rs2}(\{a, b, c\}) = 5 + 2 + 4 = 11$	1 st , 2 nd and 3 rd rules

Marginal Contribution Nets

- Calculating the Shapley value for marginal contribution nets is similar to that for induced subgraphs
 - Again, Shapley's symmetry axiom applies to each agent
 - The contributions from agents in the same rule is equal
 - The additivity property means that:
 - we calculate the Shapley value for each rule
 - sum over the rules to calculate the Shapley value for each agent
 - Handling negative values requires a different method

Calculating the Shapley Value

$$sh_i = \sum_{r \in rs; i \text{ occurs in lhs of } r} sh_i^r$$

where:

$$sh_i^{1 \wedge \dots \wedge l \rightarrow x} = \frac{x}{l}$$

Example:

$$\begin{aligned} a \wedge b &\rightarrow 5 \\ b &\rightarrow 2 \\ c &\rightarrow 4 \end{aligned}$$

$$sh_a = \frac{5}{2} = 2.5$$

$$sh_b = \frac{5}{2} + \frac{2}{1} = 4.5$$

What is sh_c ?

Coalition Structure Generation

- A *coalition structure* is a partition of the overall set of agents Ag into mutually disjoint coalitions.
- Example: if $Ag = \{1,2,3\}$; then there are seven possible coalitions:
 - $\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}$
- And five possible coalition structures:
 - $\{\{1\}, \{2\}, \{3\}\}, \{\{1\}, \{2,3\}\}, \{\{2\}, \{1, 3\}\}, \{\{3\}, \{1,2\}\}, \{\{1,2,3\}\}$

Given a coalitional game $G = \langle Ag, v \rangle$, we say that the *socially optimal coalition structure*

$$\mathbb{C}^* = \arg \max_{\mathbb{C} \in \text{partitions of } Ag} V(\mathbb{C})$$

where:

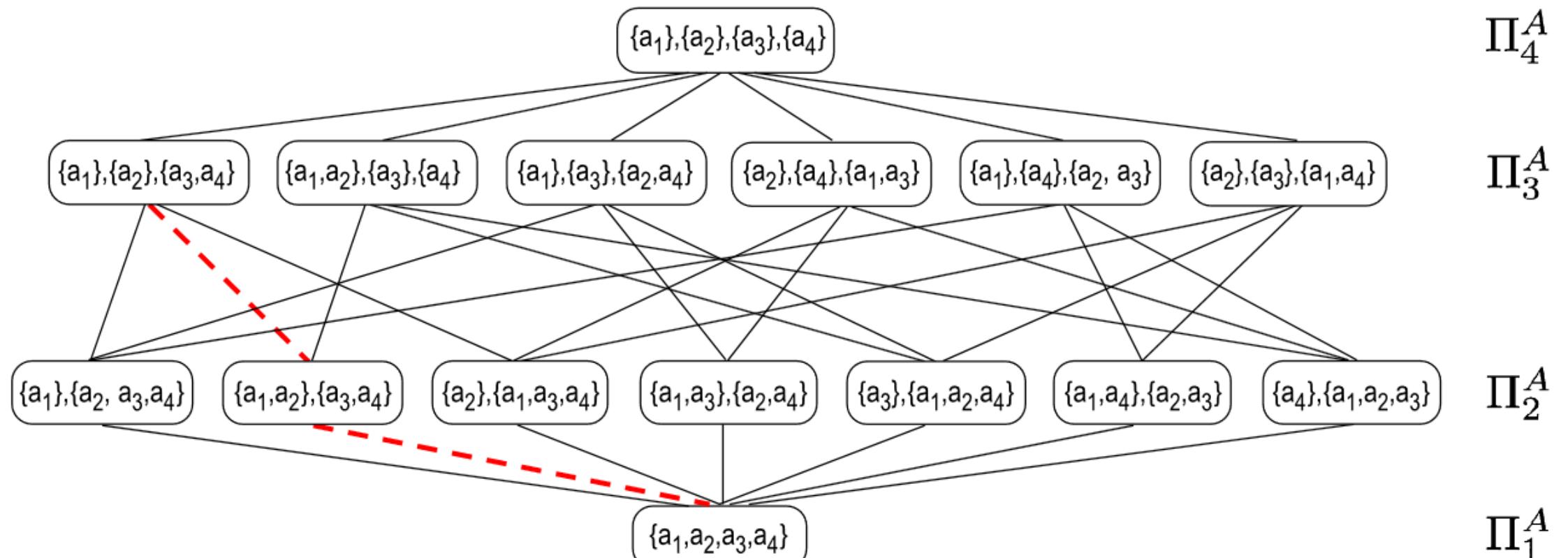
$$V(\mathbb{C}) = \sum_{C \in \mathbb{C}} v(C)$$

- This problem is computationally hard.
- Moreover, it is NP-hard to find an optimal coalition structure given oracle access to the characteristic function

Coalition Structure Generation

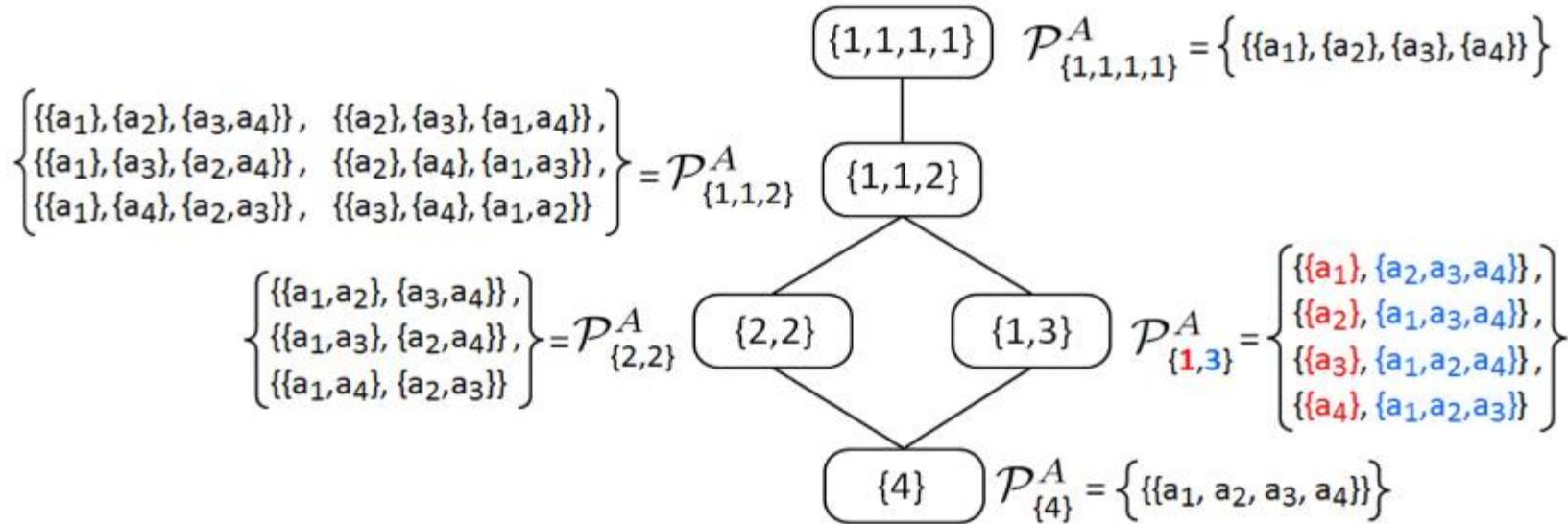
- Space Representation
 - There are two main representations of the space of possible coalition structures:
 - **Coalition structure graph** (Sandholm, 1999)
 - Every node represents a coalition structure.
 - Categorized into levels P_1, P_2, \dots, P_n , where level P_i contains the nodes that represent all coalition structures containing exactly i coalitions.
 - An edge connects two coalition structures if and only if:
 1. they belong to two consecutive levels P_i and P_{i-1} , and
 2. the coalition structure in P_{i-1} can be obtained from the one in P_i by merging two coalitions into one.
 - **Integer partition-based representation** (Rahwan et al. 2007)
 - Categorize them based on the sizes of the coalitions they contain.
 - Divide the space of coalition structures into disjoint subspaces that are each represented by an integer partition of n .

Coalition Structure Generation



The coalition structure graph of 4 agents.

Coalition Structure Generation



The integer partition-based representation for 4 agents.

Coalition Structure Generation

- **Methods:**

- An Anytime Algorithm for Optimal Coalition Structure Generation (Rahwan et al., 2009) – a.k.a **IP** algorithm
- Distributing Coalition Value Calculations to Coalition Members (Riley et al., 2015) - a.k.a (n,s)-sequences

IP Method

- Let's start by an example

P ₁	value	P ₂	value	P ₃	value	P ₄	value
{1}	30	{1, 2}	50	{1, 2, 3}	90	{1, 2, 3, 4}	140
{2}	40	{1, 3}	60	{1, 2, 4}	120		
{3}	25	{1, 4}	80	{1, 3, 4}	100		
{4}	45	{2, 3}	55	{2, 3, 4}	115		
		{2, 4}	70				
		{3, 4}	80				

- What is the optimal coalition structure?
- Answer:

IP Method

- Let's start by an example

P ₁	value	P ₂	value	P ₃	value	P ₄	value
{1}	30	{1, 2}	50	{1, 2, 3}	90	{1, 2, 3, 4}	140
{2}	40	{1, 3}	60	{1, 2, 4}	120		
{3}	25	{1, 4}	80	{1, 3, 4}	100		
{4}	45	{2, 3}	55	{2, 3, 4}	115		
		{2, 4}	70				
		{3, 4}	80				

- What is the optimal coalition structure?
- Answer: $\{\{1\}, \{2\}, \{3, 4\}\}$

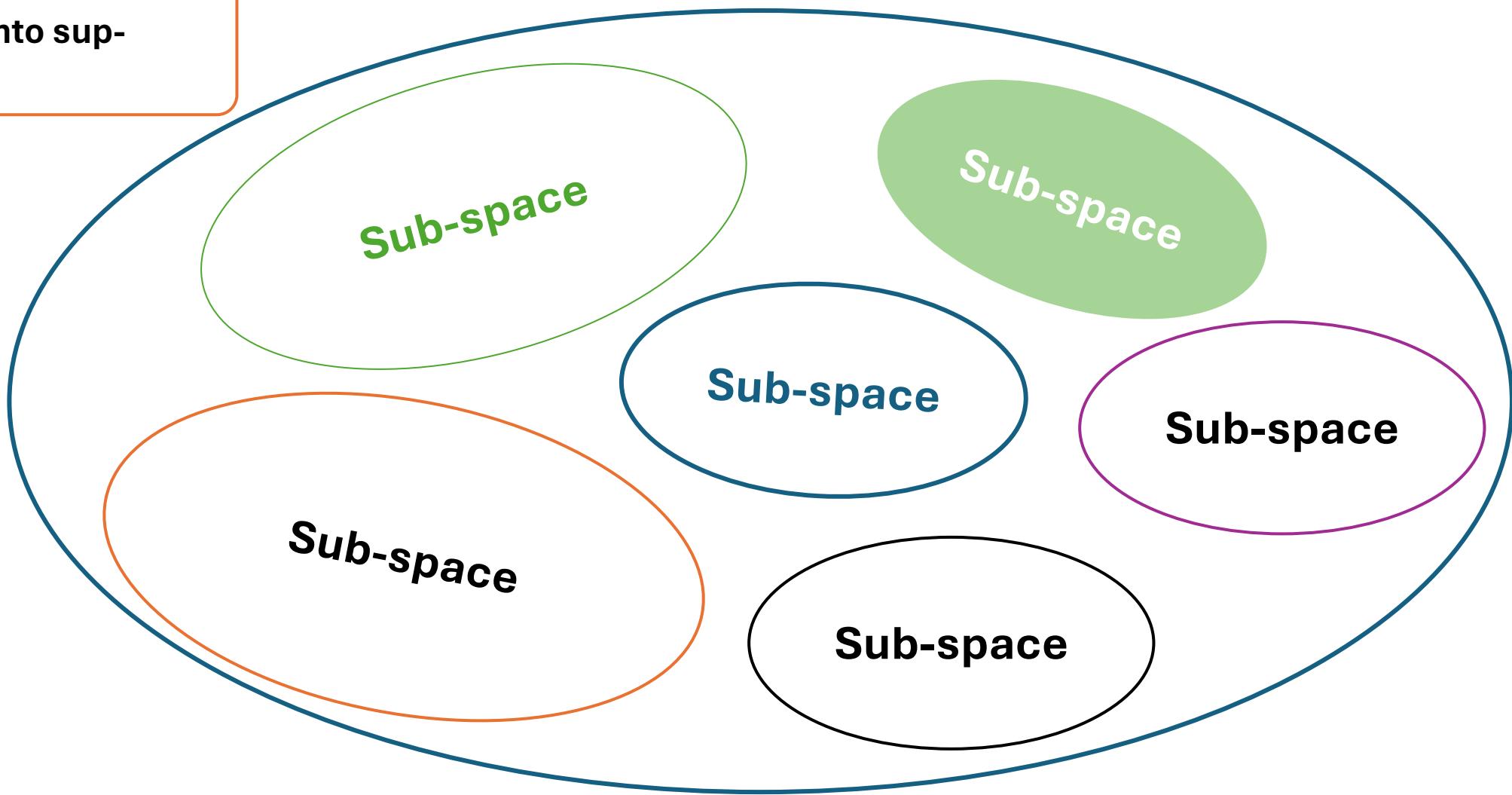
IP Method: Basic Idea



Search Space

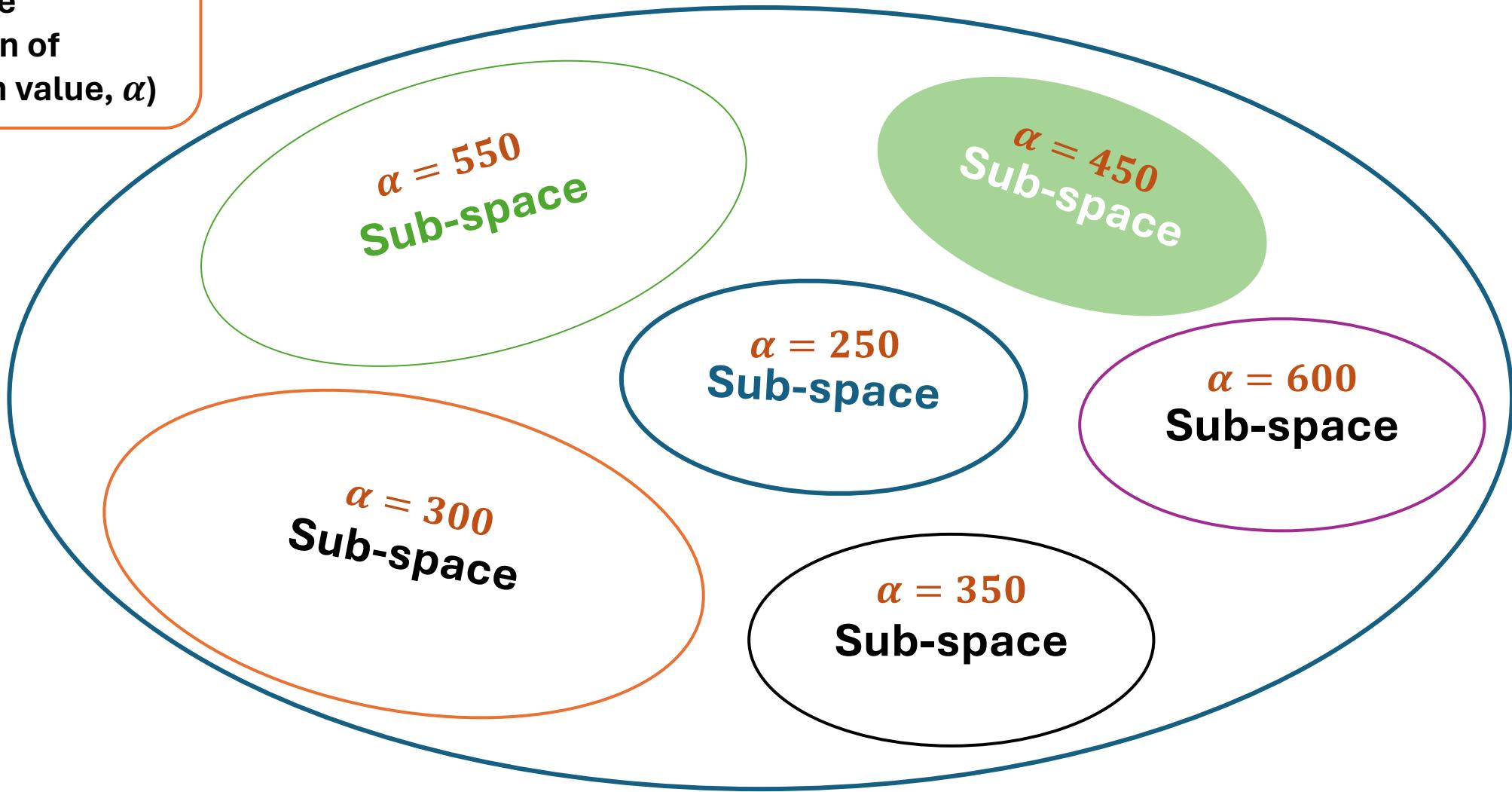
IP Method: Basic Idea

Divide the search space into sub-spaces



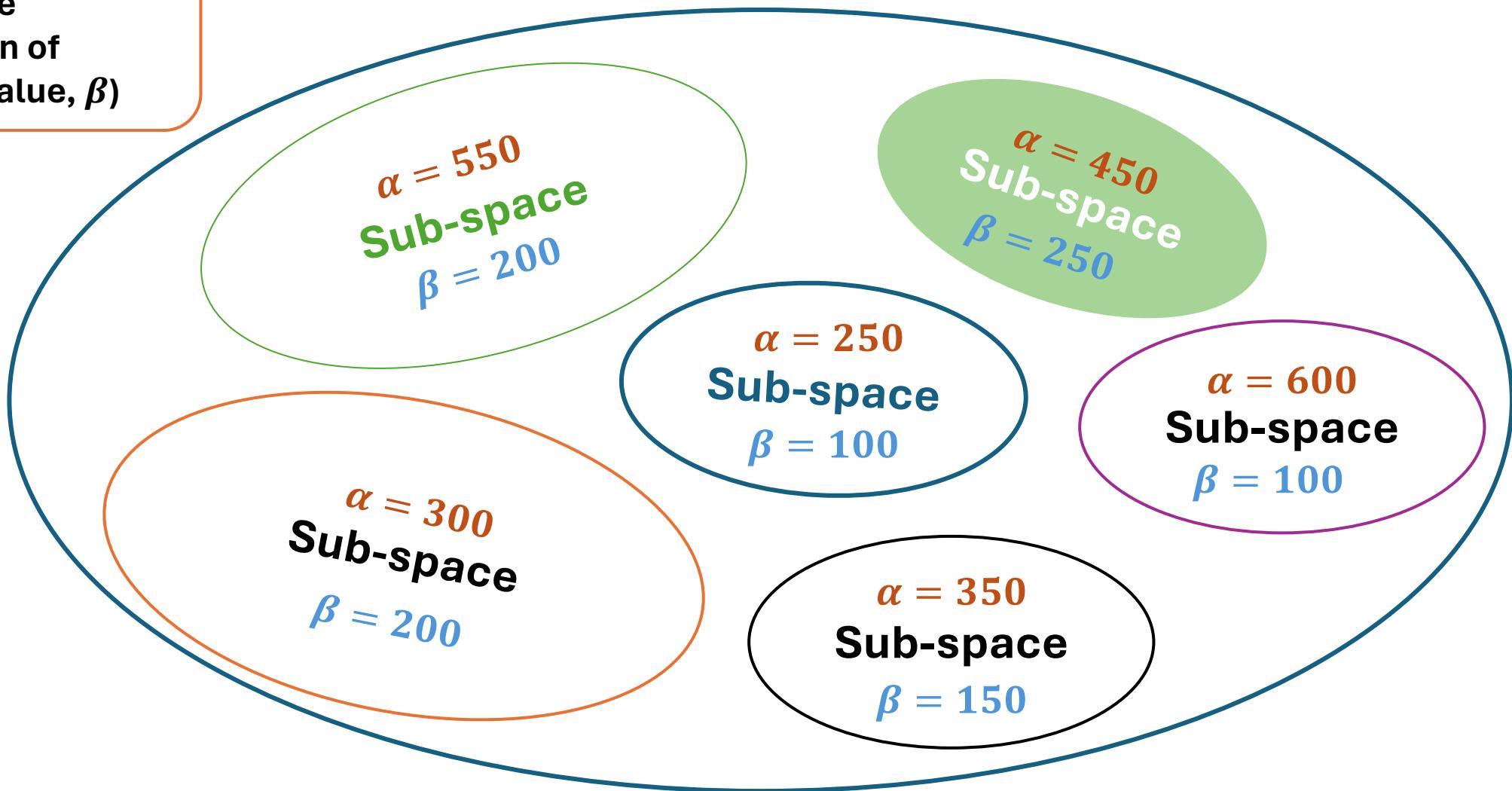
1) Calculate the upper-bound of each sub-space (that is the estimation of maximum value, α)

IP Method: Basic Idea

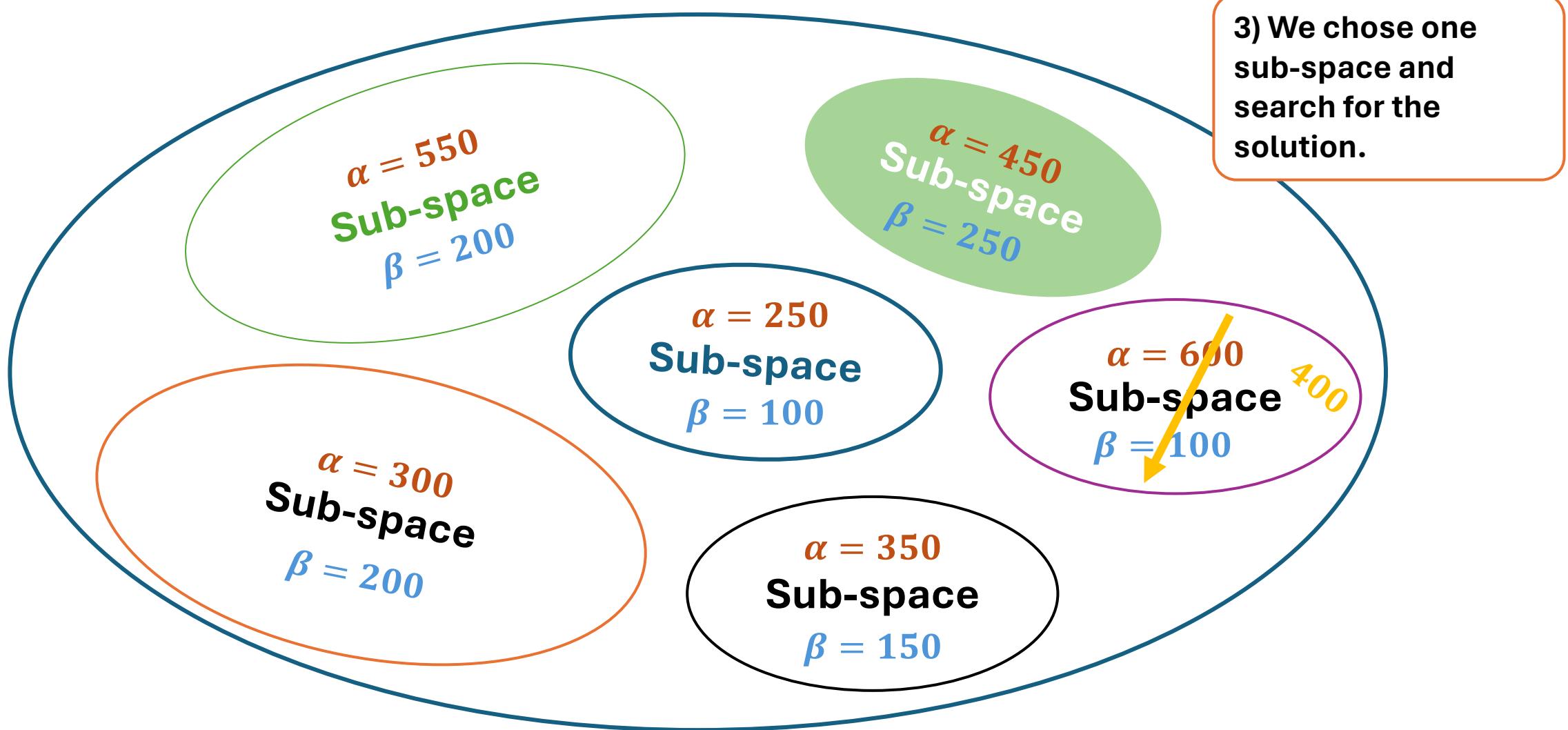


2) Calculate the lower-bound of each sub-space
(that is the estimation of average value, β)

IP Method: Basic Idea



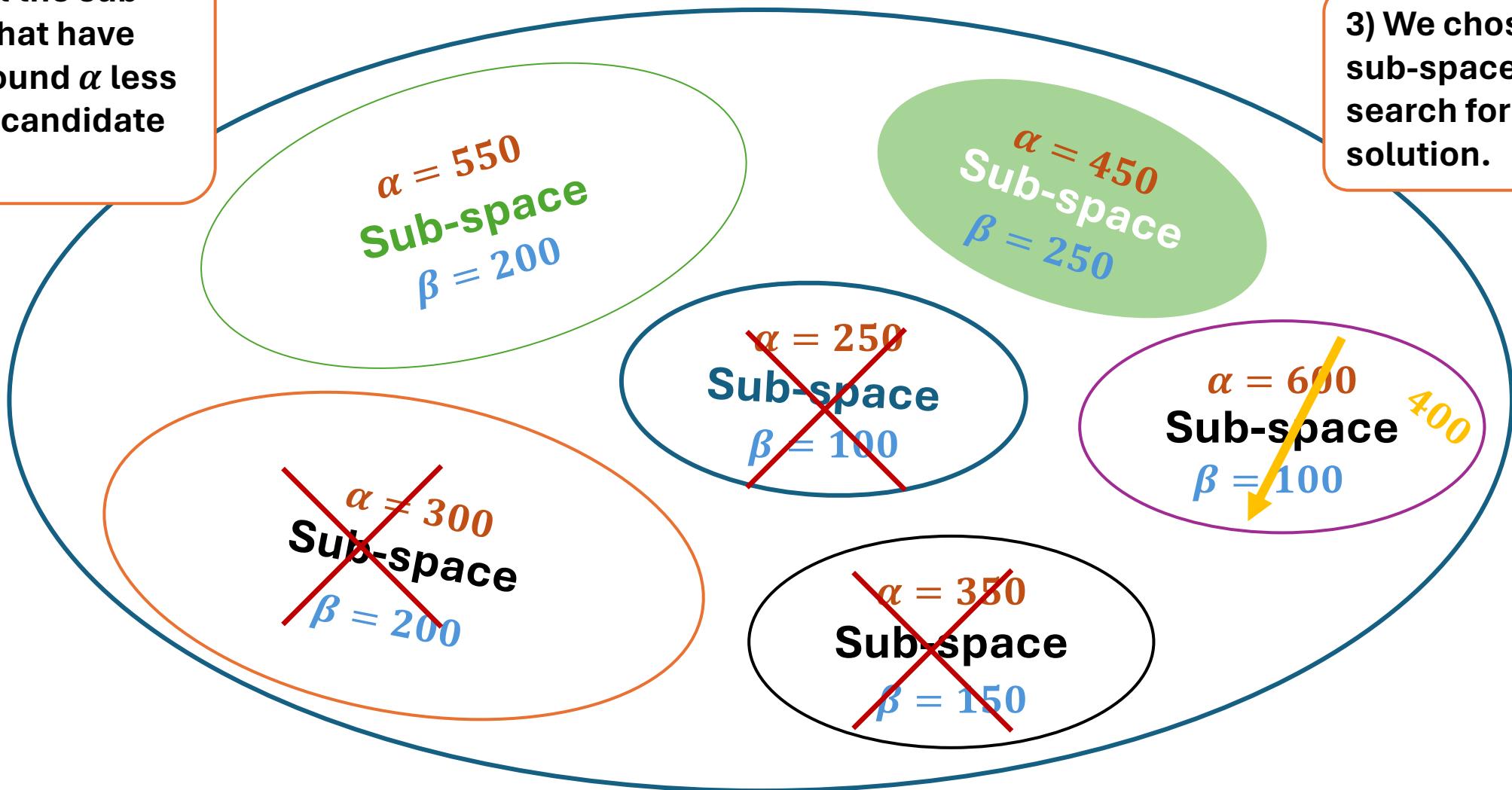
IP Method: Basic Idea



IP Method: Basic Idea

4) Burning: we ignore all the sub-spaces that have upper-bound α less than the candidate solution

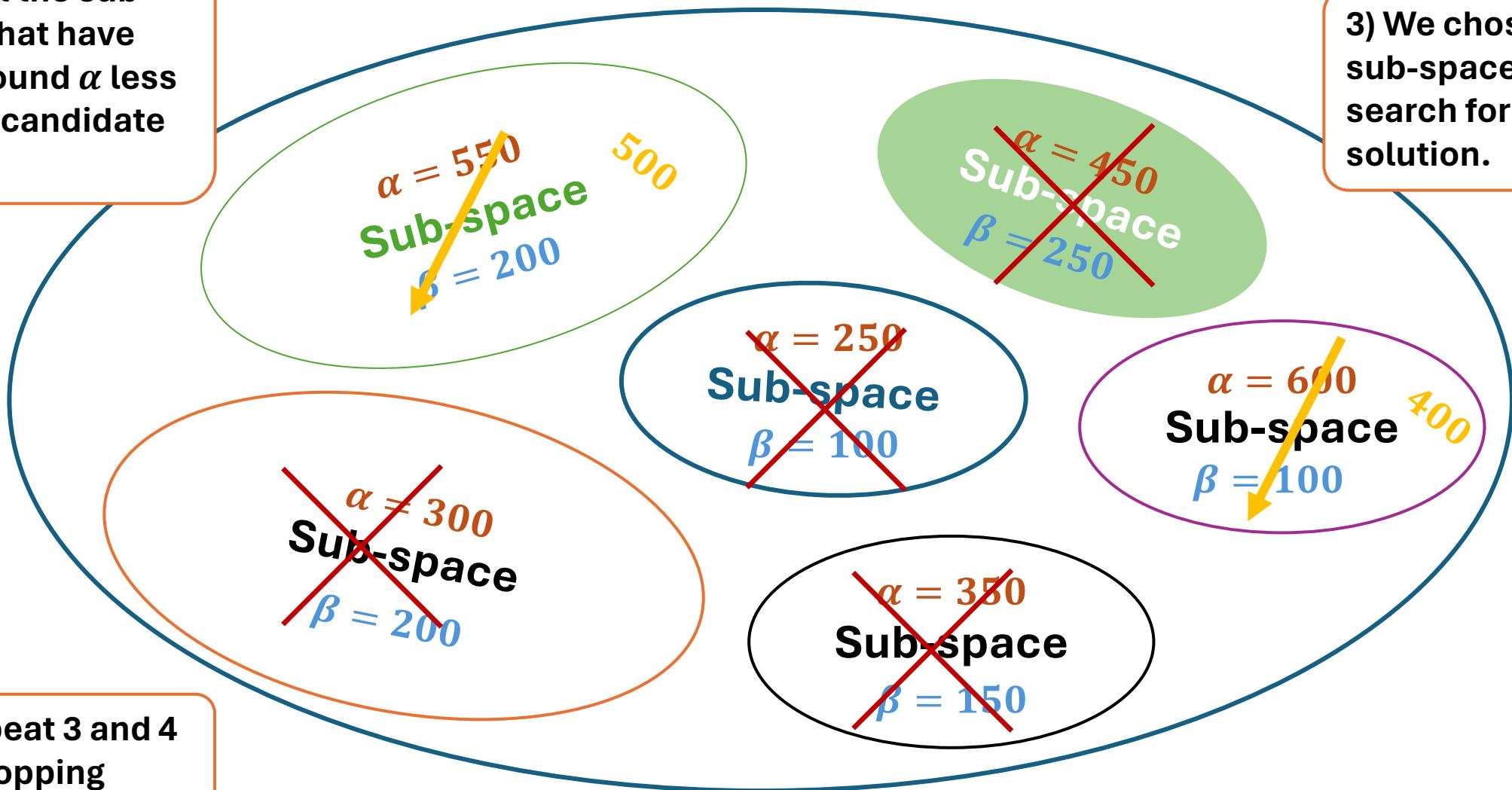
3) We chose one sub-space and search for the solution.



IP Method: Basic Idea

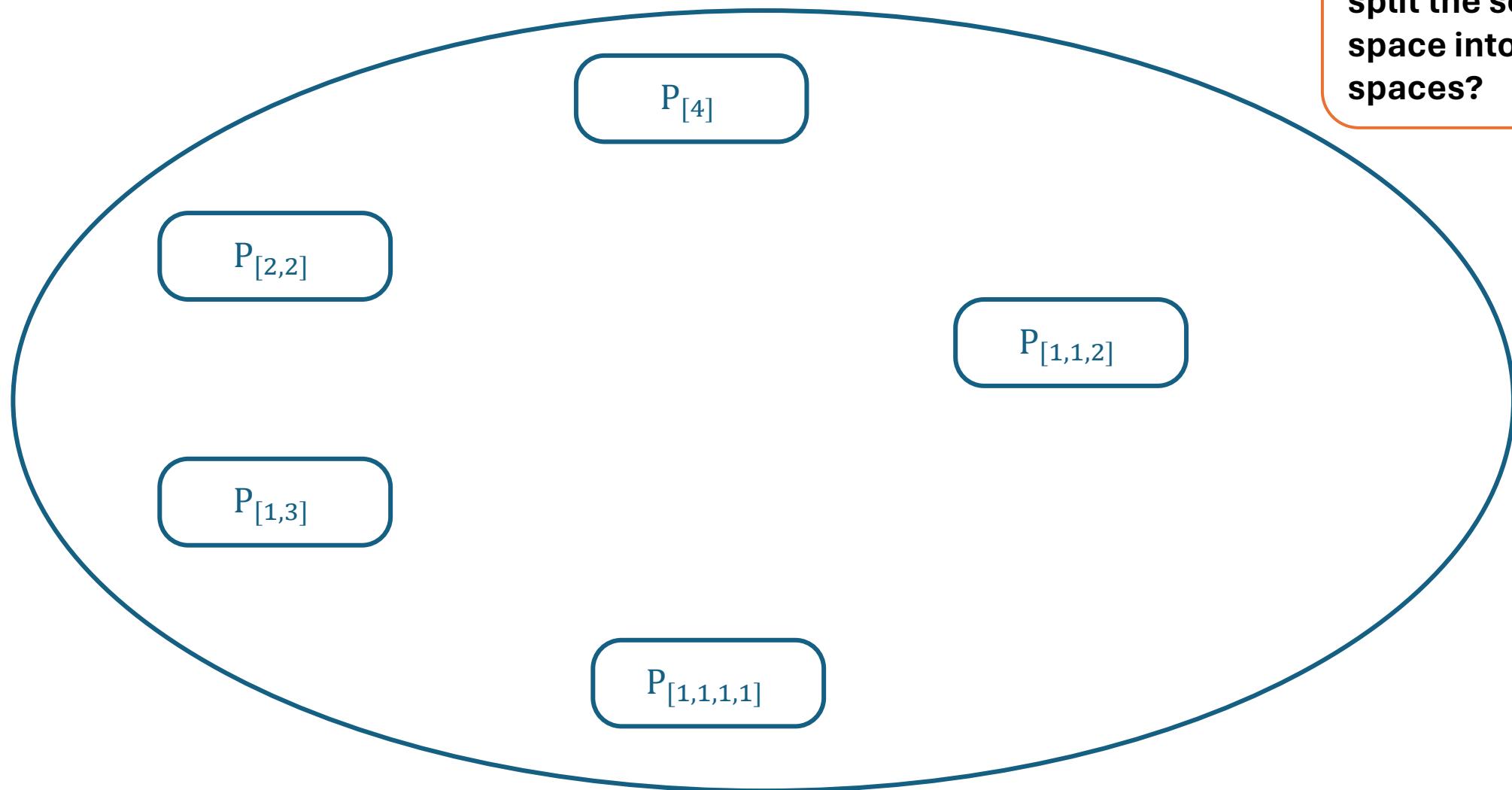
4) Burning: we ignore all the sub-spaces that have upper-bound α less than the candidate solution

3) We chose one sub-space and search for the solution.



5) We repeat 3 and 4 until a stopping criteria hold

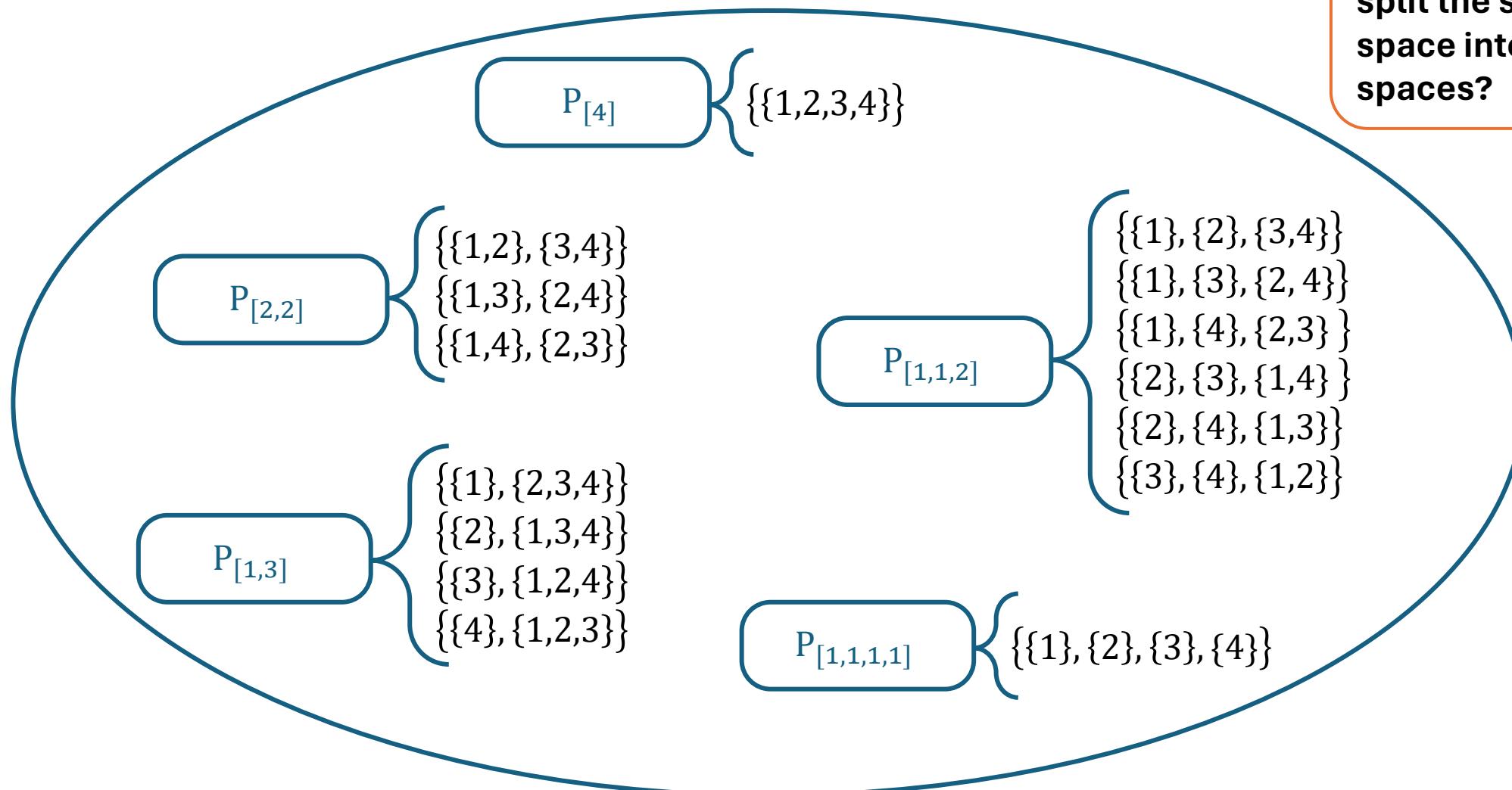
IP Method: Example



Assume we have 4 agents $Ag = \{1, 2, 3, 4\}$, how we split the search space into sub-spaces?

IP Method: Example (1)

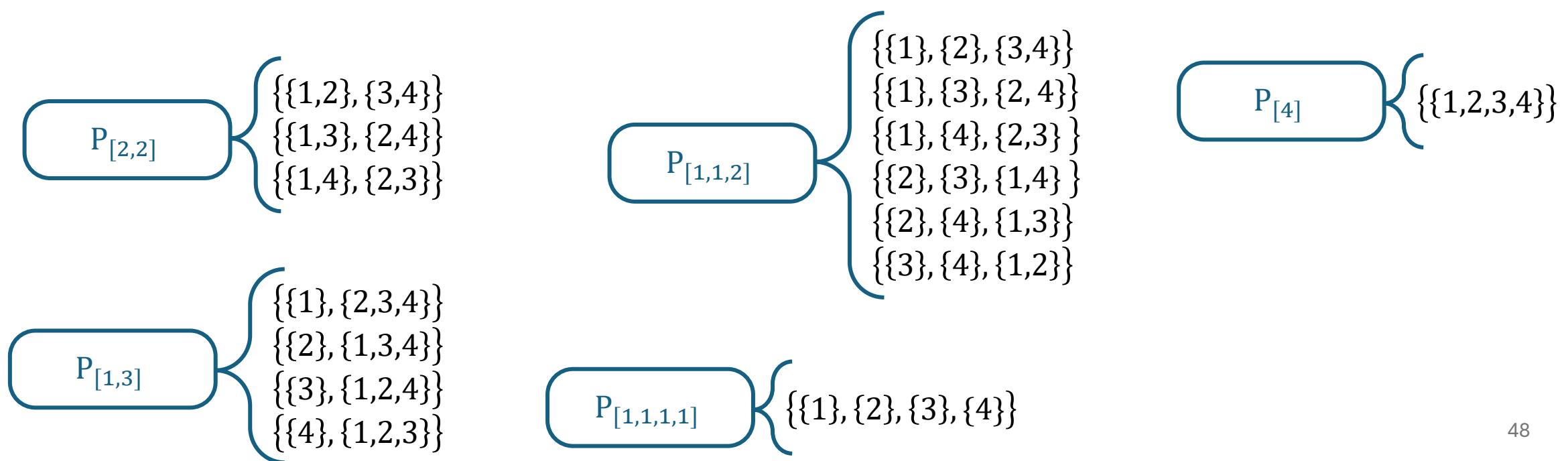
Assume we have 4 agents $Ag = \{1, 2, 3, 4\}$, how we split the search space into sub-spaces?



IP Method: Example (1)

P ₁	value	P ₂	value	P ₃	value	P ₄	value
{1}	125	{1, 2}	175	{1, 2, 3}	200	{1, 2, 3, 4}	425
{2}	50	{1, 3}	150	{1, 2, 4}	150		
{3}	75	{1, 4}	100	{1, 3, 4}	300		
{4}	150	{2, 3}	150	{2, 3, 4}	150		
		{2, 4}	200				
		{3, 4}	125				

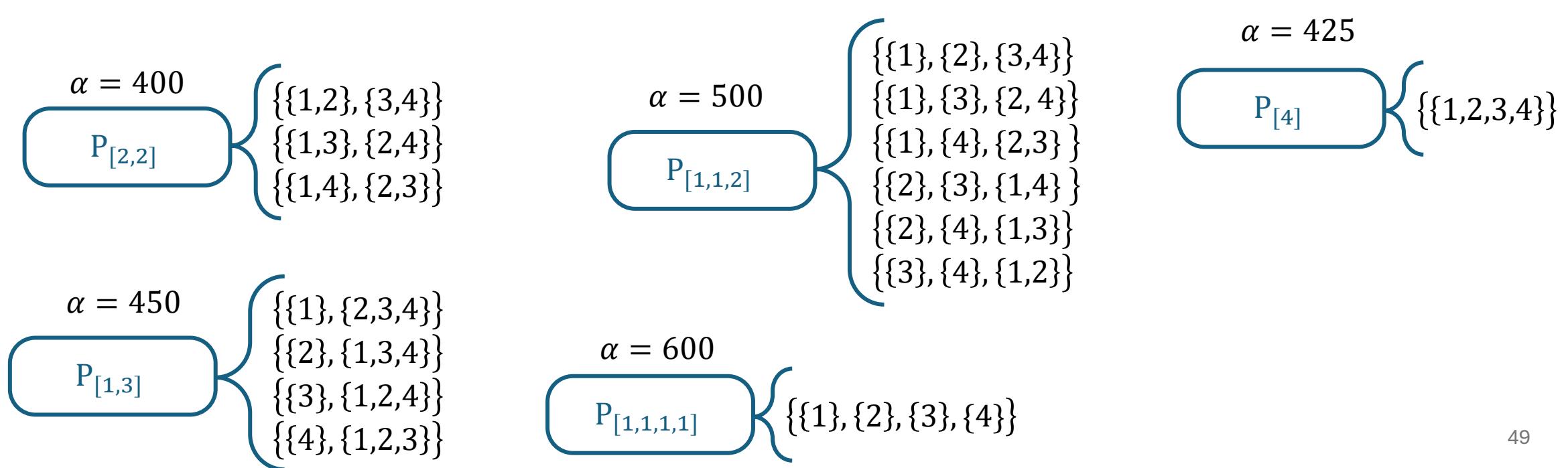
Calculate the upper bound $\alpha^* = 600$



IP Method: Example (1)

P ₁	value	P ₂	value	P ₃	value	P ₄	value
{1}	125	{1, 2}	175	{1, 2, 3}	200	{1, 2, 3, 4}	425
{2}	50	{1, 3}	150	{1, 2, 4}	150		
{3}	75	{1, 4}	100	{1, 3, 4}	300		
{4}	150	{2, 3}	150	{2, 3, 4}	150		
		{2, 4}	200				
		{3, 4}	125				

Calculate the upper bound $\alpha^* = 600$



IP Method: Example (1)

P ₁	value	P ₂	value	P ₃	value	P ₄	value
{1}	125	{1, 2}	175	{1, 2, 3}	200	{1, 2, 3, 4}	425
{2}	50	{1, 3}	150	{1, 2, 4}	150		
{3}	75	{1, 4}	100	{1, 3, 4}	300		
{4}	150	{2, 3}	150	{2, 3, 4}	150		
		{2, 4}	200				
		{3, 4}	125				
Avg	100		150		200		425

Calculate the upper bound $\alpha^* = 600$

Calculate the lower bound $\beta^* = 425$

$$\alpha = 400$$

$$P_{[2,2]}$$

$$\beta = 300$$

$$\left\{ \begin{array}{l} \{1,2\}, \{3,4\} \\ \{1,3\}, \{2,4\} \\ \{1,4\}, \{2,3\} \end{array} \right.$$

$$\alpha = 500$$

$$P_{[1,1,2]}$$

$$\beta = 350$$

$$\left\{ \begin{array}{l} \{1\}, \{2\}, \{3,4\} \\ \{1\}, \{3\}, \{2,4\} \end{array} \right.$$

$$\left\{ \begin{array}{l} \{1\}, \{4\}, \{2,3\} \\ \{2\}, \{3\}, \{1,4\} \end{array} \right.$$

$$\left\{ \begin{array}{l} \{2\}, \{4\}, \{1,3\} \\ \{3\}, \{4\}, \{1,2\} \end{array} \right.$$

$$\alpha = 425$$

$$P_{[4]}$$

$$\beta = 425$$

$$\alpha = 450$$

$$P_{[1,3]}$$

$$\beta = 300$$

$$\left\{ \begin{array}{l} \{1\}, \{2,3,4\} \\ \{2\}, \{1,3,4\} \\ \{3\}, \{1,2,4\} \\ \{4\}, \{1,2,3\} \end{array} \right.$$

$$\alpha = 600$$

$$P_{[1,1,1,1]}$$

$$\beta = 400$$

IP Method: Example (2)

P ₁	value	P ₂	value	P ₃	value	P ₄	value	P ₅	value
{1}	20	{1, 2}	40	{1, 2, 3}	65	{1, 2, 3, 4}	85	{1, 2, 3, 4, 5}	165
{2}	15	{1, 3}	30	{1, 2, 4}	55	{1, 2, 3, 5}	140		
{3}	25	{1, 4}	20	{1, 2, 5}	70	{1, 2, 4, 5}	90		
{4}	30	{1, 5}	45	{1, 3, 4}	60	{1, 3, 4, 5}	75		
{5}	10	{2, 3}	40	{1, 3, 5}	75	{2, 3, 4, 5}	110		
		{2, 4}	65	{1, 4, 5}	55				
		{2, 5}	20	{2, 3, 4}	70				
		{3, 4}	30	{2, 3, 5}	75				
		{3, 5}	45	{2, 4, 5}	65				
		{4, 5}	15	{3, 4, 5}	60				
Avg	20		35		65		100		165

We have 5 Agents

$$P_{[5]} \\ \beta = 165, \alpha = 165$$

$$P_{[1,4]} \\ \beta = 120, \alpha = 170$$

$$P_{[2,3]} \\ \beta = 100, \alpha = 140$$

The super upper bound $\alpha^* = 170$

$$P_{[1,1,3]} \\ \beta = 105, \alpha = 135$$

$$P_{[1,2,2]} \\ \beta = 90, \alpha = 160$$

The super lower bound $\beta^* = 165$

$$P_{[1,1,1,2]} \\ \beta = 95, \alpha = 155$$

$$P_{[1,1,1,1,1]} \\ \beta = 100, \alpha = 150$$

IP Method: Example (2)

P ₁	value	P ₂	value	P ₃	value	P ₄	value	P ₅	value
{1}	20	{1, 2}	40	{1, 2, 3}	65	{1, 2, 3, 4}	85	{1, 2, 3, 4, 5}	165
{2}	15	{1, 3}	30	{1, 2, 4}	55	{1, 2, 3, 5}	140		
{3}	25	{1, 4}	20	{1, 2, 5}	70	{1, 2, 4, 5}	90		
{4}	30	{1, 5}	45	{1, 3, 4}	60	{1, 3, 4, 5}	75		
{5}	10	{2, 3}	40	{1, 3, 5}	75	{2, 3, 4, 5}	110		
		{2, 4}	65	{1, 4, 5}	55				
		{2, 5}	20	{2, 3, 4}	70				
		{3, 4}	30	{2, 3, 5}	75				
		{3, 5}	45	{2, 4, 5}	65				
		{4, 5}	15	{3, 4, 5}	60				
Avg	20		35		65		100		165

We have 5 Agents

$$P_{[5]} \\ \beta = 165, \alpha = 165$$

$$P_{[1,4]} \\ \beta = 120, \alpha = 170$$

$$\cancel{P_{[1,1,3]}} \\ \beta = 105, \alpha = 135$$

$$\cancel{P_{[1,1,1,2]}} \\ \beta = 95, \alpha = 155$$

Answer is
 $\{\{4\}, \{1, 2, 3, 5\}\}$

$$\cancel{P_{[2,3]}} \\ \beta = 100, \alpha = 140$$

$$\cancel{P_{[1,2,2]}} \\ \beta = 90, \alpha = 160$$

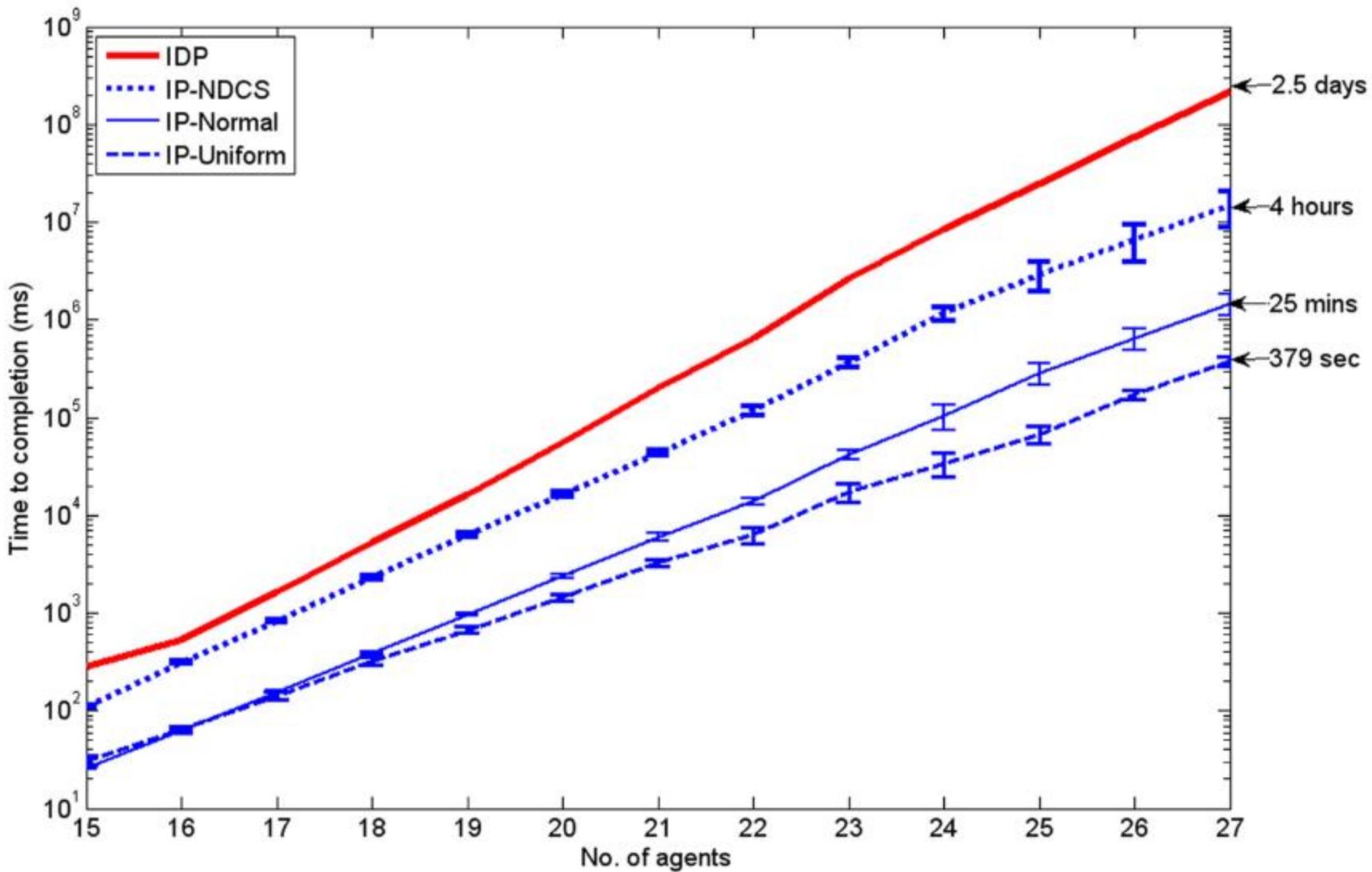
$$\cancel{P_{[1,1,1,1,1]}} \\ \beta = 100, \alpha = 150$$

The super upper bound $\alpha^* = 170$

The super lower bound $\beta^* = 165$

Let's remove the outbound sub-spaces

IP Method



Time to find the optimal solution

(n,s)-sequences

- Riley et.al. proposed a mechanism for calculating *the coalition value calculation share* for an agent, based on the agent's id
 - Given a set of agents in Ag where $n = |Ag|$, the agents are labelled $1 \dots n$
 - A coalition of size $1 \leq s < n$ can be generated given an (n,s)-sequence t by first calculating the aggregate offset for each position in t and given the agent x , determine its coalition value calculation share (for some $1 \leq x \leq n$):

$$x_i \equiv \begin{cases} x & \text{if } i = 1 \\ \left(x + \sum_{k=0}^{i-2} (t_k + 1) \bmod n \right) & \text{if } 2 \leq i \leq s \end{cases}$$

- Note that the result is an agent in the range ($1 \leq x \leq n$)
 - i.e. the result is congruent modulo n

Example

- The (n,s) -sequences for coalitions of size $s=3$, for a set of agents $A_g = \{1,2,3,4,5,6\}$ are $\langle 0,0,3 \rangle, \langle 0,1,2 \rangle, \langle 0,2,1 \rangle, \langle 1,1,1 \rangle$
 - These are used to generate the coalition value calculation shares for each agent x
 - If each agent generates their share...
 - ... all of the coalitions of size s will be generated
 - Duplications occur if there is a repeated periodic sub-sequence in the $(n-s)$ -sequence (e.g. $\langle 1,1,1 \rangle$)
 - If $s=4$, then $\langle 0,1,0,1 \rangle$ has the repeating sub-sequence $\langle \dots 0,1 \dots \rangle$, but $\langle 0,0,1,1 \rangle$ has no repeating sequence
 - By tracking which agent generates coalitions from the repeated sequence, duplications can be eliminated

Generating Coalition Value Calculation Shares

The following table lists the coalitions generated for the (n,s) -sequences where $n=6$ and $s=3$

	$\langle 0,0,3 \rangle$	$\langle 0,1,2 \rangle$	$\langle 0,2,1 \rangle$	$\langle 1,1,1 \rangle$
CV^3_1	1,2,3	1,2,4	1,2,5	
CV^3_2	2,3,4	2,3,5	2,3,6	
CV^3_3	3,4,5	3,4,6	3,4,1	
CV^3_4	4,5,6	4,5,1	4,5,2	4,6,2
CV^3_5	5,6,1	5,6,2	5,6,3	5,1,3
CV^3_6	6,1,2	6,1,3	6,1,4	

$$x_i \equiv \begin{cases} x & \text{if } i = 1 \\ (x + \sum_{k=0}^{i-2} (t_k + 1)) \bmod n & \text{if } 2 \leq i \leq s \end{cases}$$

Example

- The (n,s) -sequences for coalitions of size $s=3$, for a set of agents $A_g = \{1,2,3,4,5,6\}$ are $\langle 0,0,3 \rangle$, $\langle 0,1,2 \rangle$, $\langle 0,2,1 \rangle$, $\langle 1,1,1 \rangle$
- Therefore, if we consider agent 5:
 - $C(5, \langle 0,0,3 \rangle) \equiv \{5, 6, 1\}$
 - i.e. $\{5, (5+0+1) \bmod 6, ((5+0+1) + 0 + 1) \bmod 6\} \equiv \{5, 6, 1\}$
 - $C(5, \langle 0,1,2 \rangle) \equiv \{5, 6, 2\}$
 - i.e. $\{5, (5+0+1) \bmod 6, ((5+0+1) + 1 + 1) \bmod 6\} \equiv \{5, 6, 2\}$
 - $C(5, \langle 0,2,1 \rangle) \equiv \{5, 6, 3\}$
 - i.e. $\{5, (5+0+1) \bmod 6, ((5+0+1) + 2 + 1) \bmod 6\} \equiv \{5, 6, 3\}$
 - $C(5, \langle 1,1,1 \rangle) \equiv \{5, 1, 3\}$
 - i.e. $\{5, (5+1+1) \bmod 6, ((5+1+1) + 1 + 1) \bmod 6\} \equiv \{5, 1, 3\}$

Generating Coalition Value Calculation Shares

The following table lists the coalitions generated for the (n,s) -sequences where $n=6$ and $s=3$

	$\langle 0,0,3 \rangle$	$\langle 0,1,2 \rangle$	$\langle 0,2,1 \rangle$	$\langle 1,1,1 \rangle$
CV^3_1	1,2,3	1,2,4	1,2,5	
CV^3_2	2,3,4	2,3,5	2,3,6	
CV^3_3	3,4,5	3,4,6	3,4,1	
CV^3_4	4,5,6	4,5,1	4,5,2	4,6,2
CV^3_5	5,6,1	5,6,2	5,6,3	5,1,3
CV^3_6	6,1,2	6,1,3	6,1,4	

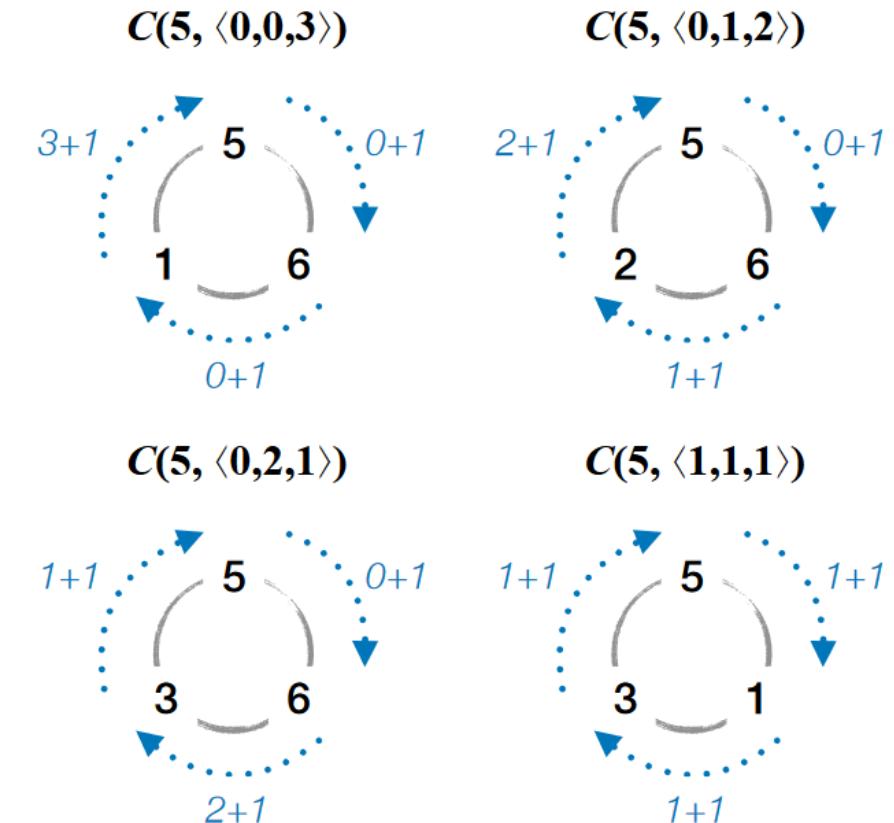
$$x_i \equiv \begin{cases} x & \text{if } i = 1 \\ (x + \sum_{k=0}^{i-2} (t_k + 1)) \bmod n & \text{if } 2 \leq i \leq s \end{cases}$$

Example

- The (n,s) -sequences for coalitions of size $s=3$, for a set of agents $A_g = \{1,2,3,4,5,6\}$ are $\langle 0,0,3 \rangle$, $\langle 0,1,2 \rangle$, $\langle 0,2,1 \rangle$, $\langle 1,1,1 \rangle$

- Therefore, if we consider agent 5:

- $C(5, \langle 0,0,3 \rangle) \equiv \{5, 6, 1\}$
 - i.e. $\{5, (5+0+1) \bmod 6, ((5+0+1) + 0 + 1) \bmod 6\} \equiv \{5, 6, 1\}$
- $C(5, \langle 0,1,2 \rangle) \equiv \{5, 6, 2\}$
 - i.e. $\{5, (5+0+1) \bmod 6, ((5+0+1) + 1 + 1) \bmod 6\} \equiv \{5, 6, 2\}$
- $C(5, \langle 0,2,1 \rangle) \equiv \{5, 6, 3\}$
 - i.e. $\{5, (5+0+1) \bmod 6, ((5+0+1) + 2 + 1) \bmod 6\} \equiv \{5, 6, 3\}$
- $C(5, \langle 1,1,1 \rangle) \equiv \{5, 1, 3\}$
 - i.e. $\{5, (5+1+1) \bmod 6, ((5+1+1) + 1 + 1) \bmod 6\} \equiv \{5, 1, 3\}$



Summary

- In this lecture we have looked at mechanisms for identifying coalitions.
 - The notion of a stable coalition game was presented, through the idea of a Core
 - The Shapley Value was then introduced, to determine the contribution that different agents may have on a coalition.
- The problem of representing coalitional games and characteristic functions was then discussed, including:
 - Induced Subgraphs
 - Marginal Contribution Nets.
- We finally talked about Coalition Structure Generation

Readings for this week

- Chapters 13 of the book by M.Wooldridge “An introduction to Multi-Agent Systems” (2nd edition).