02285 AI and MAS, SP18
Multiagent communication and cooperation

Curriculum for week 6: Only slides. Optional: Chapter 7 and 8 of Michael Wooldridge's "An Introduction to MultiAgent Systems", 2ed, Wiley 2009.

Todays subjects:

- **Multiagent communication**: Speech acts and the FIPA Agent Communication Language.
- **Multiagent cooperation**: Cooperative distributed problem solving, the Contract Net protocol.
- **Plan coordination**: Partial global planning, plan merging.

Multibody vs. multiagent approach

Repetition. Two different approaches to solving multi-**actor** problems:

- Multi-**body** approach. **Centralised** AI that control all actors.
- Multi-**agent** approach. **Decentralised** solution. Each actor is an autonomous agent.

Advantages of each approach:

- **Multibody approach**. No advanced architectures for agent communication and cooperation required. Robust against deadlocks.
- **Multiagent approach**. No complexity blowup in case of loosely coupled agents. Robust wrt. failure of individual agents. Only possible solution for, say, a team of domestic robots from different manufactures.

In this lecture we consider the **multiagent approach** and how to deal with communication and cooperation.

Communication: Speech acts

Most treatments of communication in (multi-)agent systems borrow their inspiration from **speech act theory** (due to the work of the philosopher John Austin in the 1960's).

Underlying idea: Utterances should be treated as any other type of "physical action". As with physical actions, utterances can change the state of the world, e.g. "I now pronounce you man and wife."

A speech act can be seen to have two components:

- **Performative verb**. E.g. request, inform, commit to, promise, thank, inquire, declare.
- **Propositional content**. E.g. "the door is closed", "you are man and wife".

Examples of performatives

Many speech acts with the same **propositional content** can be formed by varying the corresponding **performative verb**.

Example. Various speech acts with the propositional content "the door is closed":

- Performative: **request**. "Please close the door".
- Performative: **inform**. "The door is closed".
- Performative: **promise**. "I promise to close the door".
- Performative: **thank**. "Thanks for closing the door".
- Performative: **inquire**. "Is the door closed?".
- Performative: **declare**. "I hereby declare the door closed!"

Agent communication languages

Agent communication languages (**ACL**s) are languages developed specifically for agent communication. They typically rely on **speech act theory**.

An agent communication language defines a class of **messages**: the messages that agents can send to each other.

One of these languages is the **FIPA Agent Communication Language** (**FIPA-ACL**)...

The FIPA Agent Communication Language

A **message** in the FIPA Agent Communication Language has a **header** which is a **performative**, e.g.:

- **Inquiries and announcements**:
  1. `query-if`. "Is box B at location L?"
  2. `inform`. "Box B is at location L."
- **Requests**:
  3. `request`. "Please move Box B to location L."
  4. `agree`. "OK, I'll do it."
  5. `refuse`. "I won't do it."
  6. `failure`. "I didn't manage to do it."
- **Proposals**:
  6. `cfp`. Call for proposal. "Send me your proposals for moving box B to location L within time T."
  7. `propose`. "I can do it by ..."
  8. `accept-proposal`. "OK, I accept your proposal."
  9. `reject-proposal`. "No, that's not good enough."

FIPA-ACL contains 11 performatives in addition to the above 9.

The FIPA Agent Communication Language

The **body** of a FIPA-ACL message declares the **sender** of the message, the **receiver** of the message, and the **content** of the message.

Example. The following are well-formed FIPA-ACL messages:

```
(inform
  :sender Agent1
  :receiver Agent2
  :content (location Box1 Location1))

(query-if
  :sender Agent1
  :receiver Agent2
  :content (> (weight Box1) (weight Box2))
```

FIPA-ACL messages can contain further parameters in the body, such as **language** and **ontology** (which language and ontology are being used).

Benevolent vs. self-interested agents

Two types of agents, **benevolent** and **self-interested**.

Benevolent agents:

- Agents share a common overall goal.
- No potential for conflict (of interests).
- Agents help each other whenever needed.

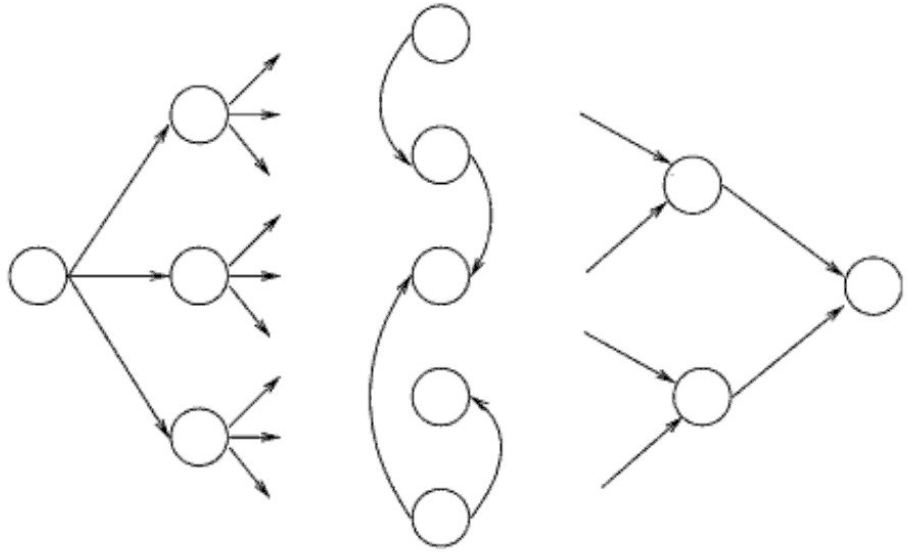
Self-interested agents:

- Agents have individual goals.
- Individual goals might conflict.
- Agents act in their own interest, possibly at the expense of others.

Systems of benevolent agents are simpler to design.

Problem-solving in systems of benevolent agents is called **cooperative distributed problem solving** (**CDPS**). This will be our focus in the following.

Cooperative distributed problem solving (CDPS)



Three-stage activity of CDPS agents:

- **Problem decomposition**. Decompose the problem into subproblems. Possibly hierarchically as in HTNs.
- **Subproblem solution**. Solve the individual subproblems. This stage typically involves information sharing between agents.
- **Solution synthesis**: Solutions to individual subproblems are integrated into an overall solution.

CDPS: Task sharing and result sharing

The three-stage activity above involves two cooperative problem-solving activities: **task sharing** and **result sharing**.

Task sharing consists of:

- **Decomposition** of the problem into smaller subproblems.
- **Allocation** of subproblems to the individual agents.

Result sharing: Information (partial results etc.) is distributed. Either **proactively** (agents send information they believe to be of interest to others) or **reactively** (as responses to requests from other agents).

Example. Martha project.

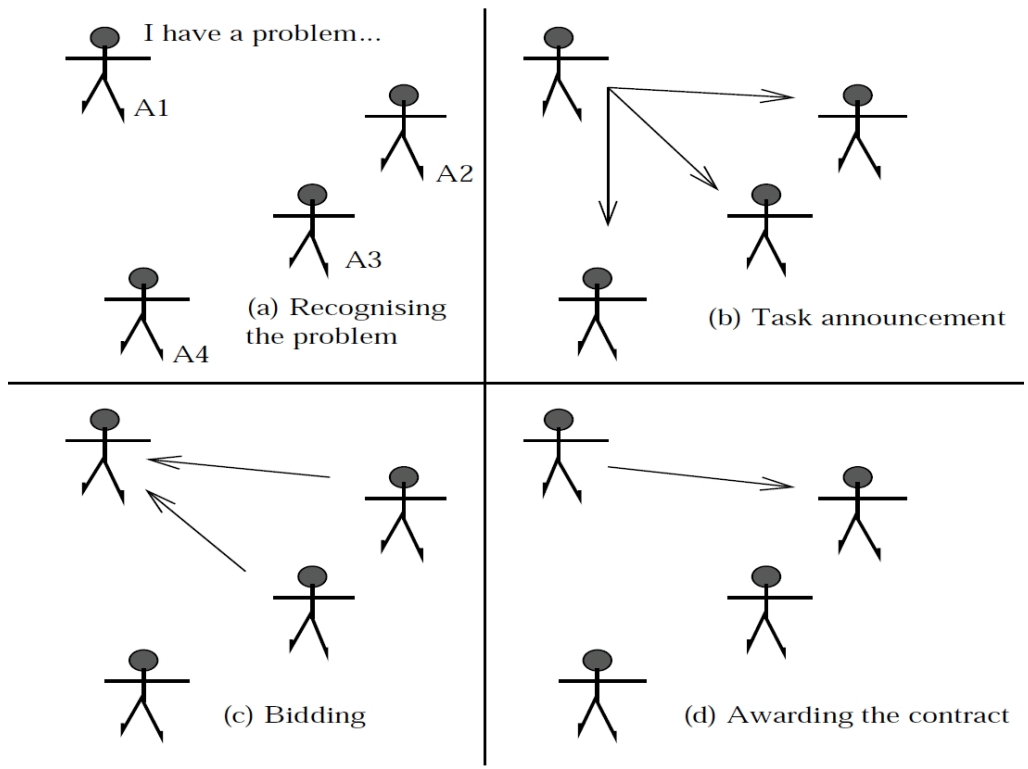
A widespread protocol for **task allocation** (second component of task sharing) is the **Contract Net (CNET) protocol**. It is based on the idea of **contracting** in companies...

The Contract Net protocol

The 3 core elements of the Contract Net (CNET) protocol are **announcement**, **bidding** and **awarding**.

Solving problems using CNET involves the following steps:

1. **Problem recognition**. What is the problem to be solved?
2. **Task announcement**. Agent 1: "Can anybody help me with this?"
3. **Bidding** for the task. Agent 2: "I can do it in time t_2." Agent 3: "I can do it in time t_3."
4. **Awarding** the task. Agent 1: "Agent 2 wins the bid".
5. **Expediting** task. Agent 2 carries out the contracted task.



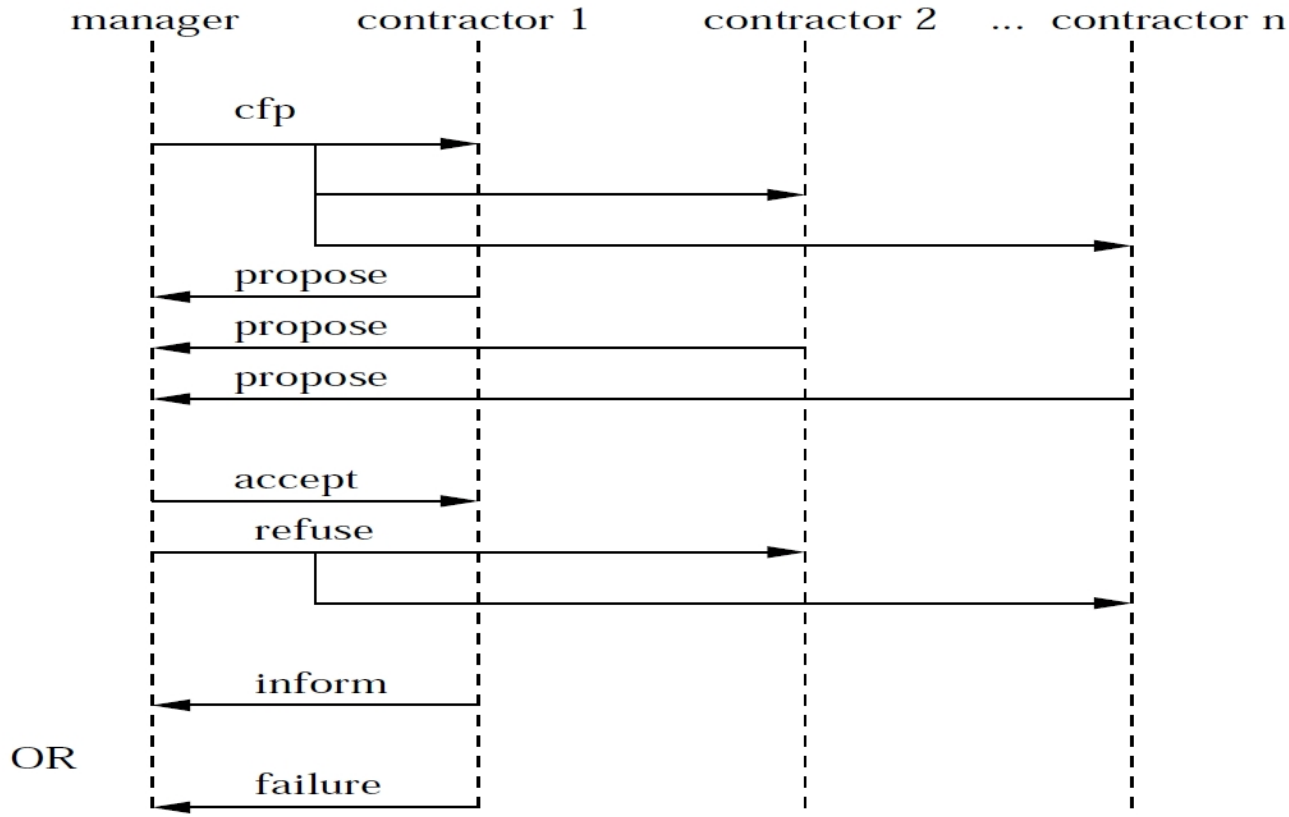
More on the Contract Net

1. **Problem recognition**. An agent A recognises it has a problem it wants help with. Either because it **can not** or because it **prefers not** to acheive its goal in isolation.
2. **Task announcement**. Agent A broadcasts an announcement of the task, including the deadline for bidding.
3. **Bidding**. Agents can decide to bid for the task. A bid must include the capabilities of the bidder relevant to the execution of the announced task.
4. **Awarding**. Agent A selects the most appropriate bid based on the information in the bids. The selection is communicated to the bidders through an **award** message.
5. **Expediting**. The succesful **contractor** expedites the task.

The Contract Net via FIPA-ACL messages

FIPA-ACL messages can be used to realise the Contract Net protocol in multiagent-systems using the following performatives:

- `cfp` (call for proposals). Used for **announcing** a task.
- `propose`, `refuse`. Used for **bidding**.
- `accept-proposal`, `reject-proposal`. Used for **awarding**.
- `inform`, `failure`. Used to indicate completion or failure.



Result sharing and the blackboard architecture

Recall: **Result sharing** is the distribution of information (partial results etc.) between agents in CDPS. E.g. the routes calculated by individual dock-worker robots.

A simple architecture for result sharing is the **blackboard architecture**.

Blackboard architecture: Results are shared via a shared data structure (the blackboard). All agents can read and write to the blackboard. Agents write partial solutions to it.

Metaphor: A group of specialists working as a team to brainstorm a solution to a problem, using a **blackboard** as the workplace. They take turn to add contributions to the blackboard until the problem has been solved.

Coordination: Positive and negative relationships

One of the most essential problems in **result sharing** is **coordination**, that is, managing inter-dependencies between the activities of agents.

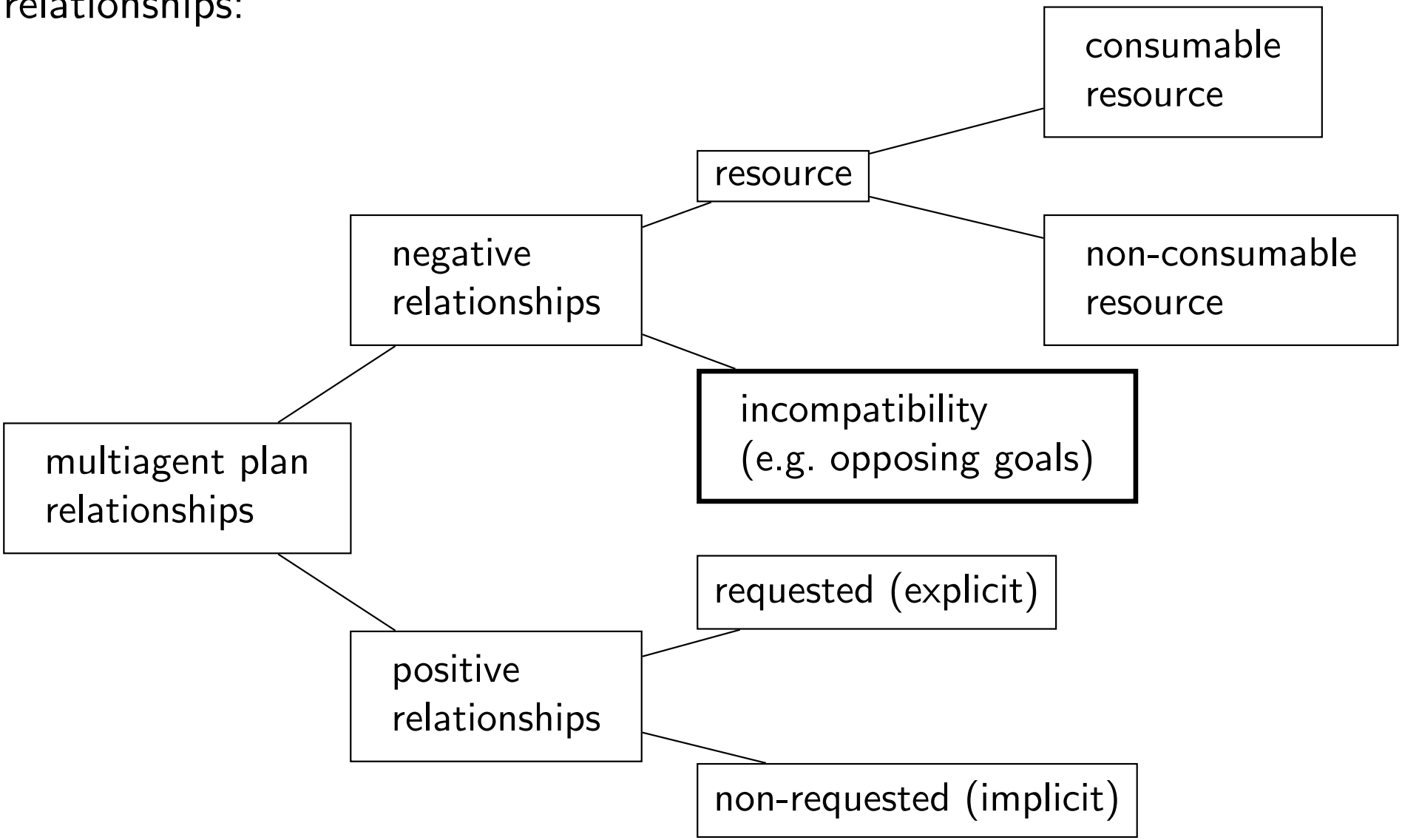
Coordination relationships can be either be **negative** or **positive**.

Negative relationship example: Two programming project agents planning to simultaneously move two distinct boxes into the same location (i.e. planning to use the same resource simultaneously).

Positive relationship example: One agent planning to clear a box from a path thereby allowing another agent to pass through (i.e. one agent helping another).

Typology of coordination relationships

The following tree illustrates the possible agent coordination relationships:



Subgoal interaction

Suppose each agent has its own subgoal (or a conjunctive goal of a single agent is split into subgoals).

Types of **interaction** between subgoals:

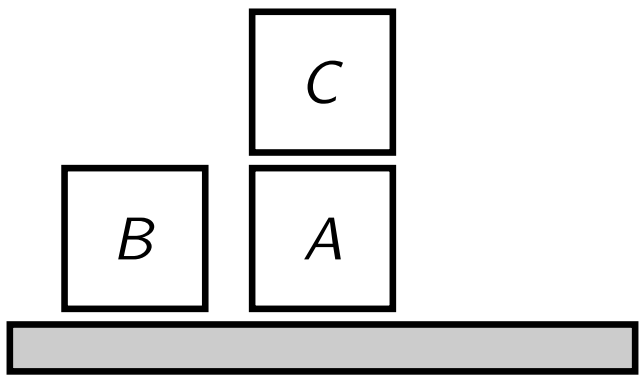
- **Independent subgoals**. Independent subgoals do not interfere with each other, and can be solved serially in any order. E.g. two dock-worker robots working in disjoint parts of a harbour.
- **Serialisable subgoals**. Serialisable subgoals can be solved serially in some orders but not in others. E.g. A can open a door so that B can afterwards pass through, but B can not first pass through the closed door, and then let agent A open it afterwards.
- **Non-serialisable subgoals**. Subgoals not allowing serialisation (e.g. Sussman anomaly). However, it might still be possible to perform a more advanced **merge** of the plans for the individual subgoals into one overall plan that achieves all subgoals, see later slides.

Subgoal interaction example

Consider the **non-serialisable subgoals** $On(A, B)$ and $On(B, C)$ in the blocks world below. Suppose one agent is given the subgoal $On(A, B)$ and another the subgoal $On(B, C)$.

Suppose further that the two agents are constructing individual plans for their individual subgoals. With no coordination and interleaving of their plans, they can not achieve both subgoals, cf. the **Sussman anomaly** considered in one of the exercises.

Thus coordination and possibly communication between agents are required to solve such problems.



Start State

Social laws

One way to achieve coordination is by **social laws**, as discussed last week. These social laws can be either:

- Hardwired into the agents (**offline design**).
- **Emerge** in run-time by agreement between the agents.

The first approach is the simplest and often most efficient, but is less flexible and more domain-specific.

What simple social law could be used to solve the coordination problem of dock-worker robots passing through crossroads simultaneously?

Partial global planning

Partial global planning (Durfee, 1988) is an approach to deal with coordination of plans and intentions in multiagent systems. It involves three iterated stages:

1. **Local plan formulation**. Each agent decides what its own goals are and generates short-term plans (**local plans**) in order to achieve them. "Short-term" means that the plans do not necessarily provide a sequence of actions leading *all* the way to the goals.
2. **Partial global plan construction and modification**. Local plans are integrated into a **partial global plan** capturing where local plans and goals interact. The partial global plan is modified to yield a more coordinated plan (e.g. avoiding redundancy and conflicts).
3. **Acting**. The partial global plan is translated back to the local level, and carried out by the individual agents.

Plan merging

Assume:

- The assignment of goals to a set of agents has been done (e.g. through task-sharing or because of inherent distributivity of the domain).
- The agents have constructed individual plans for their goals.

The problem then becomes how to **merge** the individual plans into a conflict-free multiagent plan achieving all goals.

Plan merging refers to this process: The scheduling/synchronisation of the individual plans.

SELF-STUDY

Concurrent action lists

Consider now the simplified Gripper problem involving two gripper arms (agents), 1 and 2. When constructing a **joint plan** to solve this problem, we must avoid 1 and 2 trying to move the same box at the same time.

Concurrent action list of an action a: List stating which actions must or must not be executed concurrently with a.

Example. The *Move* action could be described as follows:

ACTION: $Move(a, x, y, z)$
CONCURRENT: $\forall b(Agent(b) \wedge b \neq a \rightarrow \neg Move(b, x, y, z))$
PRECONDITION: $In(x, y)$
EFFECT: $In(x, z) \wedge \neg In(x, y)$

Alternatively, it might be that the boxes are so heavy that it takes two agents to lift one:

ACTION: $Move(a, x, y, z)$
CONCURRENT: $\exists b(Agent(b) \wedge b \neq a \wedge Move(b, x, y, z))$
PRECONDITION: $In(x, y)$
EFFECT: $In(x, z) \wedge \neg In(x, y)$

Plan merging

Suppose we are given a set of agents $A_1, \ldots, A_n$, and that each agent A_i has produced its own plan $a^i_1, a^i_2, \ldots, a^i_{m_i}$.

Now consider the following **joint plan**:

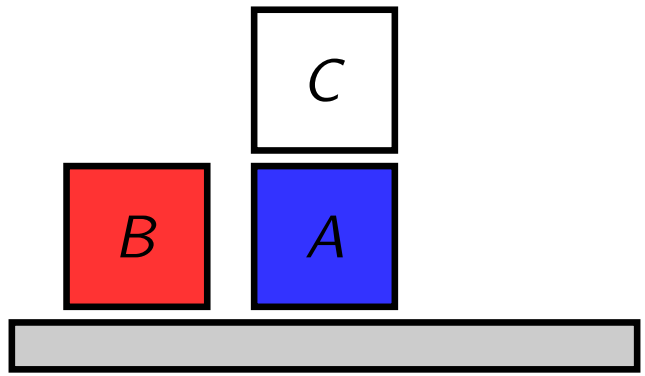
$$
\begin{aligned}
&A_1 : a^1_1, a^1_2, \ldots, a^1_{m_1} \\
&A_2 : a^2_1, a^2_2, \ldots, a^2_{m_2} \\
&\vdots \\
&A_n : a^n_1, a^n_2, \ldots, a^n_{m_n}
\end{aligned}
$$

It is now possible to check whether this constitutes a consistent joint plan, e.g. using the concurrent action lists of the involved actions. If it does, all is well.

Otherwise, **conflicts** have to be resolved.

Plan merging example

Consider a blue and a red agent with individual goals $On(A, B)$ and $On(B, C)$ in the blocks world below (e.g. the blue agent can only move blue and white blocks, and the red only red and white):

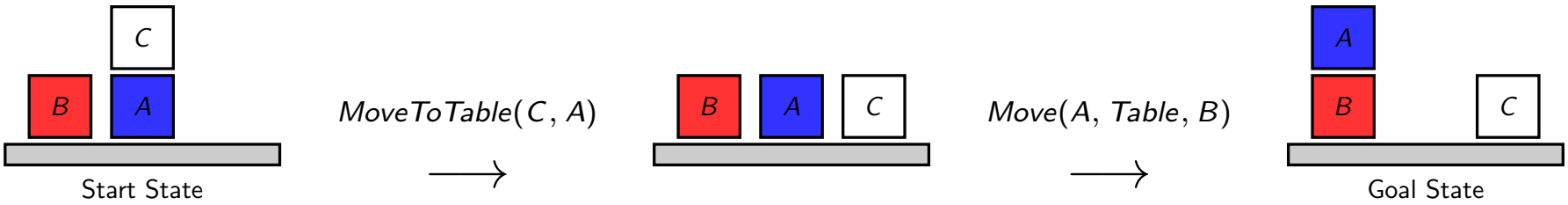


Start State

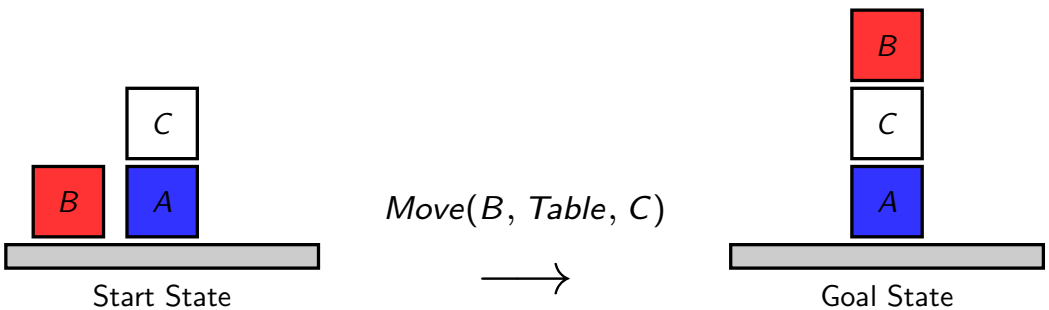
The two agents might come up with the individual plans shown on the next slide...

Plan merging example (cont'd)

Blue agent's plan for achieving $On(A, B)$:



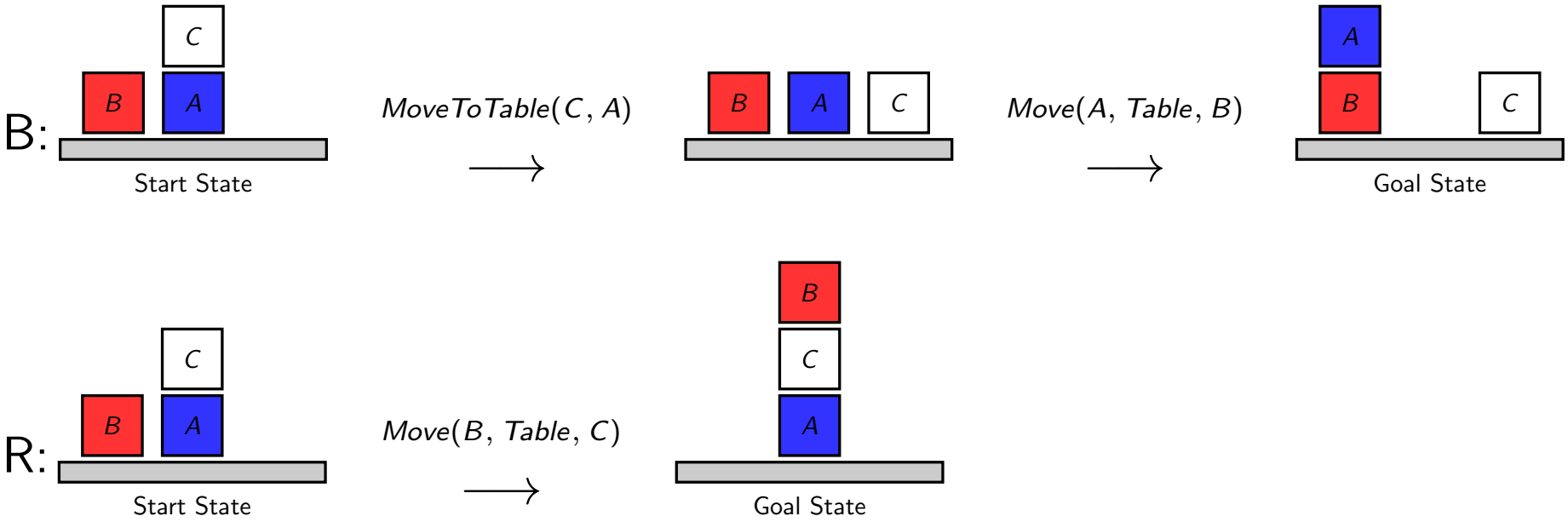
Red agent's plan for achieving $On(B, C)$:



Thus the naive joint plan looks like this:

$$Blue : MoveToTable(C, A), Move(A, Table, B)$$
$$Red : Move(B, Table, C)$$

However, there are **conflicts**: block C cannot be moved while something is moved on top of it (by concurrent action list)...



Solution: Blue performs its first action before red's first. Why not vice versa?

After this, red performs its first action. Why not blue's second action?

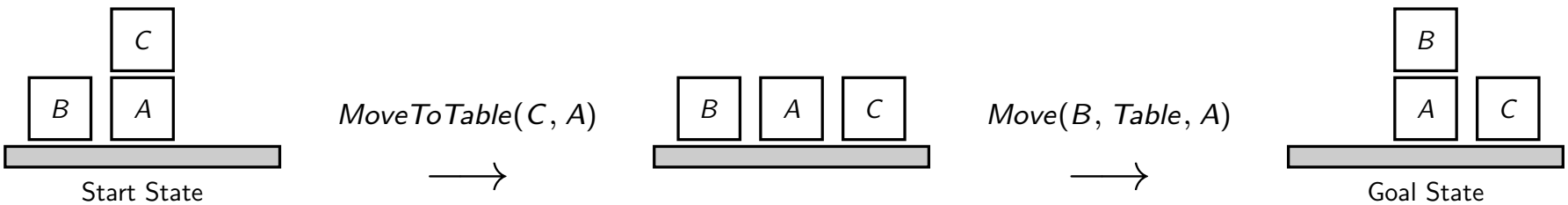
These **conflict resolutions** result in the following successful joint plan:

$$Blue : MoveToTable(C, A), NoOp, Move(A, Table, B)$$
$$Red : NoOp, Move(B, Table, C), NoOp$$

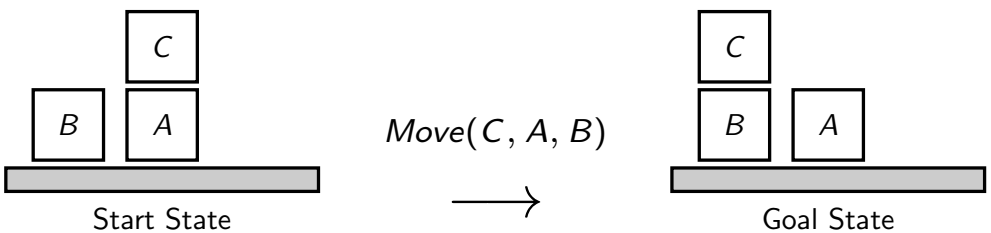
Plan merging example

Consider now instead individual goals $On(B, A)$ and $On(C, B)$ (with no constraints on who can move which blocks).

Blue's plan for achieving $On(B, A)$:



Red's plan for achieving $On(C, B)$:



Can these two individual plans be merged?

Plan merging: Final comments

- Plan merging is particularly suited for largely decoupled problems.
- Plan merging is not sufficient for all multiagent planning problems.
- Still, large parts of the plans can often be merged. The remaining parts can be solved by multibody planning (as in the Martha project).
- A more advanced solution is **iterative plan formation**: Using iterative refinements of plans (as in HTN), solving conflicts and replanning as required. (Durfee, 2001).

Concluding remarks

Distributed problem solving and distributed planning are still areas that are far from fully developed.

There are yet no commonly agreed "standard techniques" and no solid conceptual basis. A lot of further research is still needed. To quote a couple of sources:

- Durfee, 2001: "[Strategies for distributed problem solving] still has more "art" to it than we like to see in an engineering discipline."
- Russell & Norvig, 2009: "Methods for efficient planning in these kinds of environments—for example, taking advantage of loose coupling—are still in their infancy."

DTU student planning project examples

Some examples of applications of planning methods in DTU projects I have supervised:

2017 *Complexity Results for Epistemic Multi-Agent Planning* (master project). Complexity results for fragments of multi-agent epistemic planning.

2016 *Real-time Strategic Planning for StarCraft* (master project). Planning, goal recognition and operations research methods.

DTU student planning project examples cont'd

2015 *Planning with Implicit Coordination* (bachelor project). How to achieve coordination in multi-agent planning without explicit communication or negotiation.

2015 *AI for the Computer Game "Astro Kid"* (master project). Planning and learning in a 2D puzzle game.

2015 *AI for 2048 and Threes* (master project). Mini-Max, Monte-Carlo Tree Search, etc.

DTU student planning project examples cont'd

Some examples of applications of planning methods in DTU projects I have supervised:

2015 *Action Learning in Automated Planning* (master project). Learning action schemas automatically from experience.

2015 *Construction Planning in the Computer Game "Unclaimed World"* (master project).

2010–2015 Several projects (master, PhD) on **epistemic planning**: Planning for multiple agents with Theory of Mind capabilities.

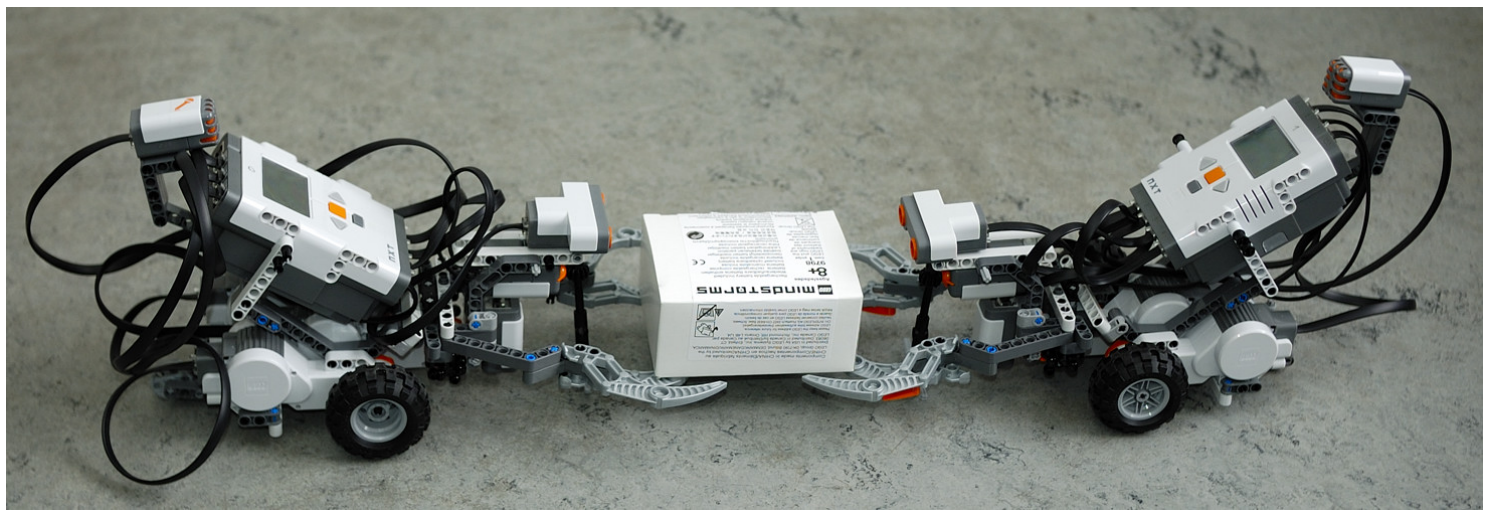
DTU student planning project examples cont'd

2012 *Planning Under Uncertainty for Mission Operations* (master project). Planning for the DTU satellite.

2011 *Implementing Fast Replanning With Regards to Game AI* (bachelor project). Effective means to replan when the world changes unexpectedly.

2010 *Multi-Agent Planning for Robots* (master project). Integrated a *multi-agent planner* with durations, partial observability, online replanning and communication.

2008 *Multi-Agent Robotics Using LEGO NXT Robots* (bachelor project). Using a **POP planner** for controlling Lego NXT robots.

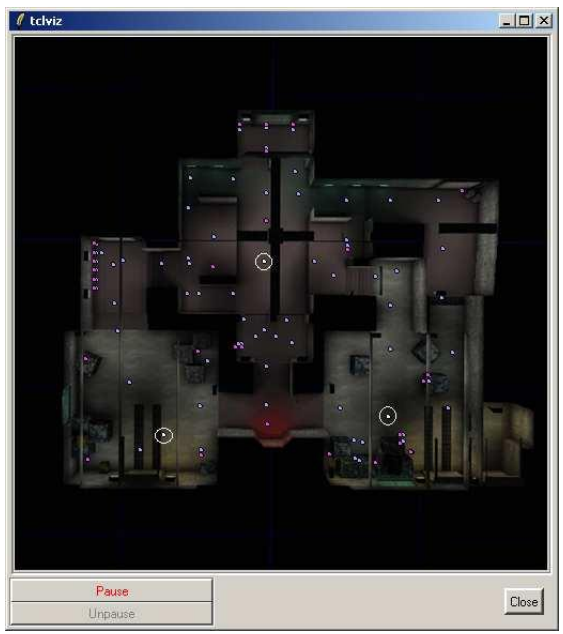


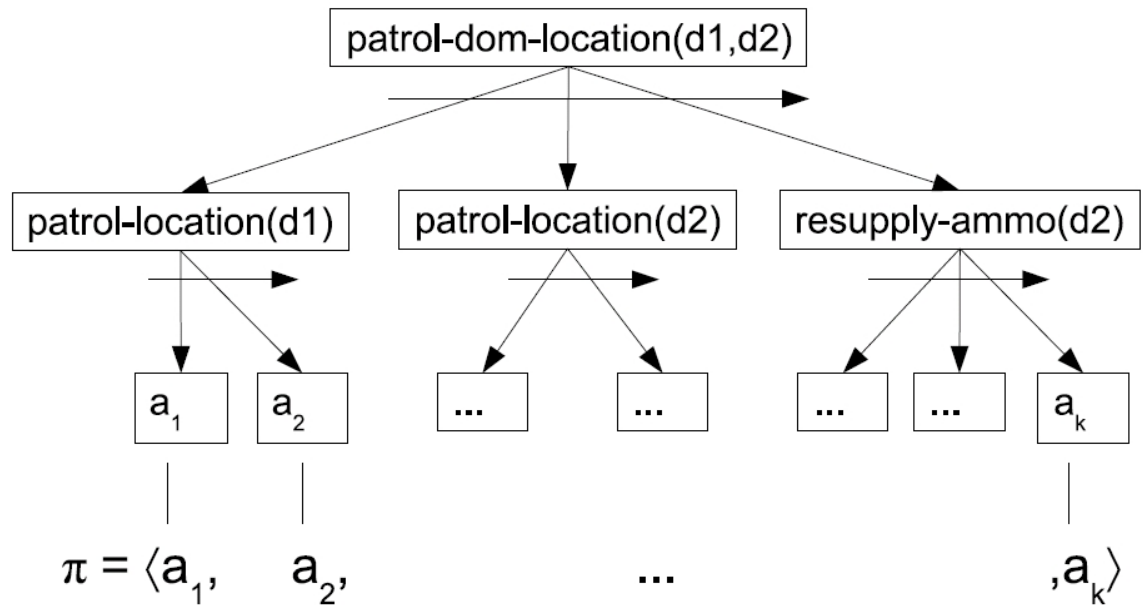
DTU student planning project examples cont'd

2008 *AI Planning in Computer Game Engines* (bachelor project). Implementing **HTN planning** in the Unreal game engine.

2007 *AI planning methods for business software* (master project in collaboration with Microsoft). Performing planning-based **automated tests** of business software.

2005 *AI in Computer Games* (master project in collaboration with IO Interactive). Story-line planning for a first-person shooter (using the GraphPlan algorithm).





Workload and time/project management

Quote from description of programming project:

> *The programming project constitutes approximately 3.75 of the 7.5 ECTS in the course. This means that the expected workload of the programming project is approximately 100 hours per student.*

This is a lot, and you have to think about how to distribute those 100 hours from now on until the report deadline **Wednesday 30 May at 20.00**. Note that all the programming has to be done before **Friday 18 May at 20.00**, as this is the deadline for submitting your competition results.

You might also find it helpful to use e.g. Gantt charts, or some other means to keep track of your estimated workload for the different tasks and their distribution among the group members. This is also a tool to check whether you are on track or whether you have to revise your goals or level of ambition.