Lecture *Hierarchical Planning*

Chapter: *Planning Capabilities Motivated by Real World Applications*

Dr. Pascal Bercher

Institute of Artificial Intelligence,
Ulm University, Germany

Winter Term 2018/2019

(Compiled on: February 20, 2019)

ulm university universität uulm

Overview:

Introduction

Recap: Possible Applications of Planning:

- Autonomous systems, like intelligent factories, robotics.
- Assistance Systems.
- Many more (cf. first lecture).

Issues in such real-world applications:

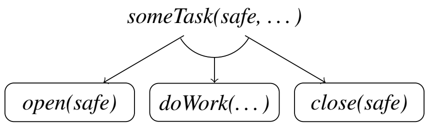
- Plans need to be generated fast: Algorithms and heuristics! ✓
- Plans executed/pursued by humans may need to be recognized. ✓
- We need to be able to cope with execution errors.
- Plans need to be communicated to a user:
  - How to convey the information? Use abstraction?
  - In which order to present the actions?
- Plans should be *explainable*, i.e., we should be able to make clear why actions are within plans.

Introduction

- Planning models have to abstract from many details of the real-world.
- The execution of plans generated using these domains may fail due to these abstractions (determinism and full observability are examples for such abstractions).
- Ordinarily, execution errors are assumed to be unanticipated state changes. This can cover:
  - Some effects of an action did not apply.
  - An action had additional effects.
  - Some “unlikely” effects happened rather than the most likely ones.
  - Some previously unknown facts got known (i.e., something assumed true (wrong) is revealed wrong (true)).
  - The environment unexpectedly changed without the agent causing it.

How to Deal with Execution Errors?

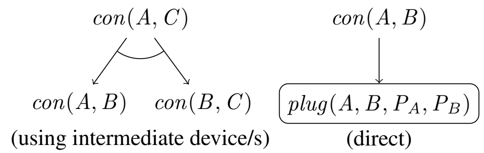
- There are two mechanisms to deal with failed plans:
  - Re-Planning: Start again from the new current state.
  - Plan Repair: Reuse the previous solution and fix (i.e., repair) it according to the unexpected execution problem.
- Simple re-planning discards HTN constraints, i.e., in general it return *false witnesses*, i.e., *wrong* results.
- Example for such a wrong witness?



Consider an execution error after opening the safe but before closing it. What happens?

How to Deal with Execution Errors?

- There are two mechanisms to deal with failed plans:
  - Re-Planning: Start again from the new current state.
  - Plan Repair: Reuse the previous solution and fix (i.e., repair) it according to the unexpected execution problem.
- Simple re-planning discards HTN constraints, i.e., in general it return *false witnesses*, i.e., *wrong* results.
- Example for such a wrong witness?

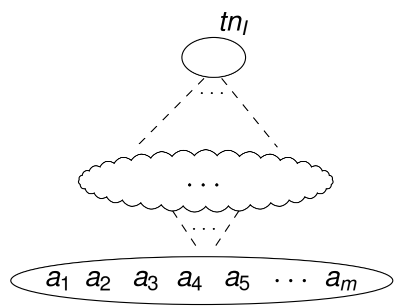


Here, the signal flow is modeled via the hierarchy, not the state. What happens with replanning?

Plan Repair

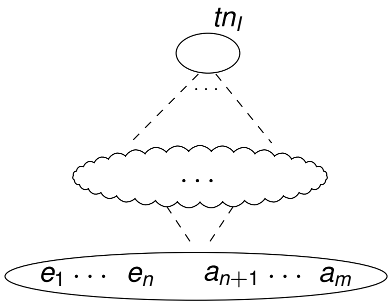
- Thus, in general, we need to take the already executed actions into account!
- More precisely, *when the execution of a plan fails, find a new plan that has the same prefix and can deal with the unexpected state transition*.
- For this, we require specialized *Plan Repair systems*, right?
- No! We can use almost exactly the same procedure as for *plan recognition*!
- Then, the observed actions (in the recognition setting) correspond the the actions already executed. What's missing? The unexpected state change.
- We add a novel action (and call it *process*) for which holds:
  - It is only executable once,
  - it will be executed exactly after the last executed action,
  - it produces exactly the unforeseen state changes.

Plan Repair as Planning



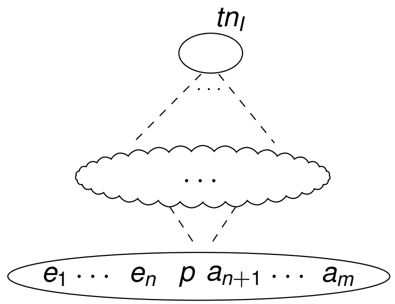
$a_1, \dots, a_m$ is the solution found.

Plan Repair as Planning



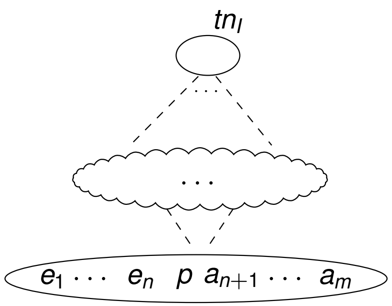
$e_1, \ldots, e_n = a_1, \ldots, a_n$ are the action already *executed* before the failure.

Plan Repair as Planning



p is the novel *process* action encoding the state space.

Plan Repair as Planning



However, to keep it simpler, we do not represent the process p with a novel action but instead add all its effects to the action a_n. Then, define $e_1, \ldots, e_n$ as the observed action from plan recognition and solve the plan recognition problem with *tn* as the single possible goal (task) network.

Conclusions

- Using this compilation technique allows us:
  - to use off-the-shelf hierarchical planners for plan repair, i.e., we do not need specialized systems for it and
  - to use existing standard heuristics without adapting them to the repair setting.

Introduction

What issues arise when conveying a plan to a user?

- How to convey the plan in general? Show the entire plan at once or convey the actions one by one?
- Only convey primitive actions (one at a time) or use abstract actions as well?
- If primitive actions should be conveyed,
  - How? I.e., how to create an adequate user interface from the action's formal description?
  - In which order to convey the actions? (→ Plan linearization, see next section)

Problem Description

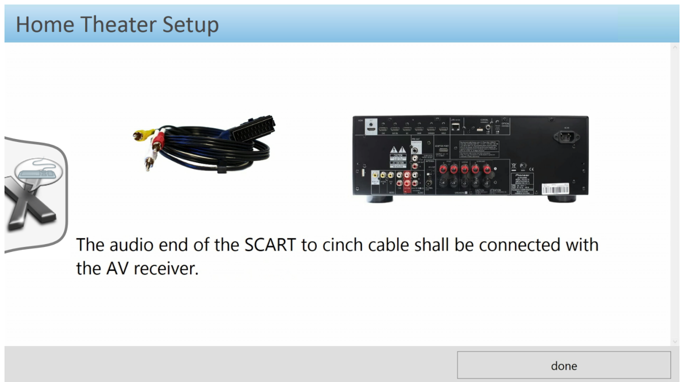
We now assume that the (original) input is given in a *lifted* fashion, e.g.:

describes a *ground instance* of a primitive action used to plugin a CINCH cable in an AV-Receiver to establish an audio signal.

So, how to convey it to a user?

Solution

- Use a template to generate natural language description, e.g. "Plug the *x* end of the *y* cable into the *z* device."
- Use pictures and/or videos to illustrate the involved objects., e.g.



Conveying Primitive vs. Abstract Tasks

- This approach works both for primitive *and abstract* tasks. However:
  - We could also use the primitive task's effects to incorporate them into the natural-language description.
  - There are extensions in which abstract tasks have effects as well, see next lecture.
  - When we want to convey abstract tasks, we need to re-infer such abstract tasks from the solution.

Conveying Abstract Tasks

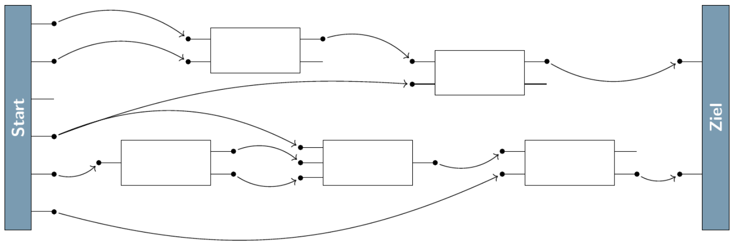
- Assume we want to convey plans via their abstract actions they rely on.
- Then we should convey them in a *reasonable order*.
- More precisely: If we want to convey, for example, an abstract task a_1 followed by an abstract task a_2, then the solution should consist of the (primitive) refinement of a_1 followed by the (primitive) refinement of a_2.
- Example: Does it make sense to use the initial grammar symbols of the grammar intersection problem to convey its solution?
- It is obviously decidable whether such a linearization of abstract tasks exists, because the decomposition tree is finite.

Conveying Primitive Tasks

- In general, (primitive) solutions are partially ordered.
- Note: Even when a solution is given totally ordered (e.g., due to using progression search), we can easily (i.e., in $\mathbb{P}$) find a partially ordered version of it.
- If a partially ordered solution is given, we need to answer the question in which order the actions should be conveyed to the user. → Plan Linearization
- Note: This question is *also* relevant in case we convey abstract tasks.

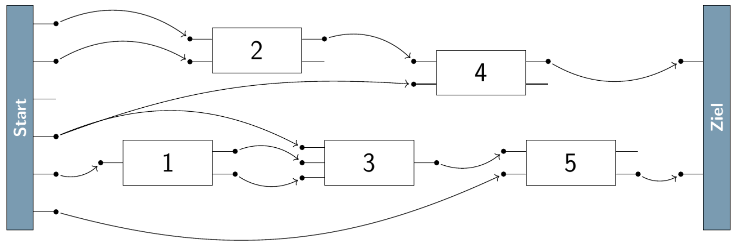
User-Friendly Plan Linearizations, Motivation

Which linearizations are well-suited for human users?



User-Friendly Plan Linearizations, Motivation

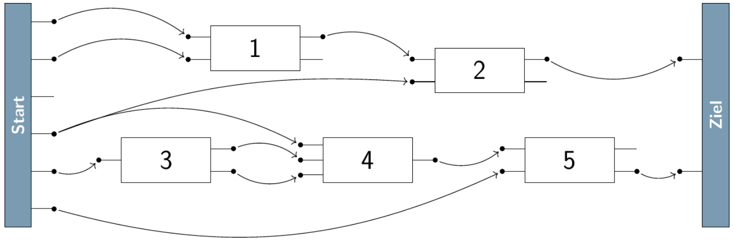
Which linearizations are well-suited for human users?



1: connect . . .
2: connect CINCH cable (one end) with Blu-ray player
3: connect . . .
4: connect CINCH cable (another end) with AV receiver
5: connect . . .

User-Friendly Plan Linearizations, Motivation

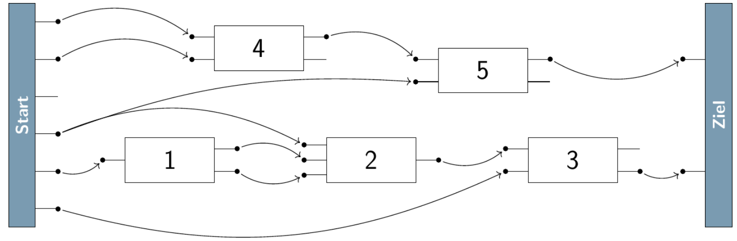
Which linearizations are well-suited for human users?



1: connect CINCH cable (one end) with Blu-ray player
2: connect CINCH cable (another end) with AV receiver
3: connect . . .
4: connect . . .
5: connect . . .

User-Friendly Plan Linearizations, Motivation

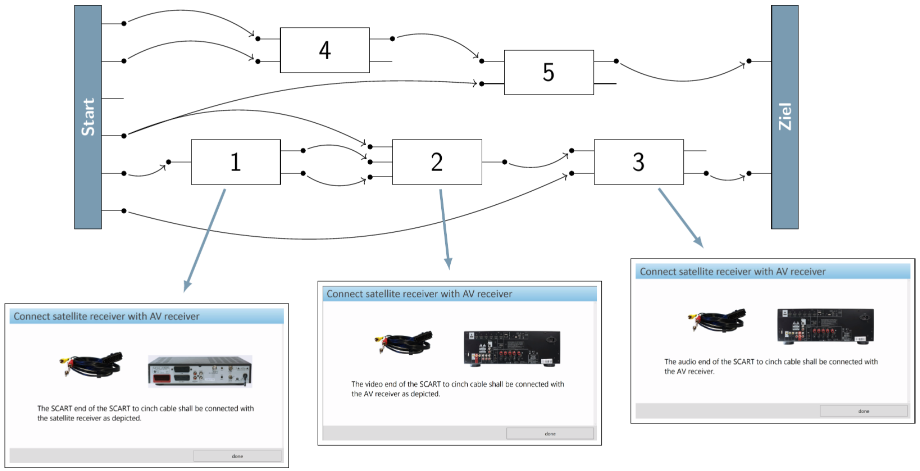
Which linearizations are well-suited for human users?



1: connect . . .
2: connect . . .
3: connect . . .
4: connect CINCH cable (one end) with Blu-ray player
5: connect CINCH cable (another end) with AV receiver

User-Friendly Plan Linearizations, Motivation

Which linearizations are well-suited for human users?



User-Friendly Linearization Strategies

Information used for finding user-friendly plan linearizations:

- The planning domain.
- The solution to the given planning problem

We show three linearization strategies, based on:

- Distance in the model's task hierarchy:
  Methods contain actions that "belong together".
- Number of identical constants:
  Perform actions that involve the same objects.
- Number of shared causal links:
  Perform actions that are causally related to each other.

Parameter-based Linearization Strategy

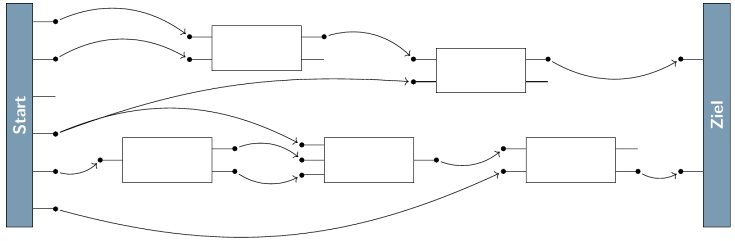
Reasoning behind using parameters for linearization:

- Actions represent activities to do.
- Parameters introduce the items/objects/subjects to use.

→ execute actions involving the same parameters consecutively.

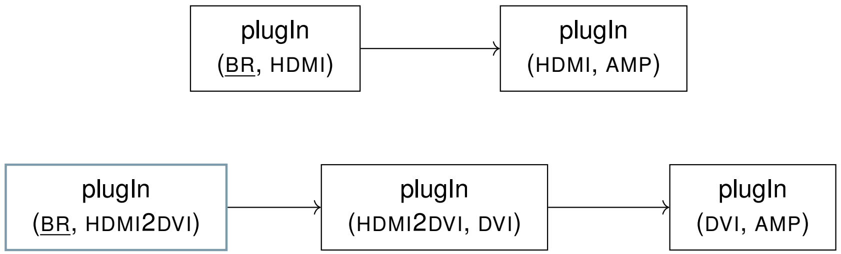
Parameter-based Linearization Strategy, Illustrating Example

Solution plan (schematically, with causal structure)



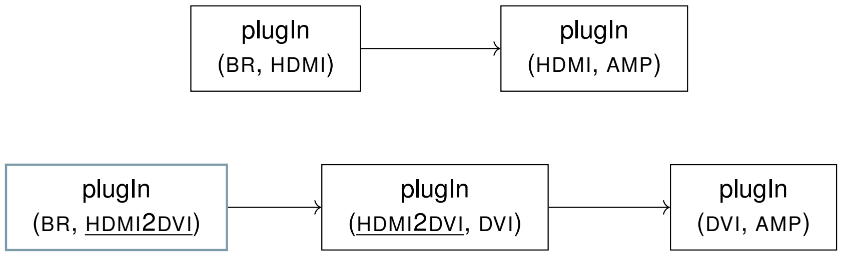
Parameter-based Linearization Strategy, Illustrating Example

Solution plan (ordering constraints, action schemata)



Parameter-based Linearization Strategy, Illustrating Example

Solution plan (ordering constraints, action schemata)

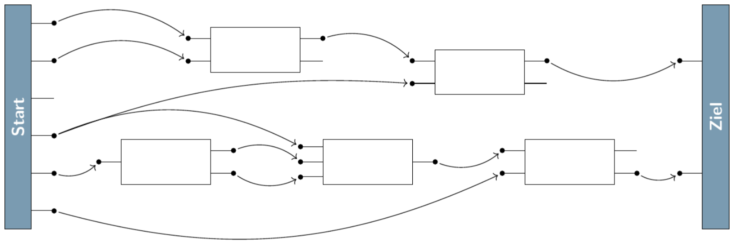


Causal Link-based Linearization Strategy

Reasoning behind using causal links for linearization:

- Causal links explicitly represent the causal dependencies between actions.
- Each link was introduced for a reason – all links are required.

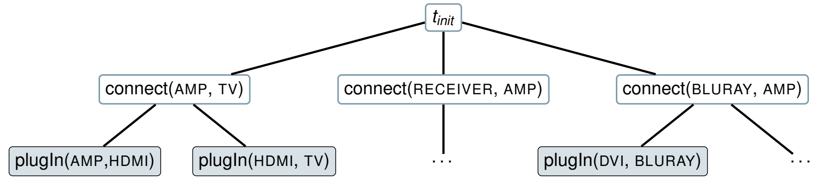
→ Execute connected actions consecutively.



Task Hierarchy-based Linearization Strategy

- Domain contains expert knowledge.
- Tasks that are introduced by the same method implement the same abstract task (→ they are semantically related).
- We generalize this relationship to tasks that are not in the same method (→ use the TDG).

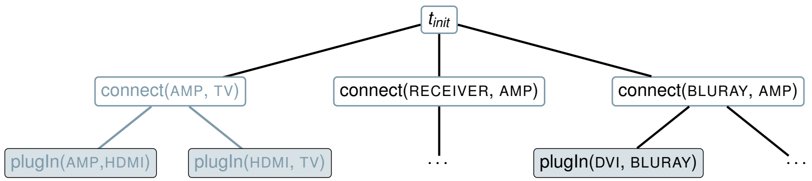
→ Execute actions consecutively that are close to each other in the TDG.



Task Hierarchy-based Linearization Strategy

- Domain contains expert knowledge.
- Tasks that are introduced by the same method implement the same abstract task (→ they are semantically related).
- We generalize this relationship to tasks that are not in the same method (→ use the TDG).

→ Execute actions consecutively that are close to each other in the TDG.

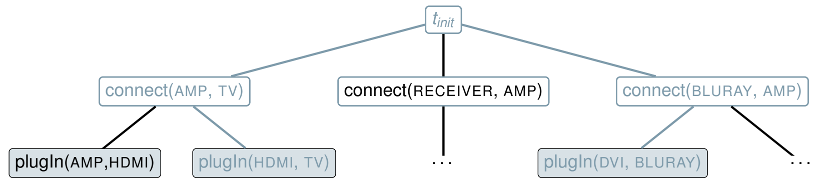


How closely to instruct plugIn(AMP,HDMI) and plugIn(HDMI, TV) next to each other?

Task Hierarchy-based Linearization Strategy

- Domain contains expert knowledge.
- Tasks that are introduced by the same method implement the same abstract task (→ they are semantically related).
- We generalize this relationship to tasks that are not in the same method (→ use the TDG).

→ Execute actions consecutively that are close to each other in the TDG.



How closely to instruct plugIn(HDMI, TV) and plugIn(DVI, BLURAY) next to each other?

Formal Descriptions of Linearization Criteria

Formal descriptions of these three optimization criteria can be found in the following paper:

Daniel Höller et al. "Finding User-friendly Linearizations of Partially Ordered Plans". In: *28th PuK Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PuK 2014)*. 2014

Introduction

- When plans get executed by human users, they might refuse to do so – or at least wonder whether there are different options.
- More precisely, a relevant example for a plan explanation question is given by:
  - Given a plan step X: "Why do I have to perform X?"
- Further possible questions are:
  - Why uses plan step $X(c_{i_1}, \ldots, c_{i_n})$ object/constant c_{i_k} as k-th argument rather than object/constant c'?
  - Why is plan step X ordered before plan step Y?

→ All these questions can be posed as *change requests*, e.g., "Can I also remove the ordering constraint between X and Y?"

→ In general change requests are as hard as planning (even though we already found a solution!).

→ Just asking for *some justification* why a certain property holds is much easier!

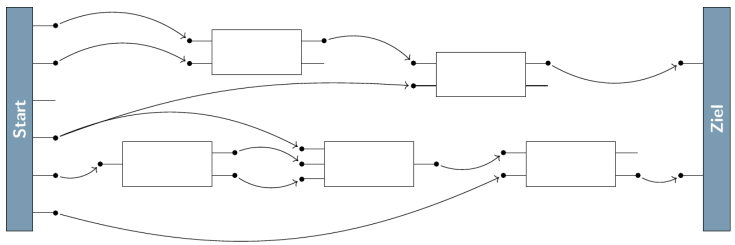
Plan Explanations – Overview

Both "explainable AI" and "explainable planning" became very prominent lately. Still, only a few approaches exist for planning:

- We focus on explaining properties of the given plan as mentioned before, in particular on questions addressing the necessity of actions.
- One approach considers *Plan Explanations as Model Reconciliation*. In a nutshell:
  - There is a *true* model of the real world and
  - another model that the user has about the world.
  - → The differences (i.e., wrong assumptions) are conveyed to the user. That way, his model can be altered as well. (See the RADAR video on `https://yochan-lab.github.io/robots/` (from 5:10))
- Another approach considers "excuses" for failed plans: Given an unsolvable planning problem, it finds alternative initial states that allow for a solution. The performed alterations to the actual state are referred to as excuses.

Explanations for Plan Step Necessity

Question: *Why should I perform action X?*



Possible answers:

- Exploit causality: X achieves effect x, which is necessary for action Y, which in turn achieves ...
- Exploit hierarchy: X is part of a (method) plan implementing action Y, which in turn implements ...

How to Compute Such Explanations?

Most canonical approach:

- Simply perform DFS/A* (with suitable heuristic) via:
  - Following the causal links to the goal state.
  - Following the DT upwards.
  - A combination for both (essential for TIHTN planning without goal state).

Another approach:

- Translation of the above-mentioned arguments to logics.

→ Despite being more complicated, this is the only approach published so far.

Plan Step Necessity Explanations via Logic – Step 1

We define various axioms:

- Following the causal links to the goal state:
  $CR(ps_1, ps_2) \wedge N(ps_2) \Rightarrow N(ps_1)$
- Following the DT upwards:
  $DR(ps_1, ps_2) \wedge N(ps_2) \Rightarrow N(ps_1)$
- Follow links to a goal state:
  $N(goal)$, where *goal* is an artificial goal action like in POCL planning.
- Follow task hierarchy until initial task network:
  $N(ps)$ for all plan steps *ps* in initial task network tn_I.
- What about *CR* and *DR*?
  - Causal relations (CR) are given for all causal links.
  - Decompositional relations (DR) are computed from the DT.

Plan Step Necessity Explanations via Logic – Step 2

Now, to answer the question *Why should I perform action X?* ...

- Collect all axioms (cf. previous slide) in a knowledge base KB.
- Ask for a proof of $KB \models N(X)$ given the current plan and its DT.

→ Its proof, a sequence of axiom applications, can be verbalized using proof verbalization techniques.

An example will be provided in the next section.

Introduction

We integrated these various user-centered planning capabilities

- plan generation,
- plan execution/monitoring/linearization,
- plan repair (though implemented differently), and
- plan explanation

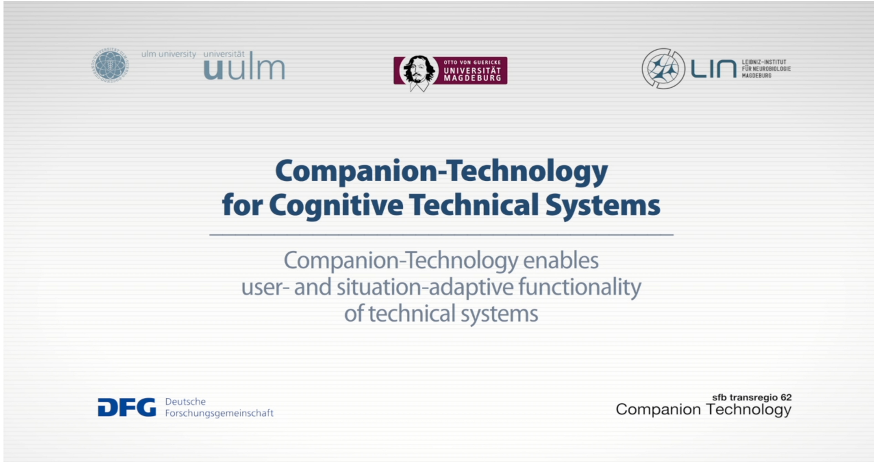
in a prototypical assistance system to assist in setting up a complex home theater.

Home Theater Assembly Assistant, Problem Setting

Four devices:

- Television (requires video)
- Blu-ray player
- Satellite receiver
- audio/video receiver (requires audio)

Home Theater Assembly Assistant, Complete Video



Video available at: `https://www.youtube.com/watch?v=Q25bGmFFc4U`

Overview

- Real-world applications require more (planning) capabilities than just the generation of plans. These comprise:
  - Plan execution/monitoring and (user-friendly) plan linearization.
  - Plan repair.
  - Plan explanation.
  - Plan recognition.
  - Allowing change requests.

→ Many of them (and more capabilities stemming from other computer science disciplines) were demonstrated in a prototypical assistance system helping in setting up a home theater.

- Many extensions to the underlying formalism would be beneficial as well, such as being able to deal with:
  - Time.
  - Resources.
  - Uncertainty.
  - And more (cf. first lecture)!

Discussed Techniques

- *Plan repair.*
  - Execution failures can be modeled as deviations from anticipated states.
  - In hierarchical planning, we have to take the executed actions into account as well!
  - Otherwise, when taking just the current state, we might get false witnesses.
  - We introduced an approach (similar to plan recognition), which reduces the plan repair problem to the plan existence problem.
- *Conveying plans.*
  - We showed how it can be done in a step-by-step (action-per-action) fashion.
  - We discussed issues when we want to convey abtract tasks as well.
  - For this step-by-step presentation, we need to commit to an ordering (→ plan linearization).

Discussed Techniques, cont'd

- *User-friendly plan linearization.*
  - We showed that different plan linearizations, though *all* being correct, might be more or less intuitive or useful.
  - We sketched three techniques to obtain *user-friendly* (i.e., intuitive) linearizations which take into account: task parameters, causal links, or the task hierarchy.
- *Plan explanation.*
  - We showed how to derive explanations stating the necessity for a plan step in a solution.
  - This technique can be implemented as simple search or via compilations, e.g., to predicate logics.
  - In any case, explanations essentially encode chains of causal links or of hierarchical decompositions.