Chapter MK:VI

VI. Planning and Configuration

- Agent Systems
- Deductive Reasoning Agents
- Planning Language

- Planning Algorithms
- State-Space Planning
- Plan Space Planning
- HTN Planning
- Complexity of Planning Problems
- Erweiterungen

- Konfigurierungsproblemstellung
- Konfigurierungsansätze

Remarks:

- Helpful books on agent systems and planning:
  - Malik Ghallab, Dana Nau, Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.
    (Lecture Slides)
  - Malik Ghallab, Dana Nau, Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.
    (Lecture Slides)
  - David Poole, Alam Mackworth, Randy Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, 1998.
    (Lecture Slides)
  - Stuart Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.
    (Lecture Slides)
  - Nils Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing, 1980
  - Jacques Ferber. *Multi-Agent Systems: An Introduction to Artificial Intelligence*. Addison-Wesley, 1999.
  - Michael R. Genesereth, Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Margan Kaufmann, 1987.

Agent Systems

Example: Monkey-and-Banana-Problem

A monkey is in a room. Suspended from the ceiling is a bunch of bananas, beyond the monkey's reach. In the corner of the room is a box. How can the monkey get the bananas?

The solution is of course that the monkey must push the box under the bananas, then stand on the box and grasp the bananas.

[Wikipedia]

Agent Systems

Example: Monkey-and-Banana-Problem

A monkey is in a room. Suspended from the ceiling is a bunch of bananas, beyond the monkey's reach. In the corner of the room is a box. How can the monkey get the bananas?

The solution is of course that the monkey must push the box under the bananas, then stand on the box and grasp the bananas.

[Wikipedia]

➜ What is the problem?

Agent Systems

Intelligent Agents

An *agent* is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

A *rational agent* is one that does the right thing [. . .].

[Russell & Norvig: Artificial Intelligence 2nd ed., 2003]

PEAS characterization of agents:

- Performance Measures
  What are the criteria for success and how are they measured?
- Environment
  What are the circumstances, objects, or conditions by which an agent is surrounded?
- Actuators
  What are prerequisites and consequences of using actuators?
- Sensors
  What sensors are available and what information do they provide?

Remarks:

- In [Russell & Norvig: Artificial Intelligence 2nd ed., 2003] also a more detailed definition of rational behavior is given:

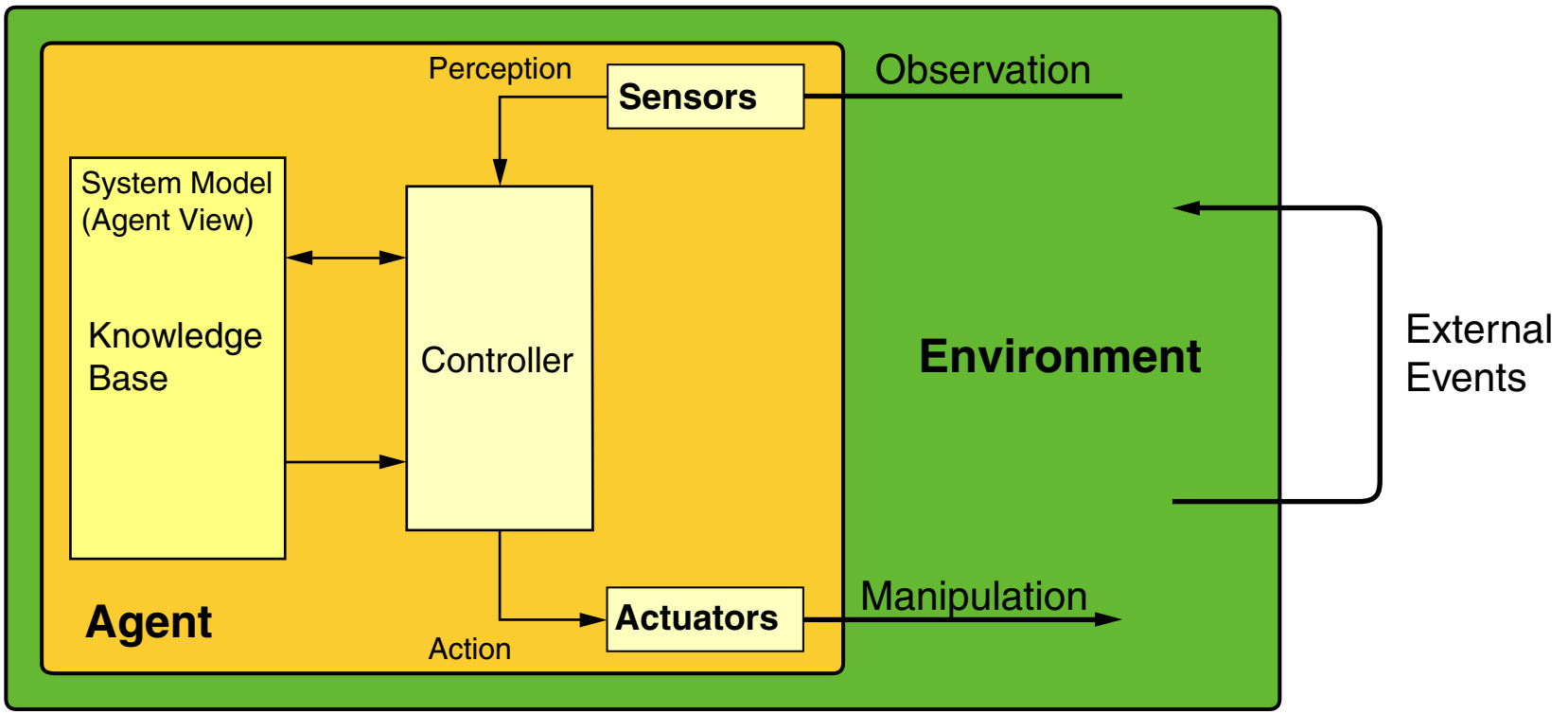
  > For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built in knowledge the agent has.

- In [Russell & Norvig: Artificial Intelligence, 1995] the PAGE characterization of agents is given that differs from the PEAS description in that agents have goals to achieve:
  - Percepts
    What sensors are available and what information do they provide?
  - Actions
    What are prerequisites and consequences of using actuators?
  - Goals
    What does an agent try to achieve?
  - Environment
    What are the circumstances, objects, or conditions by which an agent is surrounded?

  In the second edition of their book Russell and Norvig take a more pragmatic view. Success is given by a performance measure and an agent simply tries to maximize its performance. The problem is now to describe something like a *"better world"* by a performance measure.

Agent Systems

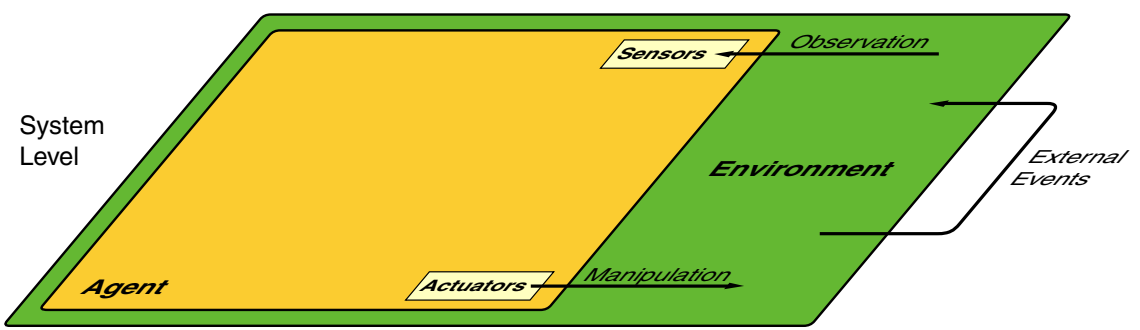
Modeling in Agent Design: Classical View [Russell & Norvig]



- The agent can be part of the environment.
  (Its position, pose, . . . are part of the "state" of the environment.)
- The "inside" of an agent (knowledge or intentions) are NOT part of the environment.

Agent Systems

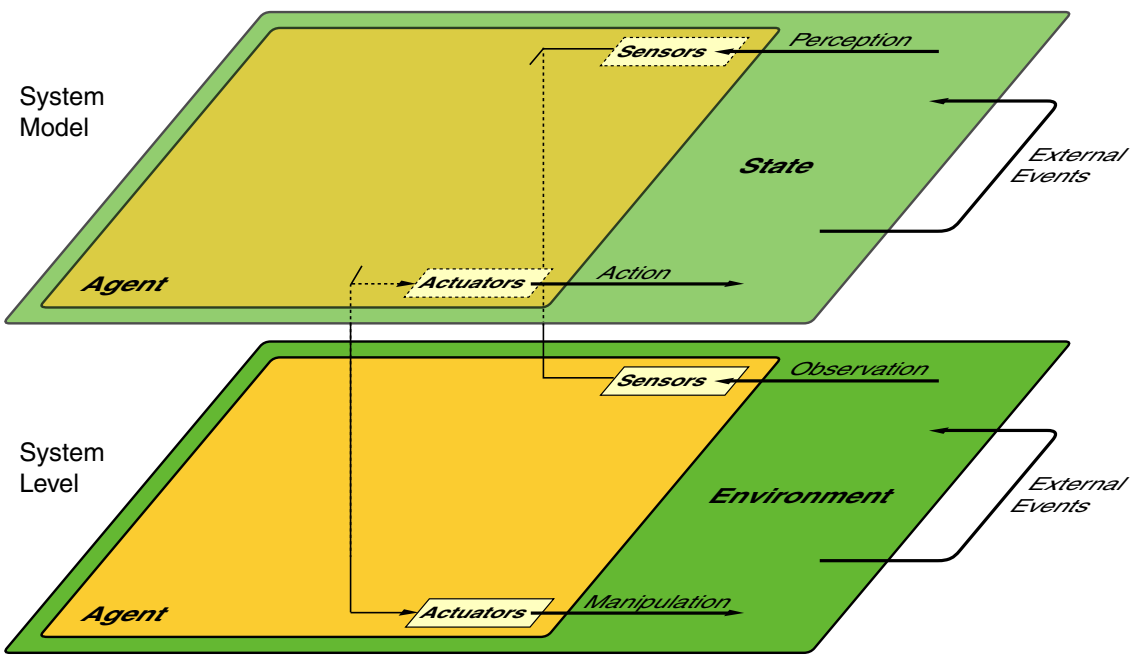
Modeling in Agent Design: Model Levels



System Level: What is the interaction of agent and environment in reality from the observer's point of view?

Agent Systems

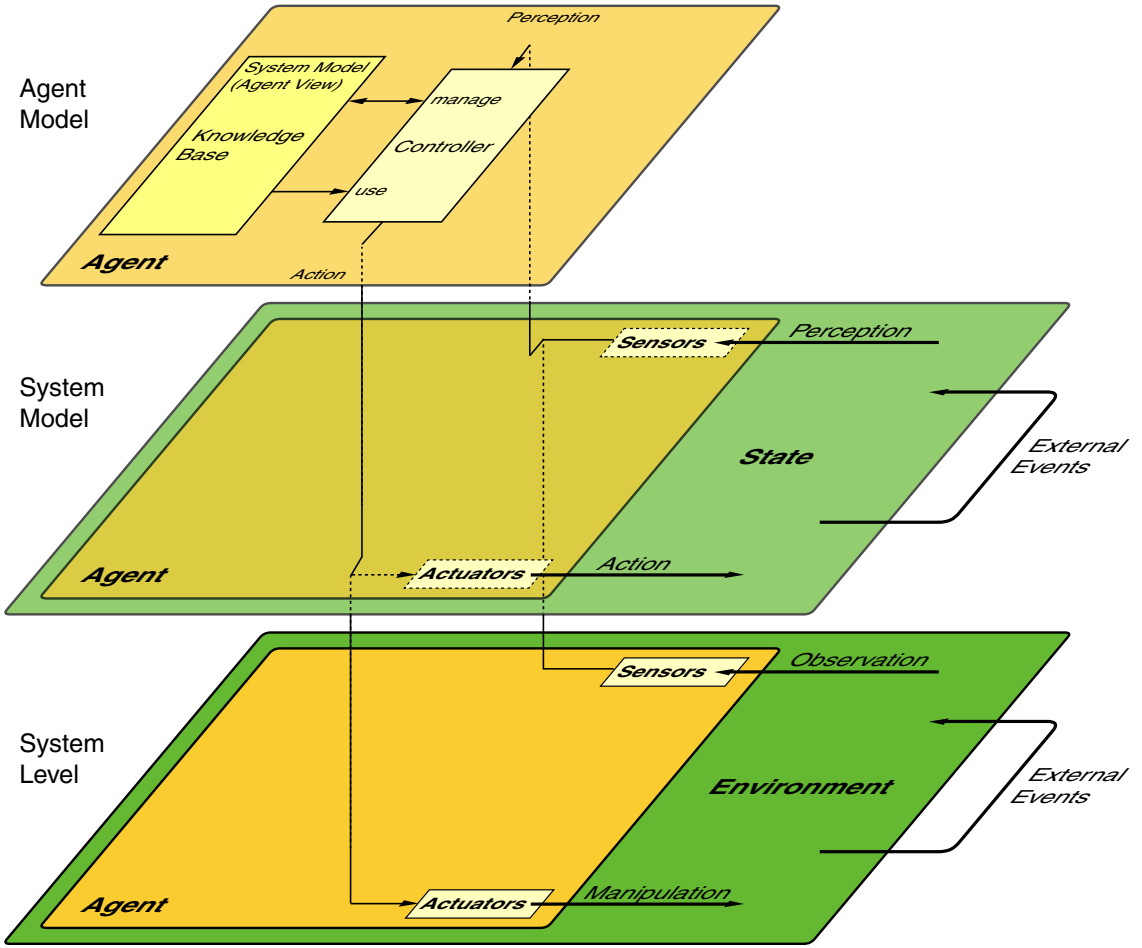
Modeling in Agent Design: Model Levels



System Model: How is the interaction of agent and environment in principle from the observer's point of view?

Agent Systems

Modeling in Agent Design: Model Levels



Agent Model: What is the interaction between agent and environment in principle from the agent's point of view?

Agent Systems

Agents as Part of the System Model: Observer's View

Definition 1 (State Transition System with Events STS-E)

A state transition system with events $\mathcal{S} = (S, A, E, T)$ consists of

- $S = \{s, s', ...\}$ countable set of states (of the environment),
- $A = \{a, a', ...\}$ countable set of actions of the agent.
- $E = \{e, e', ...\}$ countable set of external events.
- $T \subseteq (S \times A \times E) \times S$ defines possible transitions of the system. $(s, a, e, s') \in T$ denotes that a is applicable in state s, e is an external event that can happen in s, and s' is a possible successor state.

Environment Model $\mathcal{S} = (S, A, E, T)$

Agent Model $\mathcal{A} = (\mathcal{S}, A, \pi)$

- $\pi : S \to A$ denotes the policy of an agent.

Remarks:

- S may contain a specific initial state s_0 and S may contain a set of specific final states F.
- The above model of an environment is a gross simplification of stochastic processes. In general, applicability of an action and its successor state may depend on previous states and actions, i.e. the Markov property does not hold in such systems. An extension to the above model (assuming $E = \emptyset$) is to consider runs $r = (s_0, a_1, s_1, a_2, s_2, a_3, s_3 \ldots, s_{k-1}, a_k, s_k)$ that are finite alternating sequences of states and actions. Let R be the set of such sequences. Then $T \subseteq R \times A \times S$ defines possible transitions of the system. $(r, a, s') \in T$ denotes that a is applicable in the final state reached by run r and s' is a successor state. In such a case, the agent has a policy $\pi : R \to A$.
- Actions don't take time and result in an instantaneous state transition.
- Also, the above model of an environment includes only a simple representation of external events. External events are only considered together with actions of the agent. An environment without external events is modeled by a state transition system.
- An embodied agent will be part of the environment, it has a position and a pose. The mental state of an agent, however, is not part of the environmental state. It is assumed to be non-observable.
- Performance measures can be based on runs of an agent.

Remarks: (continued)

- The simple formal model suggests that we want to consider only one simple form of environments, namely environments that form a state transition system with events. A different perspective is more helpful:

  An agent uses his sensors to detect his environment. The sensors are inaccurate compared to the real environment, so that the actual state cannot be completely observed by the agent. The coarsened view is further simplified within the agent to make it more manageable. Within the agent, one works with an abstraction of the environment in the form of a state transition system.

  The simple formal model describes the agent-internal conception of the functioning of the environment. The planning task is formulated in terms of the formal model that an agent has of his environment. It can be seen as an agent internal translation of a planning task placed on the real system of agent and environment.

Agent Systems

Agent Capabilities: Designer's View

- Agents have and/or build up knowledge about the environment.
  (Learning, Exploration/Exploitation, ...)
  Simple case: $\mathcal{S} = (S, A, E, T)$ is known.
- Agents have additional knowledge (objects, relations, methods, causality,...).
  (Background knowledge, e.g., "carrying objects changes their positions", history of previous states actions, and events,...)
  Simple case: Apart from $\mathcal{S}$ nothing is known.
- Agents perform the following steps in a loop:
  1. Update the current state of the environment based on sensor readings and previous knowledge.
  2. Determine some promising action to perform.
  3. Execute that action and update knowledge.

  Simple case: $s \in S$ can be observed directly.

➜ Logical agents use logical languages to represent knowledge.

Agent Systems

Agent Capabilities: Designer's View (continued)

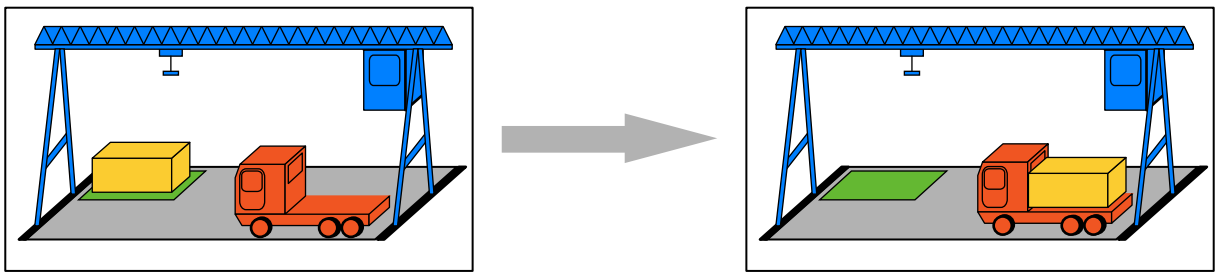
- Agents implement rational behavior by looking ahead: planning.

> Automated planning and scheduling, sometimes denoted as simply AI Planning, is a branch of artificial intelligence that concerns the realization of strategies or action sequences, typically for execution by intelligent agents, autonomous robots and unmanned vehicles. [...]
>
> In known environments with available models, planning can be done offline. Solutions can be found and evaluated prior to execution. In dynamically unknown environments, the strategy often needs to be revised online. Models and policies must be adapted.
>
> [Wikipedia, 2020]

➜ Deductive reasoning agents use logical reasoning to infer a promising action.

Agent Systems

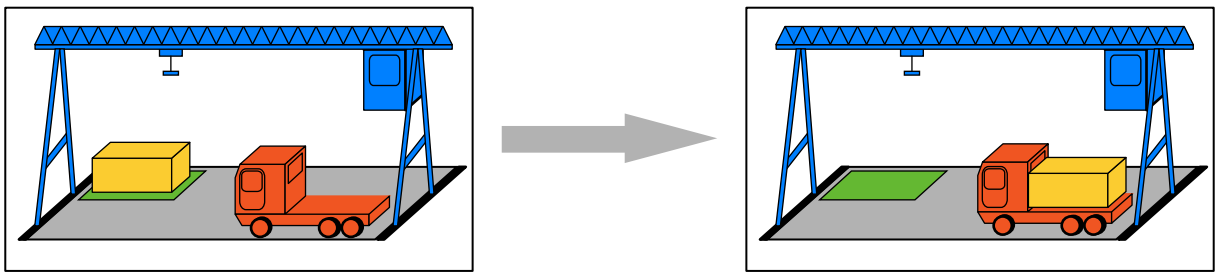
Example: Loading Dock [Ghallab, Nau, Traverso]



➜ How to load the truck?

Agent Systems

Example: Loading Dock [Ghallab, Nau, Traverso]



➜ How to load the truck?

Abstraction from concrete problems:

- simplification of possible locations for cargo, truck, crane,. . . into countable/finite number of configurations,
- simplification of the causes of state changes into finite sets of actions and events.

➜ Problem solving by searching for a suitable sequence of actions.

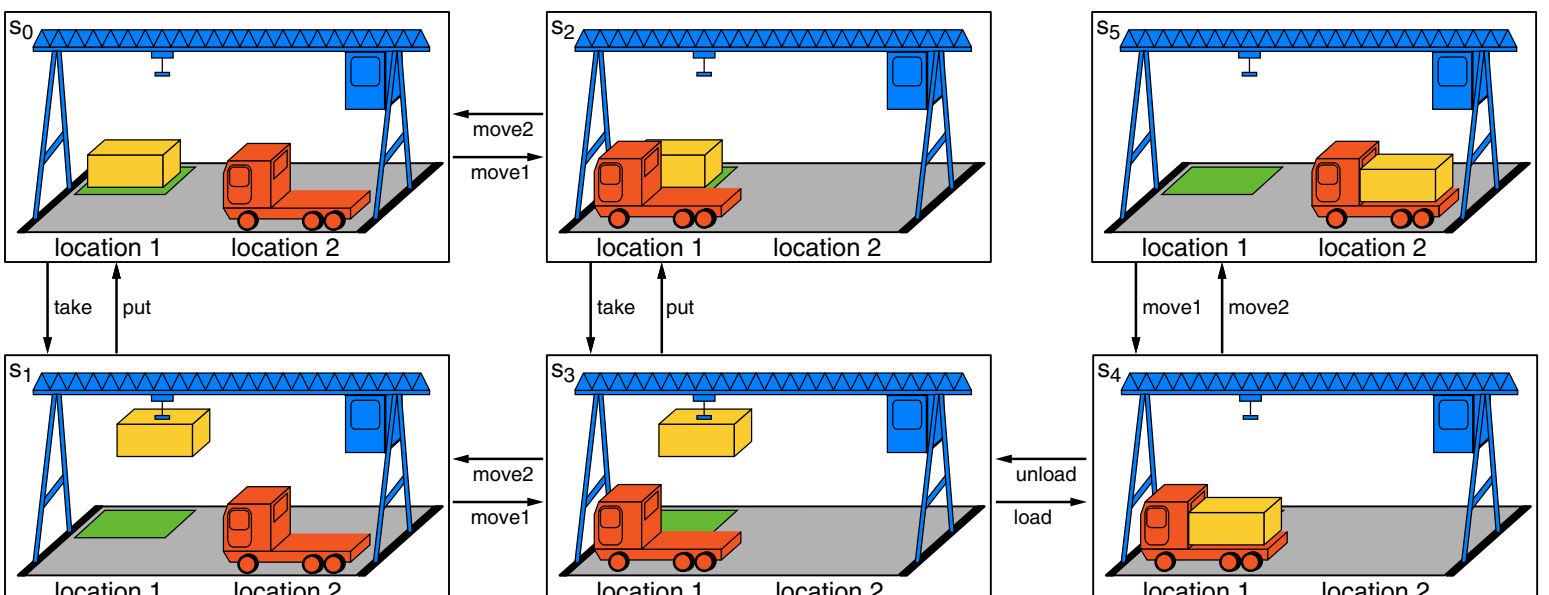
A more complex problem of this type is the Container Pre-Marshalling Problem.

Agent Systems

Example: Loading Dock (continued)

Formal model of the environment $\mathcal{S}$

- states $S = \{s_0, \ldots, s_5\}$
- actions $A = \{\mathit{move1}, \mathit{move2}, \mathit{put}, \mathit{take}, \mathit{load}, \mathit{unload}\}$
- events $E = \emptyset$
- state transitions



Definition 2 (Planning Problem, Plan)

Let $\mathcal{S} = (S, A, E, T)$ be the environment model for an agent.

- A planning problem (s_0, F) is defined by some initial state $s_0 \in S$ and some set of final states $F \subseteq S$
- A plan $(a_1, \ldots, a_k)$ is a finite sequence of actions in A.
- A plan $(a_1, \ldots, a_k)$ is a solution for the planning problem (s_0, F) if there are states $s_1, \ldots, s_k \in S$ and events $e_1, \ldots, e_k \in E$ such that $(s_{i-1}, a_i, e_i, s_i) \in T$ and $s_k \in F$.

➜ According to the above definition a plan is a "best case" solution.

- If $E = \emptyset$ and if T is deterministic, it is enough to have a plan.
- If possible events are known (e.g. two player game), a strategy is needed.
- If possible events are not known, a policy is needed.

Remarks:

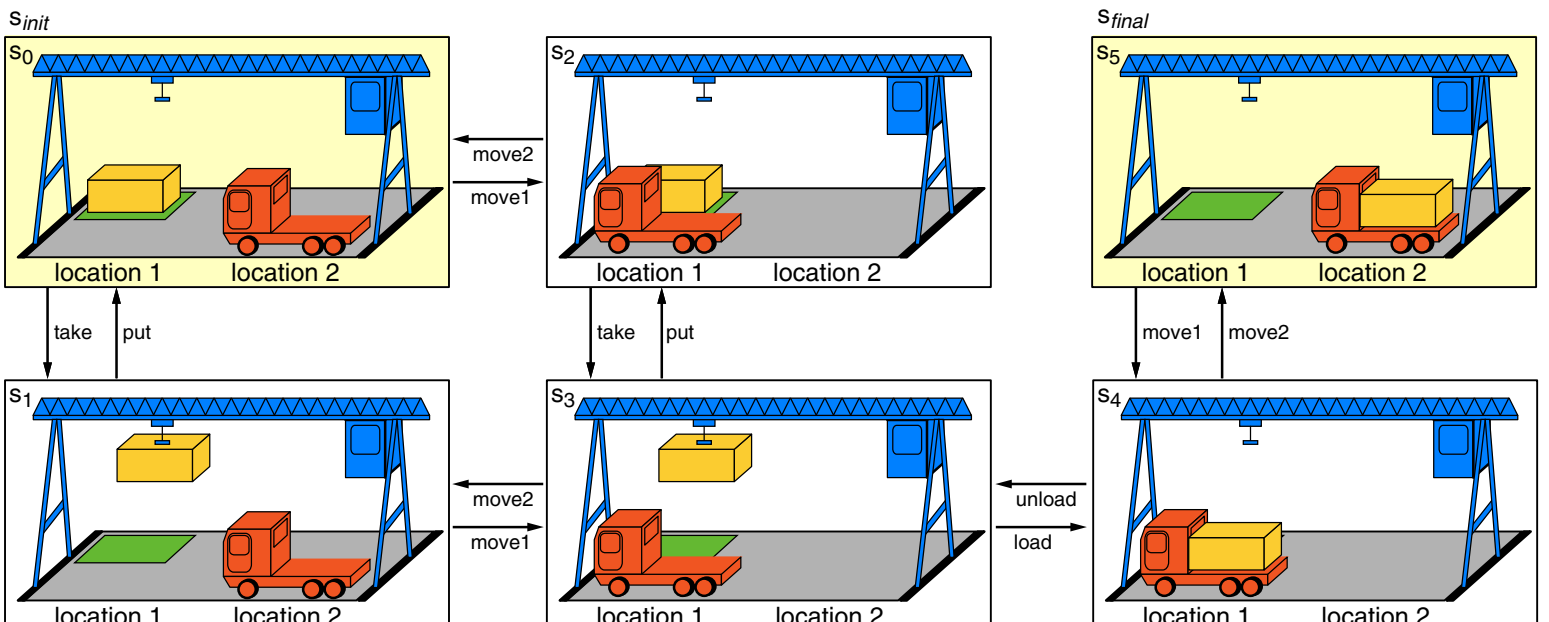
- More generally, the objective in planning task could be
  - to reach a specific goal state,
  - to reach a set of goal states in a fixed order one after the other,
  - to perform a list of tasks,
  - to satisfy some condition over the sequence of states generated by a plan,
  - to maximize some utility function,
  - ...
- The task is often limited for complexity studies:
  Is there a plan (of length k) with the required properties?

Agent Systems

Example: Loading Dock (continued)

Environment model STS-E $\mathcal{S} = (S, A, E, T)$

- planning problem

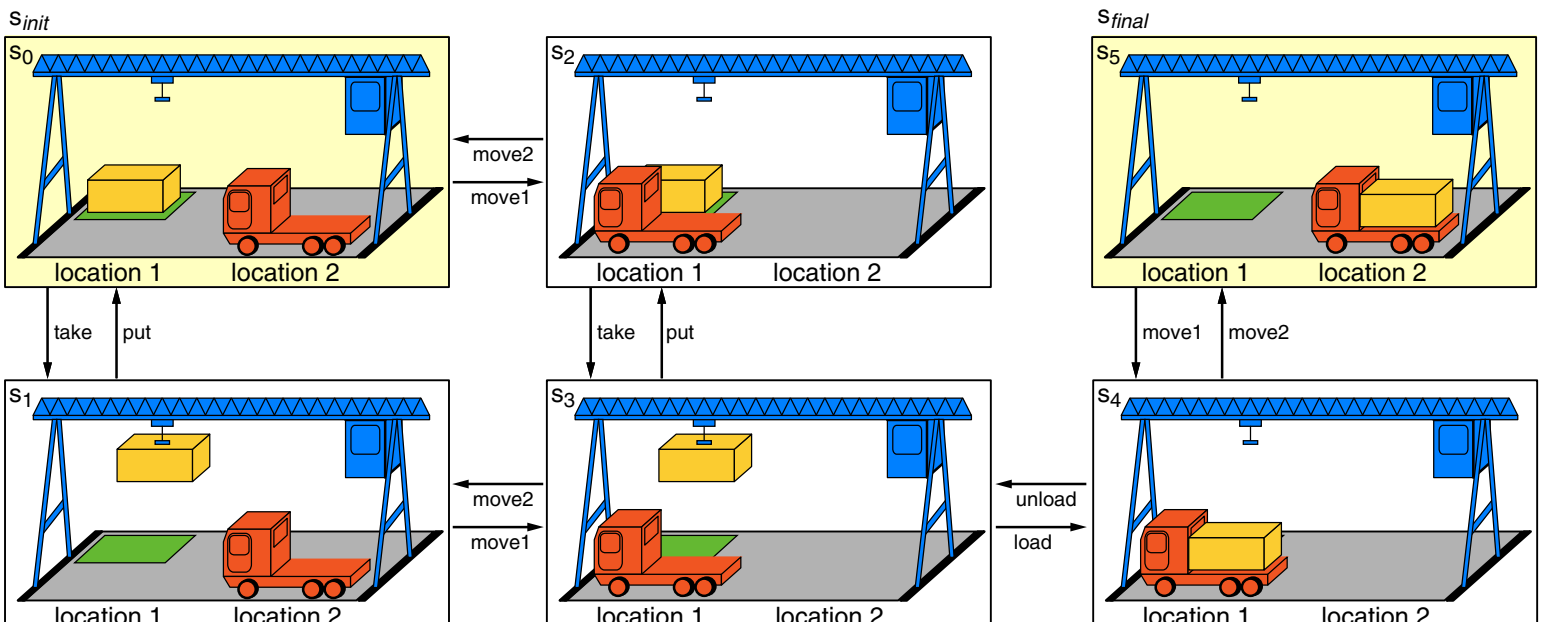


Agent Systems

Example: Loading Dock (continued)

Environment model STS-E $\mathcal{S} = (S, A, E, T)$

- planning problem



- plan $p = (\textit{take}, \textit{move1}, \textit{load}, \textit{move2})$
- policy $\pi = \{(s_0, \textit{take}), (s_1, \textit{move1}), (s_3, \textit{load}), (s_4, \textit{move2})\}$

Q. What is the difference between plans and policies?

Agent Systems

Possible Restrictions of Planning Problems

A0 Finite System
The set of states S is finite.

A1 Full Observability
The state of the system can be fully observed. There is no uncertainty about the current state.

A2 Deterministic System
For each action-event pair there is at most one successor state in each state:
$T : S \times A \times E \to S$, T is a (partial) function.

A3 Static System
There are no exogenous events, state changes only occur due to agent's actions: $E = \emptyset$.

A4 Restricted Goals
A set of possible goal states is predefined.

A5 Sequential Plans
Plans are linearly ordered sequences of actions.

A6 Implicit Time
Actions and events have no duration. State transitions are instantaneous.

A7 Offline Planning
The planner does not take into account any state changes during the planning process that may occur because of system dynamics.

➜ Given restrictions A0-A7: Classical Planning

Remarks:

- If according to A0 the set of states is finite, we can also assume that the set of actions and the set of external events are finite. In case of infinite sets of events or actions we can consider equivalence classes with respect to the transition relation instead. These number of equivalence classes is finite for a finite set of states.

Agent Systems

Possible Restrictions of Planning Problems

A0 Finite System
The set of states S is finite.

A1 Full Observability
The state of the system can be fully observed. There is no uncertainty about the current state.

A2 Deterministic System
For each action-event pair there is at most one successor state in each state:
$T : S \times A \times E \to S$, T is a (partial) function.

A3 Static System
There are no exogenous events, state changes only occur due to agent's actions: $E = \emptyset$.

A4 Restricted Goals
A set of possible goal states is predefined.

A5 Sequential Plans
Plans are linearly ordered sequences of actions.

A6 Implicit Time
Actions and events have no duration. State transitions are instantaneous.

A7 Offline Planning
The planner does not take into account any state changes during the planning process that may occur because of system dynamics.

➜ Given restrictions A1-A7: Planning can be done by state-space search.

Chapter MK:VI

VI. Planning and Configuration

- Agent Systems
- Deductive Reasoning Agents
- Planning Language

- Planning Algorithms
- State-Space Planning
- Plan Space Planning
- HTN Planning
- Complexity of Planning Problems
- Erweiterungen

- Konfigurierungsproblemstellung
- Konfigurierungsansätze

Deductive Reasoning Agents

Tasks in Planning

- Description of the Initial Situation
  What are the characteristics of the initial state, both positive (valid) and negative (non-valid)?
- Description of Actions
  What are the prerequisites for the execution of an action, what causes the execution?
- Projection Problem
  What are the characteristics of the achieved state for an action sequence with a given initial state?
- Planning Problem
  What is a suitable sequence of actions to achieve certain characteristics in a target state with a given initial state?

> Tell it what it needs to know. Then it can ask itself what to do.
>
> [Russell/Norvig]

Deductive Reasoning Agents

Agent Capabilities: Designers View (continued)

Sources of Intelligent Behavior: Concurrent AI Agent Paradigms

- The Society of the Mind (Marvin Minsky, started in early 1970s)

  "Competence emerges from a large number of relatively simple agents integrated by a clever architecture."

- Common Sense Knowledge (Douglas Lenat, started in 1980)

  "Competence arises from a large body of common sense knowledge."

- Physical Grounding Hypothesis (Rodney Brooks, 1990)

  "To build an intelligent system it is necessary to base its representation on the physical world. The real world is its own best model provided a robot can sense it appropriately and often enough."

Remarks:

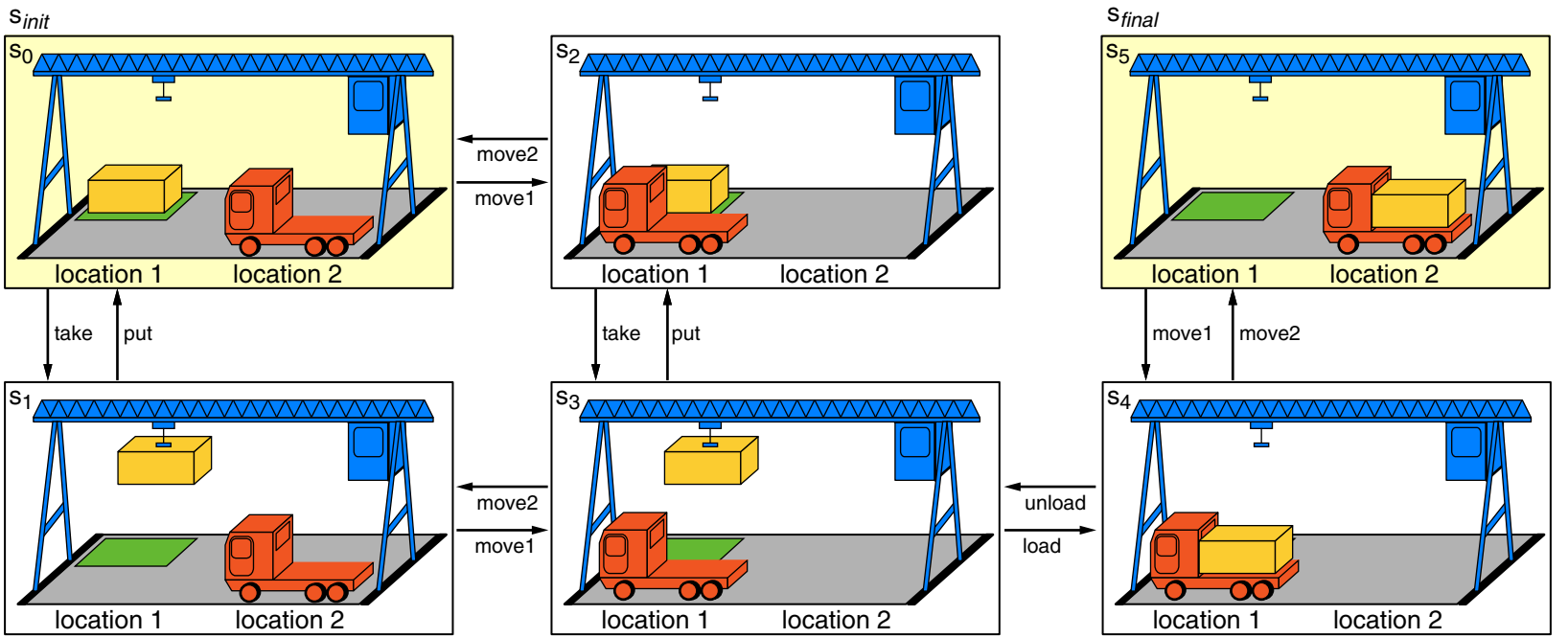
- Promoting intelligent behavior yields an inherent problem stated by

  Cole's Axiom: The sum of the intelligence on the planet is a constant; the population is growing.

Deductive Reasoning Agents

Example: Loading Dock (continued)

States $S = \{s_0, \ldots, s_5\}$



Q. How do you describe *large* state spaces?

Deductive Reasoning Agents

Describing States

Approach:

- description by specifying properties.

Objective:

- unique description for each state,
- simple semantics,
- independence from the specific application.

Deductive Reasoning Agents

Describing States

Approach:

- description by specifying properties.

Objective:

- unique description for each state,
- simple semantics,
- independence from the specific application.

➜ Using logic-based descriptions:

- first order logic $in(container1, stack1)$
- propositional logic *container1_in_stack1*

Deductive Reasoning Agents

Describing States

Approach:

- description by specifying properties.

Objective:

- unique description for each state,
- simple semantics,
- independence from the specific application.

➜ Using logic-based descriptions:

- first order logic $in(container1, stack1)$
- propositional logic *container1_in_stack1*

Q. How to describe *large* state spaces?

Deductive Reasoning Agents

Describing State Spaces

Approach:

- explicit specification of initial states,
- specification of operators as generic state transitions.

Objective:

- generic specification as operator, actions as instantiations,
- specification of conditions for application of actions,
- specification of state changes by properties changed.

➜ If required, parts of the state space can be generated by using operators, starting from the initial state.

Deductive Reasoning Agents

Example: Monkey-and-Banana-Problem (continued)

A monkey is in a room. Suspended from the ceiling is a bunch of bananas, beyond the monkey's reach. In the corner of the room is a box. How can the monkey get the bananas?

The solution is of course that the monkey must push the box under the bananas, then stand on the box and grasp the bananas.

[Wikipedia]

➜ How to find a plan for grasping the bananas?

Deductive Reasoning Agents

Example: Monkey-and-Banana-Problem (continued)

A monkey is in a room. Suspended from the ceiling is a bunch of bananas, beyond the monkey's reach. In the corner of the room is a box. How can the monkey get the bananas?

The solution is of course that the monkey must push the box under the bananas, then stand on the box and grasp the bananas.

[Wikipedia]

First solution: Prolog (First Order Logic with Resolution [SWI-Prolog Online])

$\mathit{singleMove}(\mathit{state}(X, \mathit{onbox}, X, X, \mathit{hasnot}), \mathit{grasp}, \mathit{state}(X, \mathit{onbox}, X, X, \mathit{has})).$
$\mathit{singleMove}(\mathit{state}(X, \mathit{onfloor}, X, Y, Z), \mathit{climb}, \mathit{state}(X, \mathit{onbox}, X, Y, Z)).$
$\mathit{singleMove}(\mathit{state}(X1, \mathit{onfloor}, X1, Y, Z), \mathit{push}(X1, X2), \mathit{state}(X2, \mathit{onfloor}, X2, Y, Z)).$
$\mathit{singleMove}(\mathit{state}(X1, \mathit{onfloor}, Y, Z, H), \mathit{walk}(X1, X2), \mathit{state}(X2, \mathit{onfloor}, Y, Z, H)).$

$\mathit{moves}(\mathit{state}(\_, \_, \_, \_, \mathit{has}), [\,]).$
$\mathit{moves}(S1, [A|AL]) :- \mathit{singleMove}(S1, A, S2), \mathit{moves}(S2, AL).$

$\mathit{solve}(AL) :- \mathit{moves}(\mathit{state}(\mathit{atdoor}, \mathit{onfloor}, \mathit{atwindow}, \mathit{middle}, \mathit{hasnot}), AL).$

$?- \mathit{solve}(ATALL).$

Deductive Reasoning Agents

Example: Monkey-and-Banana-Problem (continued)

A monkey is in a room. Suspended from the ceiling is a bunch of bananas, beyond the monkey's reach. In the corner of the room is a box. How can the monkey get the bananas?

The solution is of course that the monkey must push the box under the bananas, then stand on the box and grasp the bananas.

[Wikipedia]

Second solution:

$\mathit{state}(a, b, c, d).$
$\mathit{state}(X2, Y, Z, \mathit{walk}(X1, X2, S)) :- \mathit{state}(X1, Y, Z, S).$
$\mathit{state}(X2, Y, X2, \mathit{push}(X1, X2, S)) :- \mathit{state}(X1, Y, X1, S).$
$\mathit{state}(X, Y, X, \mathit{climb}(S)) :- \mathit{state}(X, Y, X, S).$

$\mathit{solve}(\mathit{grasp}(\mathit{climb}(S))) :- \mathit{state}(X, X, X, \mathit{climb}(S)).$

$?- \mathit{solve}(ATALL).$

Deductive Reasoning Agents

Modeling Knowledge in Logic

- Modeling of environmental states

  term-based vs. fact-based

  singleMove(*state*(X, *onbox*, X, X, *hasnot*), *grasp*, *state*(X, *onbox*, X, X, *has*)).
  state($X2$, Y, Z, *walk*($X1$, $X2$, S)) :– *state*($X1$, Y, Z, S).

- Modeling of state transitions for action sequences

  implicit dependencies vs. explicit dependencies

  singleMove(*state*(X, *onbox*, X, X, *hasnot*), *grasp*, *state*(X, *onbox*, X, X, *has*)).
  state($X2$, Y, Z, *walk*($X1$, $X2$, S)) :– *state*($X1$, Y, Z, S).

- Model accuracy

  explicit representation of object relations vs. implicit representation

  singleMove(*state*(X, *onbox*, X, X, *hasnot*), *grasp*, *state*(X, *onbox*, X, X, *has*)).
  state(X, Y, X, *climb*(S)) :– *state*(X, Y, X, S).

- Importance of the clause order in Prolog programs

  Program version 1 does not work correctly if clause 4 is on top.
  There is no correct clause ordering for program version 2.

Deductive Reasoning Agents

Example: Monkey-and-Banana-Problem (continued)

Extended second solution:

$$\textit{singleMove}(\textit{state}(X1, \textit{onfloor}, Y, Z, H), \textit{walk}(X1, X2), \textit{state}(X2, \textit{onfloor}, Y, Z, H)).$$
$$\textit{singleMove}(\textit{state}(X1, \textit{onfloor}, X1, Y, Z), \textit{push}(X1, X2), \textit{state}(X2, \textit{onfloor}, X2, Y, Z)).$$
$$\textit{singleMove}(\textit{state}(X, \textit{onfloor}, X, Y, Z), \textit{climb}, \textit{state}(X, \textit{onbox}, X, Y, Z)).$$
$$\textit{singleMove}(\textit{state}(X, \textit{onbox}, X, X, \textit{hasnot}), \textit{grasp}, \textit{state}(X, \textit{onbox}, X, X, \textit{has})).$$

$$\textit{moves}(\textit{state}(\_, \_, \_, \_, \textit{has}), [\,], 0).$$
$$\textit{moves}(S1, [A|AL], N) :- N1 \textit{ is } N - 1, N1 >= 0, \textit{singleMove}(S1, A, S2), \textit{moves}(S2, AL, N1).$$

$$\textit{planlength}(0).$$
$$\textit{planlength}(N) :- \textit{planlength}(N1), N \textit{ is } N1 + 1.$$

$$\textit{solve}(AL) :- \textit{planlength}(N), \textit{moves}(\textit{state}(\textit{atdoor}, \textit{onfloor}, \textit{atwindow}, \textit{middle}, \textit{hasnot}), AL, N).$$

$$?- \textit{solve}(ATALL).$$

Program implements a DFS-search with iterative deepening.

Remarks:

- Version 1 explicitly models the monkey's position in relation to the box (*onfloor*, *onbox*) and the monkey's position in relation to the banana (*has*, *hasnot*).
- Version 2 models the states less in detail, but depending on the sequence of actions performed.
- Version 1 works correctly only with the specified ordering of clauses. Changing the order of clauses 3 and 4 leads to an infinite loop.
- Version 2 does not work correctly in any order of the clauses, there is always an infinite loop.
- Version 1 searches for a sequence of actions starting with the initial situation in clause 5.
- Version 2 searches for a sequence of actions starting with the target in clause 1.
- Version 3 extends version 1: A depth limit is integrated into the depth search of Prolog, the depth is increased stepwise by clauses 6 and 7.
- Version 3 represents states as terms, because with a solution like in version 2 the depth bound would have to be integrated into the state.

Deductive Reasoning Agents

Example: Theorem Proving

Implicational calculus

Axioms

$\alpha \to (\beta \to \alpha)$

$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$

$(\neg\alpha \to \neg\beta) \to (\beta \to \alpha)$

Modus Ponens

$$\frac{\alpha \quad \alpha \to \beta}{\beta}$$

Q. Is it possible to derive $\alpha \to \neg\neg\alpha$ from the axioms using modus ponens?

Modeling

$\forall x \forall y \; \mathit{True}\,(i(x, i(y, x)))$

$\forall x \forall y \forall z \; \mathit{True}\,(i(i(x, i(y, z)), i(i(x, y), i(x, z))))$

$\forall x \forall y \; \mathit{True}\,(i(i(n(x), n(y)), i(y, x)))$

} Initial state.

$\forall x \forall y \; (\mathit{True}\,(i(x, y)) \wedge \mathit{True}\,(x) \to \mathit{True}\,(y))$ Action schema.

Find a plan to reach a state with $\forall x \, \mathit{True}\,(i(x, n(n(x))))$.

Chapter MK:VI

VI. Planning and Configuration

- Agent Systems
- Deductive Reasoning Agents
- Planning Language

- Planning Algorithms
- State-Space Planning
- Plan Space Planning
- HTN Planning
- Complexity of Planning Problems
- Erweiterungen

- Konfigurierungsproblemstellung
- Konfigurierungsansätze

Planning Language

STRIPS Planning Language

(**ST**anford **R**esearch **I**nstitute **P**roblem **S**olver)

Language for describing planning problems

- developed by Fikes and Nilsson in 1971,
- applied in *Shakey*,
  (Mobile robot of the Stanford Research Institute (SRI)
  integrating perception, planning and execution),
- emerged form an overlap of
  - state space search,
  - theorem proving, and
  - control theory.
- Planning using propositional STRIPS models is PSPACE-complete.

[Bylander 1991]

Remarks:

- STRIPS is still part of the langugage PDDL (**P**lanning **D**omain **D**escription **L**anguage) which is used in the *International Planning Competition* (IPC) for the specification of planning problems.

Planning Language

STRIPS Language (restricted version)

- The STRIPS language is based on the function-free first-order logic (with equality and order-sorted). STRIPS specifications contain
  - a finite set of constant symbols denoting objects in the domain,
  - a finite set of predicate symbols denoting properties in the domain,
  - no function symbols.

  ➜ *Domain Closure Assumption (DCA)*
  All objects in the domain are denoted by constants.

  ➜ *Unique Name Assumption (UNA)*
  Different constants denote different objects.

- In STRIPS a *state* is specified by a finite set (i.e. a conjunction) of variable-free atomic formulas (i.e. positive literals).

  ➜ *Closed World Assumption (CWA)*
  All atomic formulas of a state are assumed to be true, all other variable-free atomic formulas are assumed to be false.

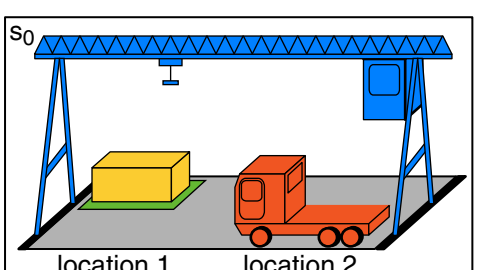
Remarks:

- A STRIPS specification is finite.
- A STRIPS specification contains only constants and variables as terms.
- Since state descriptions contain only variable-free atomic formulas and the number of constants and the number of predicates are finite, only a finite number of states can be distinguished.
- A STRIPS state is based on facts (i.e. atomic formulas). More complex formulas restrict a domains in a more intricate way. E.g., by using implications it can be expressed that each state complies with certain rules. On the basis of facts, we can only describe special cases of such situations.

Planning Language

Example: Loading Dock (continued)

- Constants, e.g.
  - *truck1* denoting vehicles,
  - *location1* and *location2* denoting possible vehicle positions,
  - *c1* denoting container,
  - *stack1* denoting possible container storage positions,
  - *crane1* denoting cranes.
- Predicates, e.g.
  - *at*(x, y) denoting "vehicle x is at position y",
  - *empty*(x) denoting "crane x is unused",
  - *unloaded*(x) denoting "vehicle x is unloaded",
  - *top*(x, y) denoting "container x is on top of stack y",
  - *in*(x, y) denoting "container x is on stack y".
- States, e.g.
  $\{$ *at*(*truck1*, *location2*), *unloaded*(*truck1*),
  in(*c1*, *stack1*), *top*(*c1*, *stack1*), *empty*(*crane1*) $\}$



Planning Language

STRIPS Language (restricted version) (continued)

- *Operator* are denoted by *op*$(x_1, \ldots, x_n)$ with unique names *op* and the list of variables $x_1, \ldots, x_n$ that occur in the specification. Preconditions and effects are specified by finite sets (i.e. conjunctions) of literals (negated or non-negated atomic formulas).

- *Actions* are applications of operators: variables are substituted by constants.

Definition 3 (STRIPS Model, STRIPS State)

A *STRIPS model* $\mathcal{S} = (C, P, O)$ is given by a finite set C of constants, a finite set P of predicate symbols with some arity, and a finite set O of operator descriptions of the following form and using only constants in C, predicates in P, and variables.

operator : *operator_name*$(x_1, \ldots, x_n)$
precond: finite list of literals describing preconditions,
effects: finite list of literals describing effects.

A *state* in a STRIPS model is described by a finite set variable-free positive literals assuming DCA, UNA, and CWA.
An *action* is a ground instance of an operator (i.e. not containing variables).

Remarks:

- Operator names should be different from names in C and P and from variable names.
- In the effects *effects* of operators, variables can occur that do not occur in the preconditions *precond*. Since actions are ground instances of operators, these variables are also instantiated.
  When searching for the actions that are applicable in a state, focus is on the precondition part *precond* of the operators. Variables that occur only in the *effects* part could then be understood as universally quantified. One could think that, therefore, all possible instances of the corresponding literals are to be be removed or added. This is NOT permitted.
- The precondition *precond* of an operator should not contain an atomic formula negated and non-negated at the same time, because such a precondition cannot be fulfilled by any state description. In principle, this requirement also applies to actions. Ground instances of operators should not cause complementary literals in actions. However, the second requirement is not always met in favour of simpler operator descriptions.
- In the effects *effects* of a ground instance of an operators, no atomic formula should occur negated and non-negated at the same time. A procedural question arises for an action that contains literals in both polarities in the effects. Are the parts of the state that are no longer valid in the subsequent state first removed and then the descriptions of the new positive facts resulting from the action are added, or vice versa?

Planning Language

Example: Loading Dock (continued)

- Operator (simple version)

  operator: *move*(t, l_1, l_2)

  ;; truck t moves from location l_1 to l_2

  precond: *at*(t, l_1)

  effects: *at*$(t, l_2), \neg$*at*(t, l_1)

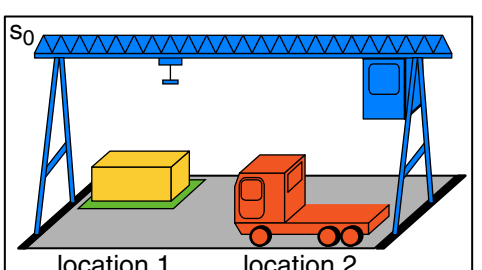
- Operator (more complex version)

  operator : *move*(t, l_1, l_2)

  ;; truck t moves from location l_1 to l_2

  precond: *at*(t, l_1), *adjacent*$(l_1, l_2), \neg$*occupied*(l_2)

  effects: *at*$(t, l_2), \neg$*at*(t, l_1), *occupied*$(l_2), \neg$*occupied*(l_1)



Q. Which instances of *move*(t, l_1, l_2) are applicable?

Planning Language

STRIPS Language (restricted version) (continued)

STRIPS Assumption (avoiding the Frame Problem):
Only those parts of a state description that are explicitly mentioned in the description of an action (*precond* resp. *effects* part) are affected by the execution of the action.

Definition 4 (STRIPS Action Application)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given and let s be a state described in $\mathcal{S}$.

An action a that is a ground instance of an operator $o \in O$ is *applicable* in s iff

- all non-negated literals in *precond*(a) are contained in s and
- no literal that occurs negated in *precond*(a) is contained in s.

A state s' described in $\mathcal{S}$ is *successor state* of s by an applicable action a iff

$$s' = (s \setminus \{l \text{ positive literal } | \neg l \in \textit{effects}(a)\}) \cup \{l \text{ positive literal } | l \in \textit{effects}(a)\})$$

Remarks:

- Successor states for an applicable action are computed from the current state and the action description:
  - *first step:* literals that occur negated in the effects part *effects* are deleted from the state description,
  - *second step:* literals that occur non-negated in the effects part *effects* are added to the state description.

- For an operator *op*$(x_1, \ldots, x_n)$ and an action *op*$(c_1, \ldots, c_n)$ denote by
  - *precond*$^+$(*op*$(c_1, \ldots, c_n)$) the set of atomic formulas that occur in the preconditions of *op*$(c_1, \ldots, c_n)$ as non-negated literals,
  - *precond*$^-$(*op*$(c_1, \ldots, c_n)$) the set of atomic formulas that occur in the preconditions in negated literals,
  - *effects*$^+$(*op*$(c_1, \ldots, c_n)$) the set of atomic formulas that occur in the effects of *op*$(c_1, \ldots, c_n)$ as non-negated literals, and
  - *effects*$^-$(*op*$(c_1, \ldots, c_n)$) the set of atomic formulas that occur in the effects in negated literals.

  Then, an action *op*$(c_1, \ldots, c_n)$ is applicable in state s (described by a set of ground atomic formulas) if and only if

$$\textit{precond}^+(\textit{op}(c_1, \ldots, c_n)) \subseteq s \quad \text{and} \quad \textit{precond}^-(\textit{op}(c_1, \ldots, c_n)) \cap s = \emptyset$$

  The successor state s' is described by

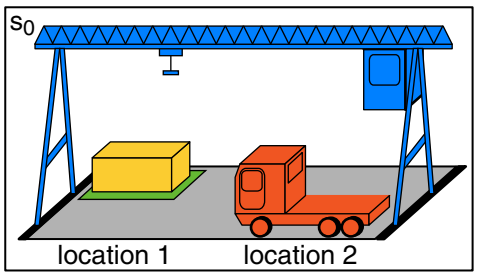
$$s' = (s \setminus \textit{effects}^-(\textit{op}(c_1, \ldots, c_n))) \cup \textit{effects}^+(\textit{op}(c_1, \ldots, c_n))$$

Planning Language

Example: Loading Dock (continued)

- State

  $\{at(truck1, location2), unloaded(truck1),$
  $in(c1, stack1), top(c1, stack1), empty(crane1)\}$



- Operator

  operator: $move(t, l_1, l_2)$

  ;; truck t moves from location l_1 to l_2

  precond: $at(t, l_1)$

  effects: $at(t, l_2), \neg at(t, l_1)$

- Action defined by ground substitution $[t/truck1,\ l_2/location1,\ l_1/location2]$

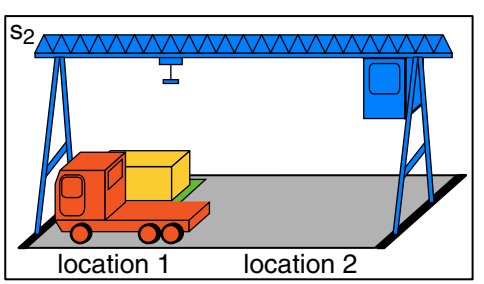
  action: $move(truck1, location2, location1)$

  precond: $at(truck1, location2)$

  effects: $at(truck1, location1), \neg at(truck1, location2)$

- Successor state

  $\{\ at(truck1, location1), unloaded(truck1),$
  $in(c1, stack1), top(c1, stack1), empty(crane1)\ \}$



Planning Language

STRIPS Language (restricted version) (continued)

Definition 5 (STRIPS Goal, STRIPS Plan)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given.

A *STRIPS goal* is described by a finite set (i.e. conjunction) of variable-free literals.

A goal is satisfied (or achieved) in a state if all non-negated literals of the goal description are contained in the state description and no literal occurring negated in the goal description is contained in the state description.

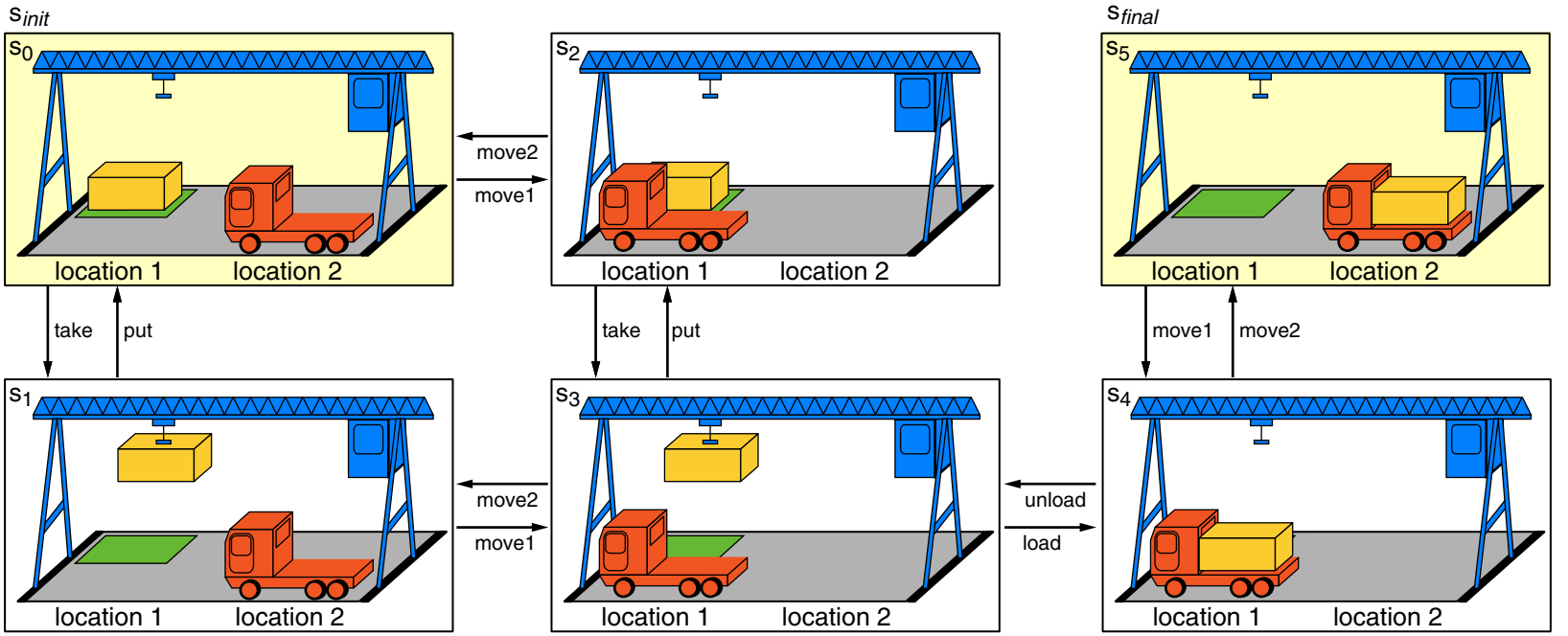
A state satisfying the goal is a *goal state*. A *plan* is a sequence of actions.

- A goal may contain negated literals. In contrast to the initial state a goal is an incompletely specified state.
- A goal is satisfied (or achieved) in a state if the state "contains" the goal. (This subset relation has to realize CWA: literals occurring non-negated in the goal are contained in the literal set specifying the state, no literal occurring negated in the goal is contained in the state.)
- A *plan* should be applicable in a given state. A successor state of a plan can be defined inductively.

➜ We are interested in plans leading from an initial state to a goal state.

Planning Language

Example: Loading Dock (continued)



Plan:

1. $move(truck1, location2, location1)$,
2. $take(crane1, c1, ground, stack1)$,
3. $load(crane1, c1, truck1)$,
4. $move(truck1, location1, location2)$

Planning Language

Example: Blocks World [Nilsson]

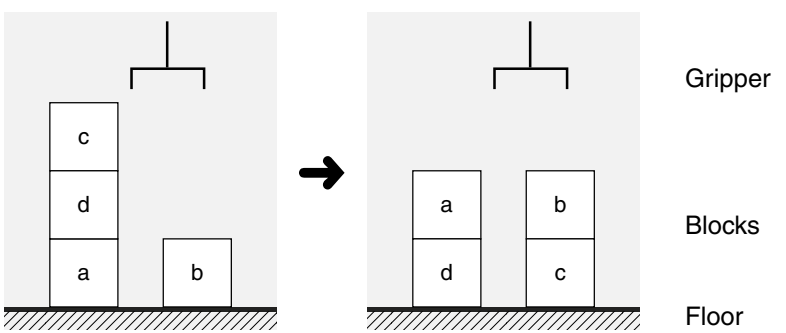
Compact Model

- Constants: a, b, c, d, *floor*
- Predicates: *on*(x, y), *clear*(x)
- Operator:

  operator : *move*(x, y, z)

  precond : *on*(x, y), *clear*(x), *clear*(z)

  effects : $\neg$*on*(x, y), $\neg$*clear*(z), *on*(x, z), *clear*(y), *clear*$(floor)$



- Initial state: {*on*(d, a), *on*(c, d), *on*$(a,$ *floor*$)$, *on*$(b,$ *floor*$)$, *clear*(b), *clear*(c), *clear*$(floor)$}
- Goal: {*on*(a, d), *on*$(d,$ *floor*$)$, *on*(b, c), *on*$(c,$ *floor*$)$}
- Necessary actions: *move*$(a,$ *floor*$, d)$, *move*$(b,$ *floor*$, c)$, *move*$(c, d,$ *floor*$)$, *move*$(d, a,$ *floor*$)$
- Plan: (*move*$(c, d,$ *floor*$)$, *move*$(d, a,$ *floor*$)$, *move*$(a,$ *floor*$, d)$, *move*$(b,$ *floor*$, c)$)

Remarks:

- Obviously, not every ground instance of the operator is a meaningful resp. legal action in this modeling:
  - Substitution $[x/a, y/\textit{floor}, z/\textit{floor}]$ leads to conflicts in the effects part, since atom $\textit{clear}(\textit{floor})$ occurs negated as well as non-negated.
    The order in which the next state is created resolves the conflict:

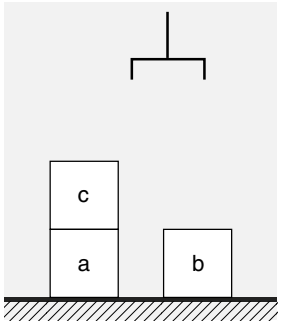
    > "First remove negated atoms in the effects part from the state description, then add the non-negated atoms to the state description."

  - Substitution $[x/\textit{floor}, y/a, z/b]$ results in an action that is not applicable in any "real" state, i.e. in a state that represents a real world situation.
    Assuming a "good" modeling of situations in the environment as states and adequate modeling of state transitions, no action will be applicable that is not meaningful.
    Alternatively, sort predicates can be used, e.g. $\textit{block}(x)$ in the precond part.

- For the sake of brevity of the description, the property $\textit{clear}(\textit{floor})$ is modeled in such a way that it can be consumed. Therefore, this atom has simply been added to the effects part of the operator, but this is not a new property of the successor state in every instantiation.

Planning Language

Example: Blocks World [Nilsson] (continued)



- Constants: a, b, c
- Predicates: *on*(x, y), *ontable*(x), *clear*(x)
  holding(x), *handempty*$()$
- Operators:

  operator : *pickup*(x)

  precond : *ontable*(x), *clear*(x), *handempty*$()$
  effects : *holding*(x), $\neg$*ontable*(x), $\neg$*clear*(x), $\neg$*handempty*$()$

  operator : *putdown*(x)

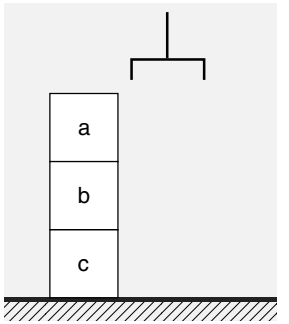
  precond : *holding*(x)
  effects : *ontable*(x), *clear*(x), *handempty*$()$, $\neg$*holding*(x)

  operator : *stack*(x, y)

  precond : *holding*(x), *clear*(y)
  effects : *on*(x, y), *clear*(x), $\neg$*clear*(y), $\neg$*holding*(x), *handempty*$()$

  operator : *unstack*(x, y)

  precond : *on*(x, y), *clear*(x), *handempty*$()$
  effects : *holding*(x), *clear*(y), $\neg$*on*(x, y), $\neg$*clear*(x), $\neg$*handempty*$()$



- Initial state: {*clear*(b), *clear*(c), *on*(c, a), *handempty*$()$, *ontable*(a), *ontable*(b)}
- Goal: {*on*(b, c), *on*(a, b)}
- Plan: (*unstack*(c, a), *putdown*(c), *pickup*(b), *stack*(b, c), *pickup*(a), *stack*(a, b))

Remarks:

- Despite a more detailed description of states, not all basic instances of operators are useful in this specification, e.g. action *unstack*(a, a).
- A viable alternative is to describe operators using sort predicates and inequalities in the condition parts and an adequate initial state:

operator : *move*(x, y, z)
precond : *block*(x), *block*(z), $x \neq y, y \neq z, x \neq z$, *on*$(x, y)$, *clear*$(x)$, *clear*$(z)$
effects : *on*(x, z), $\neg$*on*(x, y), *clear*(y), $\neg$*clear*(z)

operator : *move*$(x, y,$ *floor*$)$
precond : *block*(x), *block*(y), $x \neq y, y \neq$ *floor*, $x \neq$ *floor*, *on*(x, y), *clear*(x), *clear*$($*floor*$)$
effects : *on*(x, z), $\neg$*on*(x, y), *clear*(y)

Initial state: {*block*(a), *block*(b), *block*(c), *block*(d), $a = a, b = b, c = c, d = d$, *floor* $=$ *floor*, *on*(d, a), *on*(c, d), *on*$(a,$ *floor*$)$, *on*$(b,$ *floor*$)$, *clear*(c), *clear*$($*floor*$)$}

Planning Language

Classical Planning in STRIPS

Definition 6 (STRIPS Planning Problem, STRIPS Plan)

A STRIPS planning problem $(\mathcal{S}, s_{init}, c_{goal})$ is given by

- a STRIPS model $\mathcal{S}$,
- a start state s_{init}, and
- a goal c_{goal} defining conditions for goal states.

A plan $(a_1, \ldots, a_k)$ of actions, i.e. a sequence of ground instances of operators in O, is a solution for the planning problem $(\mathcal{S}, s_{init}, c_{goal})$ if and only if there are states $s_1, \ldots, s_k$ such that with $s_0 = s_{init}$

- a_i is applicable in s_{i-1} and leads to state s_i for $i = 1, \ldots, k$ and
- s_k satisfies the goal condition c_{goal}.

Searching for plans on basis of a STRIPS model is *domain-independent planning*, because no additional knowledge about meaningful actions, possible dead-ends, illegal states, etc. is involved. But domain knowledge is needed for defining useful STRIPS models.

Remarks:

- Plan described by a sequence of actions are also called linear plans.
- The length of a plan is the number of actions contained in the sequence.
- The language of a STRIPS planning problem is implicitly given by the start state s_{init} and the preconditions *precond* and the effects *effects* of the operators.
  - Only predicates mentioned in these descriptions can occur in successor states.
  - Only constants mentioned in these descriptions can occur in successor states.
- As the set of constants is finite, the finite set of operators O defines a finite set of actions A.
- Starting from the start state, the transitive closure of states reachable by actions defines a finite set of states that can be described in $\mathcal{S}$.
- States and actions define a transition function.
- The set of goal states F can be given as

$$F := \{s \text{ state in } \mathcal{S} \mid c_{goal} \text{ is satisfied in } s\}$$

- Goal conditions in STRIPS planning problem could contain variables that are considered to be existentially quantified. This is no increase in expressiveness. (A new operator can be defined that uses this goal as precondition and some new ground atom l as consequence. A plan for the planning problem to achieve l defines a plan for the original problem.)