APLA: Automated planning

Loïg Jezequel

Master CORO – 2019/2020

Outline

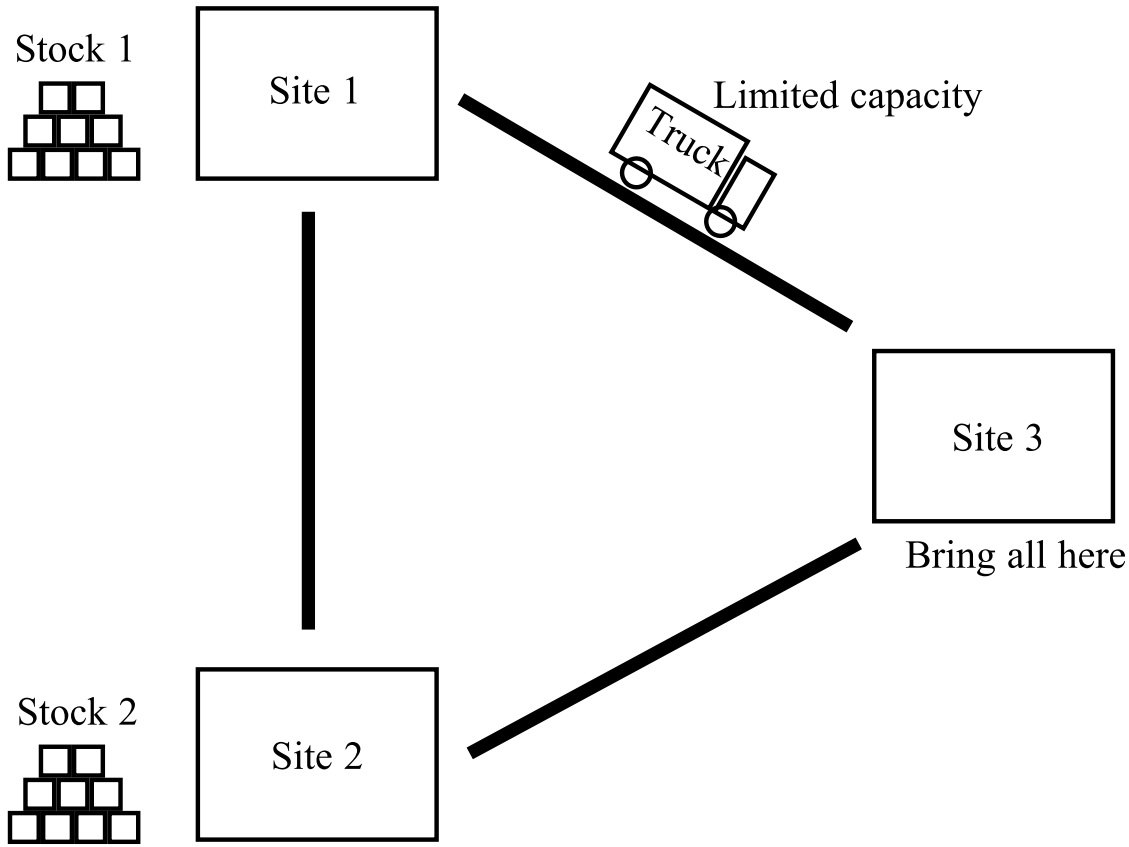
What is planning?

Informal definition

Choose and order actions in order to achieve as best as possible some objective.

Automated planning

AKA AI Planning



Different kinds of planning

Motion planning

Synthesis of a geometric path from a starting position in space to a goal, control trajectory along that path.

Examples

Move a truck, a mechanical arm, a robot, a virtual character, etc.

Perception planning

Plans involving sensing actions for gathering information.

Example

Build a virtual model of an environment from cameras.

Navigation planning

Combination of motion planning and perception planning

Different kinds of planning, continued

Manipulation planning

Handling objects.

Examples of actions

Pick objects, insert objects in assemblies, push objects, etc.

Communication planning

Solving cooperation problems between agents.

Different kinds of planning, continued

Manipulation planning

Handling objects.

Examples of actions

Pick objects, insert objects in assemblies, push objects, etc.

Communication planning

Solving cooperation problems between agents.

And many more problems...

Vocabulary

Plan

A solution to a planning problem.

Planner

A tool for solving planning problems.

Planning domain

The set of actions, initial situation, goals. . . defining a planning problem

Domain-dependent VS. domain-independent planning

Domain-dependent planner

- Made for a specific domain.
- Won't work in any other domain.
- Generally uses techniques that do not generalize to other domains.
- Most successful real-world planners work this way.

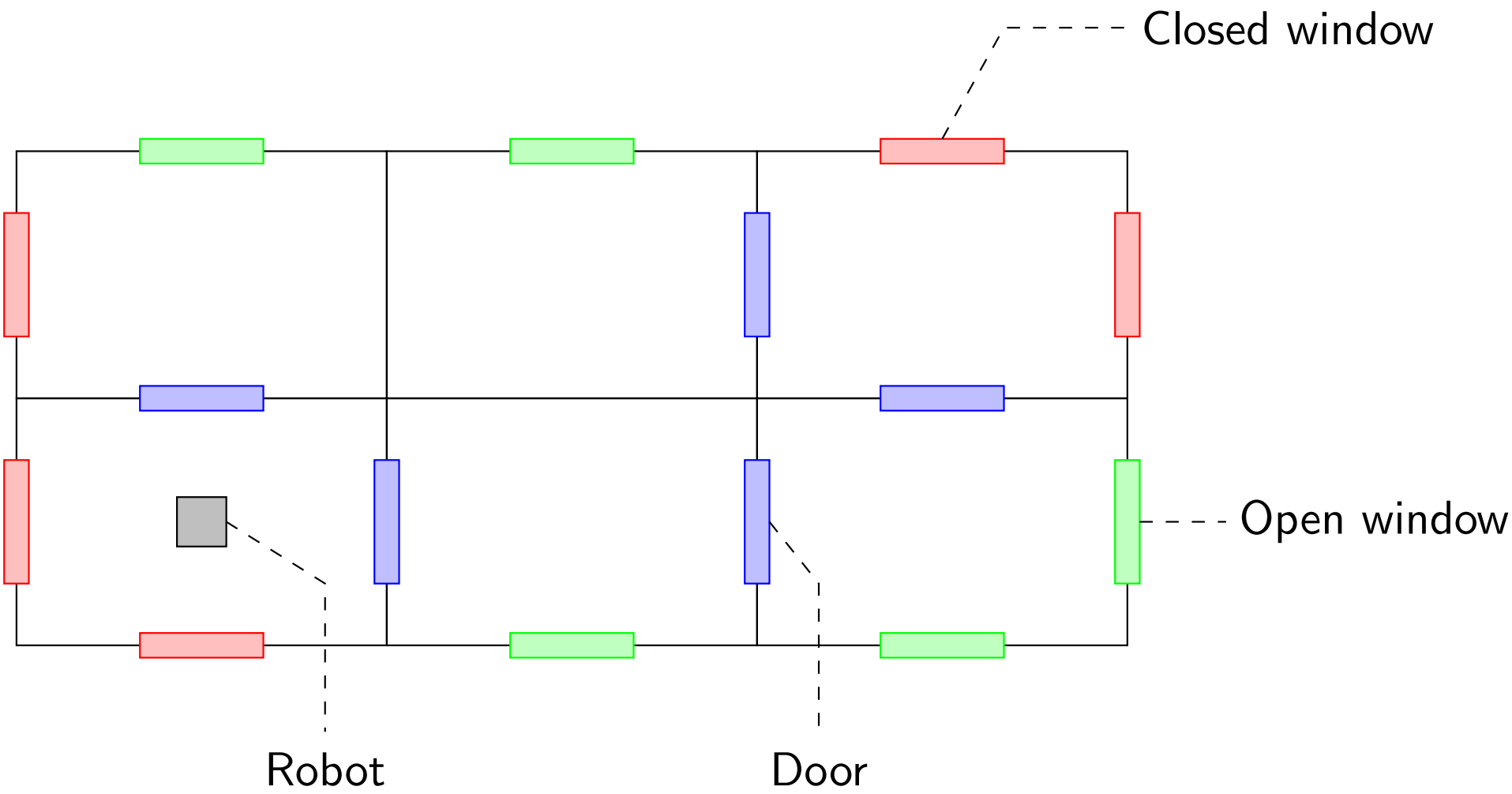
Domain-independent planner

- Works in any domain.
- In practice, needs restrictions of the domains.

  (Fundamental assumptions of domain-independent planning)
- No domain-specific knowledge (except the description of the domain).

Modeling planning domains

An example: rooms and robot



State-transition systems

What do we need to model

- States of a system:
  - position of the robot,
  - state of each window (open/closed).
- Actions moving the system from one state to another state:
  - moves of the robot,
  - opening/closing of windows.

State-transition systems

What do we need to model

- States of a system:
  - position of the robot,
  - state of each window (open/closed).
- Actions moving the system from one state to another state:
  - moves of the robot,
  - opening/closing of windows.

Solution

Use of state-transition systems (also called discrete-event systems):

- states of the system represented by a set of abstract states,
- actions represented by a set of transitions between states.

State-transition systems formalism

Syntax

$\Sigma = (S, A, E, \gamma)$ with:

- $S = \{s_1, s_2, \dots\}$ a recursively enumerable set of *states*,
- $A = \{a_1, a_2, \dots\}$ a recursively enumerable set of *actions*,
- $E = \{e_1, e_2, \dots\}$ a recursively enumerable set of *events*,
- $\gamma : S \times A \times E \to 2^S$ a *state-transition function*.

Semantics

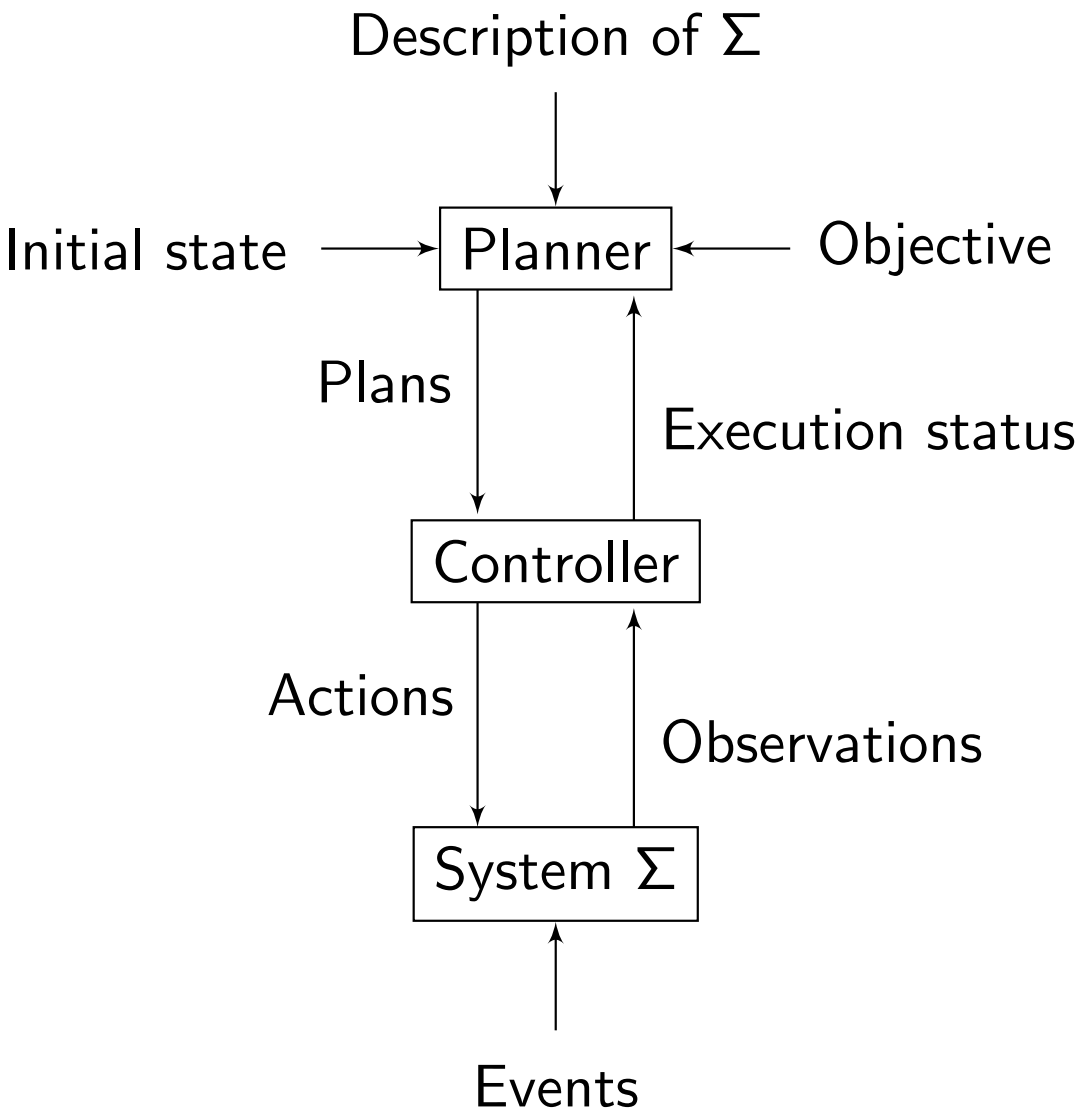
- $\gamma(s, a, e) = \{s_1, \dots, s_n\}$,
- $\gamma(s, \text{no-op}, e) = \{s_1, \dots, s_n\}$, internal dynamics of the system,
- $\gamma(s, a, \varepsilon) = \{s_1, \dots, s_n\}$, action controlled by the planner.

Back to our example

Exercise

Propose a model for the rooms and robot example.

Conceptual model for planning



Initial state

State of the system at the start of the planning task.

Objective

Some conditions on the sequence of states followed by the system:

- set of goal states,
- states to avoid,
- states to go through,
- in a particular order,
- etc.

Fundamental assumptions

A0: Finiteness

Assumption

The system Σ has a finite set of states.

Relaxation

Some systems may have enumerable infinite sets of states.

- Think about the creation of new objects in a plan.
- Decidability issue.
- Termination issue.

A1: Full observability

Assumption

The observation function η is the identity function.

Remark

Only needed at the initial state in deterministic systems (A2).

Relaxation

Some systems are not fully observable: planning under uncertainty.

A2: Determinism

Assumption

For every state s and every event or action u, $|\gamma(s, u)| \leq 1$.

Relaxation

The plans must be able to contain alternatives and iterations:

- do a and depending on the result do b or c;
- do a until a given result is obtained.

A3: Stasis

Assumption

The set of events E is empty.

Relaxation

The systems becomes non-deterministic from the viewpoint of the planner: from a state s an action a can lead to several results.

A4: Restricted goals

Assumption

Goals are defined by final states only, not by the states traversed.

Not allowed

- States to be avoided.
- Constraints on state trajectories.
- Utility functions.
- ...

A5: Sequential plans

Assumption

A plan is a linear sequence of actions.

Alternatives

- Partial order of actions.
- Conditional plan.
- Universal plan (maps states to actions).
- Automaton (maps histories of execution to actions)
- ...

A6: Implicit time

Assumption

Actions and events have no duration.

Consequences

- No temporally constrained goals.
- Synchronicity of the controller.

A7: Offline planning

Assumption

Changes in Σ while planning are not possible.

Relaxation

- Implies plan revision and/or re-planning.
- May be modeled as offline planning (with state-explosion).

Restricted model

The models we consider for now

Verify the previous 8 assumptions:

1. finiteness,
2. full observability,
3. determinism,
4. statis,
5. restricted goals,
6. sequential plans,
7. implicit time,
8. offline planning.

Later on

Relaxation of some of the hypothesis.

Bibliography

Success stories

Deep space 1

NASA' spacecraft launched in 1998, encountered the comet Borrelly.
Driven from May 17 to May 21, 1999 by an automated planning system.

MAPGEN

Planning and scheduling system currently in daily use in the Mars Exploration Rover mission of NASA.

And many more...

Bibliography

- *Automated Planning – Theory and Practice*. Malik Ghallab, Dana Nau, Paolo Traverso. Morgan Kaufmann Publishers, 2004.
- *Automated Planning and Acting*. Malik Ghallab, Dana Nau, Paolo Traverso. Cambridge University Press, 2016.
- *Artificial Intelligence – A Modern Approach*, third edition. Stuart Russel, Peter Norvig. Pearson Education, 2016.
- *Constraint Processing*. Rina Dechter. Morgan Kaufmann Publishers, 2003.

Outline

Motivation

Why don't we use the state-transition systems directly?

Far too many states to represent them explicitly.

Solution

- Represent each state as a set of features, for example:
  - a vector of values for a set of variables,
  - a set of ground atoms in some logic.
- Define a set of *operators* from which state-transitions can be computed:
  - operators work on the set of features representing the states,
  - actions work on the states.
- Only give the initial state explicitly, generate the others from the operators.

Classical representation

The language

Constant symbols (finitely many)

For example:

- rbt (the robot),
- rm1, rm2 (two rooms),
- w1, w2 (two windows).

Predicates (finitely many) – called *atoms*

For example:

- isIn(rbt, rm)
- windowOf(w, rm)
- isOpen(w)

Expressions

Ground expression

Predicate with no variable symbols in args (only constants):

- isIn(rbt, rm1)
- isIn(rbt, rm2)

Unground expression

Predicate with variable symbols in args:

- isIn(rbt, x)
- isOpen(y)

States

State

A set of ground atoms.

Exercise

List the states for the rooms and robot example.

States

State

A set of ground atoms.

Exercise

List the states for the rooms and robot example.

Exercise

Is it possible to have an infinite number of states in a planning problem?

Operators

Operator

$o = (name(o), precond(o), effects(o))$ with:

- $precond(o)$ a set of literals that must be true to use o,
- $effects(o)$ a set of literals that o will make true,
- $name(o) = n(x, y, z, \dots)$ a name for the operator with:
  - n a unique operator symbol,
  - $x, y, z, \dots$ a list of variables that appear in o.

Exercise

Propose a set of operators for the rooms and robot problem.

Action

A ground *instance* of an operator.

Some notations

For a an action (or operator):

- $precond^+(a)$ is the set of atoms that appear positively in a's preconditions,
- $precond^-(a)$ is the set of atoms that appear negatively in a's preconditions,
- $effects^+(a)$ is the set of atoms that appear positively in a's effects,
- $effects^-(a)$ is the set of atoms that appear negatively in a's effects.

Executing actions

Executable action

An action a is executable in a state s if:

- $precond^+(a) \subseteq s$, and
- $precond^-(a) \cap s = \emptyset$.

Executing an action

If an action a is executable in s, the execution of a results in the new state s' such that:

$$s' = (s \setminus effects^-(a)) \cup effects^+(a) = \gamma(s, a)$$

Planning problem

Planning domain

Constants + atoms + operators

Planning problem

Planning domain + initial state + set of goal states

Planning problem (statement)

$P = (O, s_0, g)$ with:

- O a set of operators,
- s_0 an initial state,
- g a set of goal states.

Back to our example

Exercise

Give a problem statement for the rooms and robot example. What actions are applicable from the initial state ?

Plans

Execution

A sequence $a_0, a_1, \ldots, a_n$ is an execution of the problem (O, s_0, g) if:

- $s_1, \ldots, s_{n+1}$ are states of the problem,
- $\forall 0 \leq i \leq n, a_i$ is executable from s_i, and
- $\forall 0 \leq i \leq n, \gamma(s_i, a_i) = s_{i+1}$.

Plan

An execution of a problem is a plan if $s_{n+1} \in g$.

Plans

Execution

A sequence $a_0, a_1, \ldots, a_n$ is an execution of the problem (O, s_0, g) if:

- $s_1, \ldots, s_{n+1}$ are states of the problem,
- $\forall 0 \leq i \leq n, a_i$ is executable from s_i, and
- $\forall 0 \leq i \leq n, \gamma(s_i, a_i) = s_{i+1}$.

Plan

An execution of a problem is a plan if $s_{n+1} \in g$.

Exercise

Give a plan for your problem statement of the rooms and robot example.

Set-theoretic representation

Classical representation and set-theoretic representation

Intuition

Set-theoretic representation corresponds to classical representation with ground atoms only.

In practice: states

Instead of ground atoms one uses propositions (i.e. boolean variables), for example:

- isIn(rbt, rm1) becomes isIn-rbt-rm1,
- windowOf(w1, rm1) becomes windowOf-w1-rm1,
- isOpen(w1) becomes isOpen-w1.

Classical representation and set-theoretic representation 2

In practice: operators and actions

No operators, only actions:

- ground atoms $\rightarrow$ propositions,
- negative effects $\rightarrow$ delete list,
- positive effects $\rightarrow$ add list,
- negative preconditions $\rightarrow$ new atoms:
  - instead of $\neg$a as precondition use not-a,
  - delete a iff not-a is added,
  - delete not-a iff a is added.

Classical representation and set-theoretic representation 2

In practice: operators and actions

No operators, only actions:

- ground atoms $\rightarrow$ propositions,
- negative effects $\rightarrow$ delete list,
- positive effects $\rightarrow$ add list,
- negative preconditions $\rightarrow$ new atoms:
  - instead of $\neg$a as precondition use not-a,
  - delete a iff not-a is added,
  - delete not-a iff a is added.

Exercise

Rewrite the rooms and robot problem in set-theoretic representation.

Important remark

Problem representation size

There can be an exponential blowup of the problem representation size between the classical representation and the set-theoretic representation.

Exercise

Estimate this blowup on a generalised version of the rooms and robot problem (i.e. with parameterized numbers of rooms and windows).

State-variable representation

Overview of the state-variable representation

Principle

- Use ground atoms for properties that do not change:
  - windowOf(w1, rm1),
- assign values to state variables for properties that change:
  - isIn(rbt) $=$ rm1 (precondition),
  - isIn(rbt) $\leftarrow$ rm1 (effect).

Overview of the state-variable representation

Principle

- Use ground atoms for properties that do not change:
  - windowOf(w1, rm1),
- assign values to state variables for properties that change:
  - isIn(rbt) $=$ rm1 (precondition),
  - isIn(rbt) $\leftarrow$ rm1 (effect).

exercise

Rewrite the rooms and robot problem in state-variable representation.

Comparisons between representations

Expressive power

Any problem that can be represented in one representation can also be represented in the other two.

Set-theoretic to classical

Trivial: each proposition is a 0-ary predicate.

state-variable to classical

$f(x_1, \ldots, x_n) = y$ becomes $P_f(x_1, \ldots, x_n, y)$.

Classical to state-variable

$P(x_1, \ldots, x_n)$ becomes $f_P(x_1, \ldots, x_n) = 1$.

classical to set-theoretic

Write all the ground instances $\Rightarrow$ exponential blowup.

Last remarks

Classical representation

The most popular for classical planning (historical reasons).

Set-theoretic representation

Useful in algorithms that manipulate ground atoms directly, useful for theoretical studies.

State-variable representation

Useful as a way to handle numbers, functions, times, etc, more naturally than in the classical representation.

The PDDL language

Languages for describing planning problems

Starting point

We have a formal representation of planning problems (usually the classical representation).

In practice

If one implements a planner, she needs an input language to describe the problems to solve (many have existed).

PDDL

Planning Domain Definition Language: a language for describing planning problems, used in the IPC (International Planning Competition).

A very wide language

PDDL encompasses a very wide range of problem representations

- STRIPS (= classical representation),
- various kinds of preconditions (negative, disjunctive, existential, universal, etc)
- conditional effects,
- numeric functions,
- durative actions (with constraints, with effects on numeric functions),
- timed literals,
- action costs,
- etc.

Planners and PDDL

Complete syntax of PDDL

`https://helios.hud.ac.uk/scommv/IPC-14/repository/kovacs-pddl-3.1-2011.pdf`

In practice, planners deal with subsets of PDDL.

PDDL fragment for IPC 2018

STRIPS, action costs, negative preconditions, conditional effects.

An example (1)

```
(define (domain BLOCKS)
(:requirements :strips)
(:predicates (on ?x ?y)
             (ontable ?x)
             (clear ?x)
             (handempty)
             (holding ?x))

(:action pick-up
   :parameters (?x)
   :precondition (and (clear ?x) (ontable ?x) (handempty))
   :effect (and (not (ontable ?x))
                (not (clear ?x))
                (not (handempty))
                (holding ?x)))
```

An example (2)

```
(:action put-down
   :parameters (?x)
   :precondition (holding ?x)
   :effect (and (not (holding ?x))
                (clear ?x)
                (handempty)
                (ontable ?x)))

(:action stack
    :parameters (?x ?y)
    :precondition (and (holding ?x) (clear ?y))
    :effect (and (not (holding ?x))
                 (not (clear ?y))
                 (clear ?x)
                 (handempty)
                 (on ?x ?y)))
```

An example (3)

```
(:action unstack
    :parameters (?x ?y)
    :precondition (and (on ?x ?y) (clear ?x) (handempty))
    :effect (and (holding ?x)
                 (clear ?y)
                 (not (clear ?x))
                 (not (handempty))
                 (not (on ?x ?y)))))
```

An example (4)

```
(define (problem BLOCKS-10-0)
(:domain BLOCKS)
(:objects D A H G B J E I F C )
(:INIT (CLEAR C) (CLEAR F) (ONTABLE I) (ONTABLE F)
       (ON C E) (ON E J) (ON J B) (ON B G) (ON G H)
       (ON H A) (ON A D) (ON D I) (HANDEMPTY))
(:goal (AND (ON D C) (ON C F) (ON F J) (ON J E)
            (ON E H) (ON H B) (ON B A) (ON A G)
            (ON G I))))
```

An example (4)

```
(define (problem BLOCKS-10-0)
(:domain BLOCKS)
(:objects D A H G B J E I F C )
(:INIT (CLEAR C) (CLEAR F) (ONTABLE I) (ONTABLE F)
       (ON C E) (ON E J) (ON J B) (ON B G) (ON G H)
       (ON H A) (ON A D) (ON D I) (HANDEMPTY))
(:goal (AND (ON D C) (ON C F) (ON F J) (ON J E)
            (ON E H) (ON H B) (ON B A) (ON A G)
            (ON G I))))
```

Exercise

Propose a PDDL coding of the rooms and robot problem.

Outline

Starting point

Huge state spaces

We have seen that, even small planning problems can have huge state spaces,

- solving a problem is almost the same as exploring its full state space,
- so it can be a difficult task.

Starting point

Huge state spaces

We have seen that, even small planning problems can have huge state spaces,

- solving a problem is almost the same as exploring its full state space,
- so it can be a difficult task.

Can we state formally to which extent planning is a difficult task?

Starting point

Huge state spaces

We have seen that, even small planning problems can have huge state spaces,

- solving a problem is almost the same as exploring its full state space,
- so it can be a difficult task.

Can we state formally to which extent planning is a difficult task?

Is planning even a feasible task?

Decidability and complexity

Decidability

What you should know

There exist problems that are not decidable: no algorithm can solve them

Example of undecidable problem: the halting problem

Given an arbitrary program and a finite input for this program, decide whether the program finishes running or not.

Showing that a problem P is undecidable

Usually by reduction of an undecidable problem P': show that if one can decide P it can decide P' as well by giving a way to transform any instance of P' into an instance of P.

How can one evaluate the difficulty of solving a decidable problem?

Time-complexity vs. space-complexity

Complexity mesure

- A function of the size of the problem,
- expressed in time or space needed for solving it.

Link between time and space

x-space implies x-time at least

- time needed to use the space

Complexity of an algorithm vs. complexity of a problem

(Worst case) complexity of an algorithm

Time/space used by this algorithm for solving the instance of a problem that is the worst for it:

- time: number of elementary operations,
- space: number of elementary variables.

Our case

We are interested in the general complexity of solving planning problems:

- worst case complexity of the best possible algorithm for solving planning problems from a given class,
- sometimes we may not know this algorithm.

Complexity classes

Notations

- Space or time:
  - SPACE: in space,
  - TIME: in time.
- Kind of procedure (deterministic by default):
  - N: with a non-deterministic procedure.
- Complexity level:
  - LOG: logarithmic
  - P: polynomial (time)
  - EXP: exponential

Some complexity classes

$$\text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \text{EXPSPACE}$$

Examples of problems in various complexity classes

NLOGSPACE

2-SAT.

NP

3-SAT.

EXPTIME

Halting in k steps problem.

NEXPTIME

Determining whether two regular expressions ($|$, ., ^{2}) generate different languages

P

Determining if a number is prime.

PSPACE

Determining whether a regular expression generates every string over its alphabet.

EXPSPACE

Determining whether two regular expressions ($|$, ., ^{2}, *) generate different languages

A few last words on general complexity

Hardness

A problem P is C-hard for a complexity class C if any P' in C can be reduced to P "cheaply" (the reduction problem is in a class below C).

Completeness

A problem P is C-complete if it is C-hard and belongs to C.

Remark: P = NP?

A few last words on general complexity

Hardness

A problem P is C-hard for a complexity class C if any P' in C can be reduced to P "cheaply" (the reduction problem is in a class below C).

Completeness

A problem P is C-complete if it is C-hard and belongs to C.

Remark: P = NP?

Decision problems

In practice, in computational complexity theory, we focus on decision problems: problems for which the answer is always yes or no.

Decidability of planning

The decision problems considered

Decision problem 1: PLAN-EXISTENCE

Is there a solution to a planning problem?

Decision problem 2: PLAN-LENGTH

Is there a solution of length smaller than n to a planning problem?

The decision problems considered

Decision problem 1: PLAN-EXISTENCE

Is there a solution to a planning problem?

Decision problem 2: PLAN-LENGTH

Is there a solution of length smaller than n to a planning problem?

Decidability of planning

Function symbols?	PLAN-EXISTENCE	PLAN-LENGTH
no	decidable	decidable
yes	semi-decidable	decidable

Complexity of planning

Setting

Focus on classical planning

Both decision problems are decidable.

Setting

Focus on classical planning

Both decision problems are decidable.

Complexity depends of the representation

- Classical representation,
- set-theoretic representation,
- state-variable representation.

Setting

Focus on classical planning

Both decision problems are decidable.

Complexity depends of the representation

- Classical representation,
- set-theoretic representation,
- state-variable representation.

Complexity depends of the kind of preconditions and effects allowed

- Negative preconditions or not,
- negative effects or not.

Complexity results for the classical representation

Neg. effects?	Neg. precond.?	PLAN-EXISTENCE	PLAN-LENGTH
Yes	-	EXPSPACE-C.	NEXPTIME-C.
No	Yes	NEXPTIME-C.	NEXPTIME-C.
No	No	EXPTIME-C.	NEXPTIME-C.
No	No[1]	PSPACE-C.	PSPACE-C.

[1] and no operator has more than 1 precondition

Complexity results for the classical representation

Neg. effects?	Neg. precond.?	PLAN-EXISTENCE	PLAN-LENGTH
Yes	-	EXPSPACE-C.	NEXPTIME-C.
No	Yes	NEXPTIME-C.	NEXPTIME-C.
No	No	EXPTIME-C.	NEXPTIME-C.
No	No[1]	PSPACE-C.	PSPACE-C.

State-variable representation

Essentially the same as for classical representation.

[1] and no operator has more than 1 precondition

Complexity results for the set-theoretic representation

Remark

As explained before, the set-theoretic representation corresponds to the classical representation where everything is grounded:

- exponential blowup of the size of the representation,
- complexities look smaller (functions of the size of the input).

Neg. effects?	Neg. precond.?	PLAN-EXISTENCE	PLAN-LENGTH
Yes	-	PSPACE-C.	PSPACE-C.
No	Yes	NP-C.	NP-C.
No	No	P	NP-C.
No	No[2]	NLOGSPACE-C.	NP-C.

[2] and no operator has more than 1 precondition OR every operator with more than 1 precondition is the composition of other operators

What if we know more? (domain-dependent planning)

Classical representation with operators known in advance

Neg. effects?	Neg. precond.?	PLAN-EXISTENCE	PLAN-LENGTH
Yes	-	PSPACE	PSPACE
No	Yes	NP	NP
No	No	P	NP
No	No[a]	NLOGSPACE	NP

[a] and no operator has more than 1 precondition OR every operator with more than 1 precondition is the composition of other operators

Set-theoretic representation with operators known in advance

Everything can be done in constant time!

Summary on decidability and complexity

Classical planning + functions

Arbitrary computations can be encoded as planning problems:

- PLAN-EXISTENCE is semi-decidable,
- PLAN-LENGTH is decidable,

without functions, both decision problems are decidable.

Classical planning is a complex task

- PLAN-EXISTENCE is EXPSPACE-Complete,
- PLAN-LENGTH is NEXPTIME-Complete,

but these are worst-case results: domain-dependent planning is in general much easier.

Example of a proof on complexity

Sketch

Theorem (what we want to prove)

In the set-theoretic representation (with negative preconditions and negative effects), PLAN-EXISTENCE is PSPACE-Complete.

Notation: from now on, PLAN-EXISTENCE means PLAN-EXISTENCE in the setting of the above theorem.

Sketch of the proof

- Prove that PLAN-EXISTENCE is in PSPACE,
  - using a counting argument.
- Prove that PLAN-EXISTENCE is PSPACE-Hard,
  - by reduction of another PSPACE-Hard problem.

PLAN-EXISTENCE is in PSPACE

Note n the number of propositions in the planning problem.

Bound on plan length

Show that there exists a plan iff there exists a plan of length 2^n or less.

PLAN-EXISTENCE is in PSPACE

Note n the number of propositions in the planning problem.

Bound on plan length

Show that there exists a plan iff there exists a plan of length 2^n or less.

Non-deterministic choices

Deduce from the previous statement the maximum number of non-deterministic choices that have to be performed in order to find a plan.

PLAN-EXISTENCE is in PSPACE

Note n the number of propositions in the planning problem.

Bound on plan length

Show that there exists a plan iff there exists a plan of length 2^n or less.

Non-deterministic choices

Deduce from the previous statement the maximum number of non-deterministic choices that have to be performed in order to find a plan.

Conclusion

We get that PLAN-EXISTENCE is in NPSPACE. Moreover, NPSPACE = PSPACE. Thus, PLAN-EXISTENCE is in PSPACE.

A little interlude: Turing machines

Turing machines: informal definition

A Turing machine is constituted of:

- a *tape*, with sequentially ordered cells, each one able to contain one symbol from a finite alphabet,
- a *head*, that can read/write symbols on the tape and move to the left/to the right (of one cell at a time),
- a *state register*, that stores the state of the machine (finitely many states), initialized with a particular initial state,
- a *table of instructions*, that given the current state and the current symbol read by the tape, says what should be done:
  1. write a new symbol,
  2. move the head,
  3. change the state.

A little interlude, continued

Turing machines: input acceptance

A Turing machine accepts an input if:

- it reaches an acceptance state from this input, and
- it can no longer move from this state.

Polynomially bounded Turing machines

The space of a Turing machine is said to be polynomially bounded if the maximum number of cells used on the tape is bounded by a polynomial of the size of the rest of the machine (states, symbols).

A little interlude, continued

Turing machines: input acceptance

A Turing machine accepts an input if:

- it reaches an acceptance state from this input, and
- it can no longer move from this state.

Polynomially bounded Turing machines

The space of a Turing machine is said to be polynomially bounded if the maximum number of cells used on the tape is bounded by a polynomial of the size of the rest of the machine (states, symbols).

A PSPACE-Hard problem $\mathcal{P}$

Decide if a Turing machine with a polynomially bounded space accepts a given input.

PLAN-EXISTENCE is PSPACE-Hard

Reduction

Propose a representation of a Turing machine as a planning problem.

PLAN-EXISTENCE is PSPACE-Hard

Reduction

Propose a representation of a Turing machine as a planning problem.

Complexity of the reduction

Show that the complexity of the above reduction is polynomial when the Turing machine has a polynomially bounded space.

PLAN-EXISTENCE is PSPACE-Hard

Reduction

Propose a representation of a Turing machine as a planning problem.

Complexity of the reduction

Show that the complexity of the above reduction is polynomial when the Turing machine has a polynomially bounded space.

Conclusion

We have found a polynomial reduction of $\mathcal{P}$ as PLAN-EXISTENCE. Thus, by definition, PLAN-EXISTENCE is PSPACE-Hard.

General conclusion

PLAN-EXISTENCE is PSPACE-Hard and is in PSPACE. Thus PLAN-EXISTENCE is PSPACE-Complete.

Outline

A few words on search algorithms

Why do we study them?

Most of the planners are based on search algorithms.

Graph terminology

One searches among *nodes* linked by *edges*.

Search space may differ between planners

- State-space planning: each node is a state, a plan is a path through the space,
- plan-space planning: each node is a set of operators associated with some constraints, a plan is a fully constrained node.

A few words on search algorithms

Why do we study them?

Most of the planners are based on search algorithms.

Graph terminology

One searches among *nodes* linked by *edges*.

Search space may differ between planners

- State-space planning: each node is a state, a plan is a path through the space,
- plan-space planning: each node is a set of operators associated with some constraints, a plan is a fully constrained node.

$\Rightarrow$ **Focus on state-space planning.**

Non-deterministic generic forward search algorithm

The algorithm

procedure SEARCH(O,s_0,g)
    $s \leftarrow s_0$
    $\pi \leftarrow \varepsilon$
    loop
        if s satisfies g **then return** π
        end if
        $E \leftarrow \{a \mid a$ is a ground instance of an operator in O,
                and $\mathit{precond}(a)$ is true in $s\}$
        if $E = \emptyset$ **then return** failure
        end if
        non-deterministically choose an action $a \in E$
        $s \leftarrow \gamma(s, a)$
        $\pi \leftarrow \pi.a$
    end loop
end procedure

Properties of the non-deterministic forward search

Soundness

Any plan returned by the search is guaranteed to be a solution to the planning problem (O, s_0, g).

Completeness

If a solution to the planning problem (O, s_0, g) exists, at least one execution of the search will return a plan.

Properties of the non-deterministic forward search

Soundness

Any plan returned by the search is guaranteed to be a solution to the planning problem (O, s_0, g).

Completeness

If a solution to the planning problem (O, s_0, g) exists, at least one execution of the search will return a plan.

Main limitation

The branching factor (number of actions possible at a given step) can be huge, so deterministic implementations will waste time trying irrelevant actions. $\Rightarrow$ Use heuristics to prune.

Deterministic implementations of forward search

Generic deterministic forward search

procedure SEARCH(O, s_0, g)
  $nexts \leftarrow \{s_0\}$; $pred[s_0] \leftarrow nil$
  while $nexts \neq \emptyset$ **do**
    choose s in $nexts$
    if s satisfies g **then return** π built from $pred[s]$
    end if
    $nexts \leftarrow nexts \setminus \{s\}$
    $E \leftarrow \{a \mid a$ is a ground instance of an operator in O,
        and $precond(a)$ is true in $s\}$
    for all $a \in E$ **do**
      if $pred[\gamma(s, a)]$ not yet defined **then**
        $nexts \leftarrow nexts \cup \{\gamma(s, a)\}$; $pred[\gamma(s, a)] \leftarrow (s, a)$
      end if
    end for
  end while
end procedure

Specific deterministic forward searches

Depending on the data-structure used for *nexts* one obtains different search algorithms.

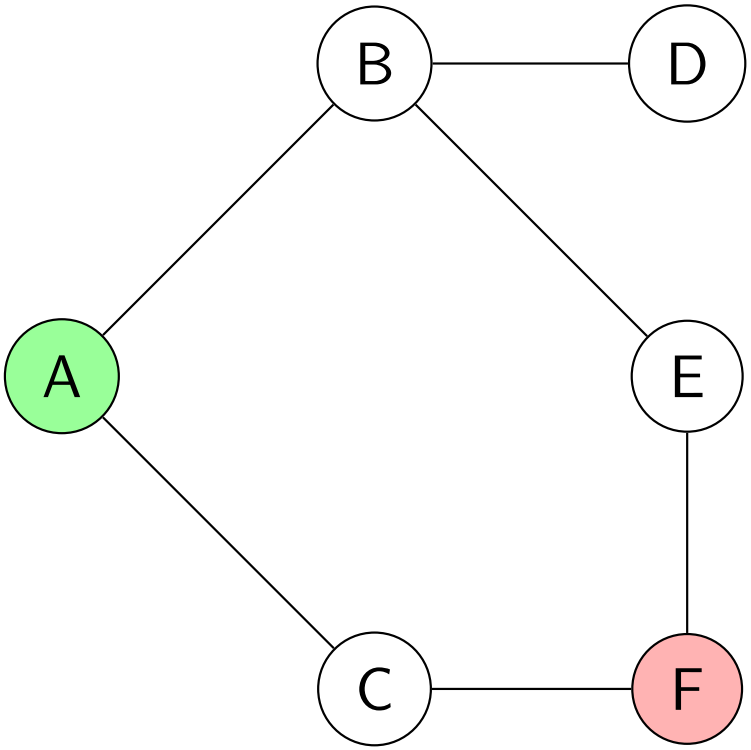
When *nexts* is a FIFO data-structure

Breadth-first search (BFS).

When *nexts* is a LIFO data-structure

Depth-first search (DFS).

Examples



When several nodes have to be considered, we will do it using lexicographical order.

DFS

Apply the SEARCH procedure to the graph on the left using a LIFO data-structure for *nexts*.

BFS

Apply the SEARCH procedure to the graph on the left using a FIFO data-structure for *nexts*.

Forward search with action costs

Planning with action costs

Principle

Associate a cost (in $\mathbb{N}$) to each operator/action in a planning problem.

$$c(o), c(a)$$

Cost of a plan

Associate a cost to each plan: the sum of the costs of the actions it uses.

$$c(\pi) = c(a_1 \dots a_n) = \sum_{i=1}^{n} c(a_i)$$

Goal

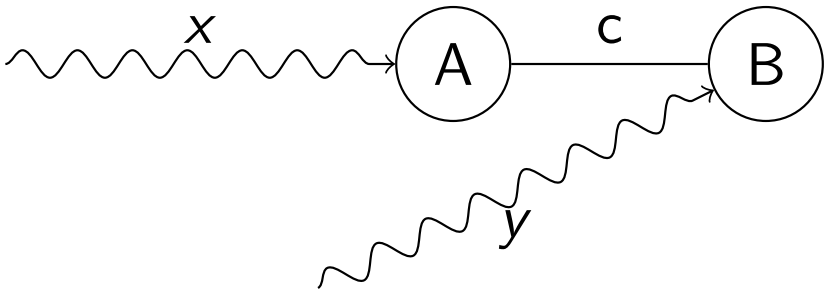
Find a plan with small cost (ideally minimal cost).

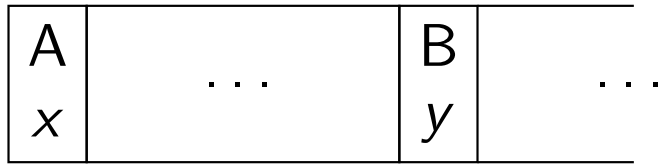
$$\pi_m = argmin_\pi c(\pi)$$

Forward search with action costs: Dijkstra's algorithm

Idea

- Standard forward search,
- priority queue for *nexts*,
- cost of a node: best cost known at the moment to reach it,
- if cost for reaching s is reduced: update it in nexts.



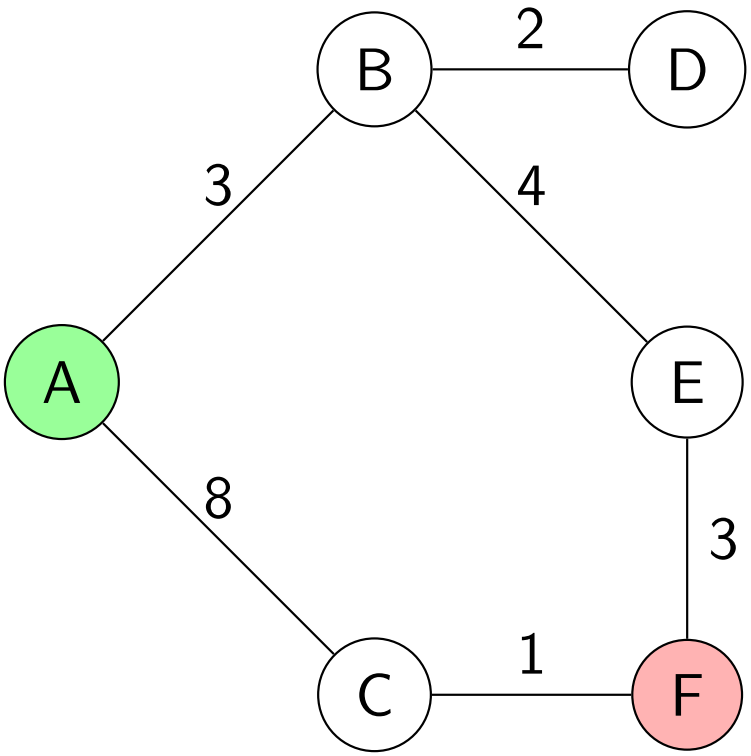


$$x \leq y$$
$$x + c < y$$

Dijkstra's algorithm

procedure DIJKSTRA(O, s_0, g, c)
  $nexts \leftarrow \{s_0\}$; $pred[s_0] \leftarrow nil$; $f[s_0] \leftarrow 0$; $\forall s \neq s_0, f[s] \leftarrow +\infty$
  while $nexts \neq \emptyset$ **do**
    take s in $nexts$ which minimizes $f(s)$
    if s satisfies g **then return** π built from $pred[s]$
    end if
    $E \leftarrow \{a \mid a$ is a ground instance of an operator in O,
        and $precond(a)$ is true in $s\}$
    for all $a \in E$ **do**
      if $f(\gamma(s,a)) > f(s) + c(a)$ **then**
        $nexts \leftarrow nexts \cup \{\gamma(s,a)\}$; $pred[\gamma(s,a)] \leftarrow (s,a)$
        $f[\gamma(s,a)] \leftarrow f(s) + c(a)$
      end if
    end for
  end while
end procedure

Example



DFS

Apply the DIJKSTRA procedure to the graph on the left.

Proof of Dijkstra's algorithm

Termination

Prove that Dijkstra's algorithm always terminates.

Proof of Dijkstra's algorithm

Termination

Prove that Dijkstra's algorithm always terminates.

Completeness

Prove that Dijkstra's algorithm returns a plan if and only if there exists a solution to (O, s_0, g, c).

Proof of Dijkstra's algorithm

Termination

Prove that Dijkstra's algorithm always terminates.

Completeness

Prove that Dijkstra's algorithm returns a plan if and only if there exists a solution to (O, s_0, g, c).

Soundness

Prove that if Dijkstra's algorithm outputs a plan it is:

- a valid plan,
- a minimum cost plan.

Heuristics

Starting point

Dijkstra's algorithm does not use knowledge about the goal to find minimum cost paths.

Example

Consider the previous example, when E and C are in the queue one could already know that C should be used to reach the goal.

Heuristics, principle

A heuristic is a function h that, for any state s, gives an estimate of the cost to reach the goal from s.

Heuristics

Starting point

Dijkstra's algorithm does not use knowledge about the goal to find minimum cost paths.

Example

Consider the previous example, when E and C are in the queue one could already know that C should be used to reach the goal.

Heuristics, principle

A heuristic is a function h that, for any state s, gives an estimate of the cost to reach the goal from s.

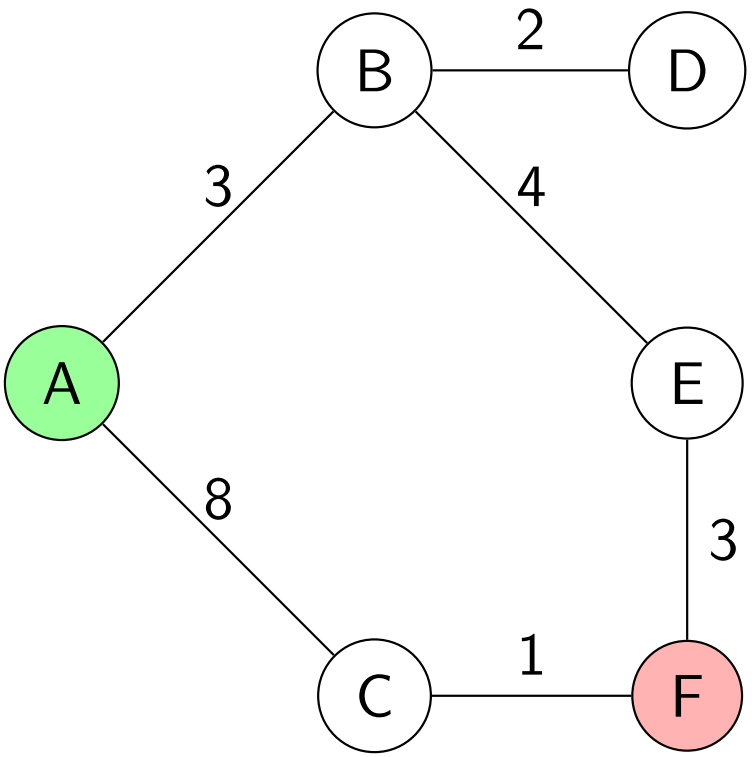
Heuristics in Dijkstra's algorithm: A* algorithm

Use $f(s) + h(s)$ to order states in the queue *nexts*.

A* algorithm

procedure $\textsc{Astar}(O, s_0, g, c)$
  $nexts \leftarrow \{s_0\}$; $pred[s_0] \leftarrow nil$; $f[s_0] \leftarrow 0$; $\forall s \neq s_0, f[s] \leftarrow +\infty$
  while $nexts \neq \emptyset$ **do**
    take s in $nexts$ which minimizes $f(s) + h(s)$
    if s satisfies g **then return** π built from $pred[s]$
    end if
    $E \leftarrow \{a \mid a$ is a ground instance of an operator in O,
        and $precond(a)$ is true in $s\}$
    for all $a \in E$ **do**
      if $f(\gamma(s,a)) > f(s) + c(a)$ **then**
        $nexts \leftarrow nexts \cup \{\gamma(s,a)\}$; $pred[\gamma(s,a)] \leftarrow (s,a)$
        $f[\gamma(s,a)] \leftarrow f(s) + c(a)$
      end if
    end for
  end while
end procedure

Example



DFS

Apply the ASTAR procedure to the graph on the left using the following heuristics:

- exact cost,
- exact cost + 5,
- 6 times the number of edges
- always 0.

Proof of A* algorithm

Termination (independent of the heuristic)

Prove that A* algorithm always terminates.

Proof of A* algorithm

Termination (independent of the heuristic)

Prove that A* algorithm always terminates.

Completeness (independent of the heuristic)

Prove that A* algorithm returns a plan if and only if there exists a solution to (O, s_0, g, c).

Proof of A* algorithm

Termination (independent of the heuristic)

Prove that A* algorithm always terminates.

Completeness (independent of the heuristic)

Prove that A* algorithm returns a plan if and only if there exists a solution to (O, s_0, g, c).

Soundness part 1 (independent of the heuristic)

Prove that if A* algorithm outputs a plan it is:

- a valid plan.

Proof of A* algorithm, continued

Admissible heuristic

A heuristic h is said to be admissible if for any state s, $h(s)$ underestimates the cost to reach a goal from s.

Soundness part 2 (only for admissible heuristics)

Prove that if one uses an admissible heuristic and if A* algorithm outputs a plan it is:

- a minimum cost plan.

(Admissible) heuristics, remarks

Classical example

Distance as the crow flies. Not always a good heuristic:

- may be inaccurate (twisty roads),
- may cost a lot to compute (depends of the problem representation).

In practice

A good heuristic must be:

- as accurate as possible,
- easy to compute.

Backward search

The idea of backward search

Search from the goal

Start from the goal and compute inverse state transitions

Relevance of action a for a goal g

- $g \cap \mathit{effects}(a) \neq \emptyset$
  - (a makes at least one of g's literals true),
- $g^+ \cap \mathit{effects}^-(a) = \emptyset$ and $g^- \cap \mathit{effects}^+(a) = \emptyset$
  - (a does not make any of g's literals false).

Inverse state transitions

If a is relevant for g, then:

$$\gamma^{-1}(g, a) = (g \setminus \mathit{effects}(a)) \cup \mathit{precond}(a)$$

(otherwise $\gamma^{-1}(g, a)$ is undefined)

A non-deterministic backward search algorithm

Intuition on backward search

From a (set of sub)goal g, compute the new set of subgoals $\gamma^{-1}(g, a)$.

procedure BACKWARDSEARCH(O,s_0,g)
  $\pi \leftarrow \varepsilon$
  loop
    if s_0 satisfies g **then return** π
    end if
    $A \leftarrow \{a \mid a$ is a ground instance of an operator in O,
      and $\gamma^{-1}(g, a)$ is defined$\}$
    if $A = \emptyset$ **then return** failure
    end if
    non-deterministically choose an action $a \in A$
    $g \leftarrow \gamma^{-1}(g, a)$; $\pi \leftarrow a.\pi$
  end loop
end procedure

Interest of backward search

Branching factor

Not smaller than for forward search in general.

Lifting

It is possible, with backward search to partially ground operators. This can drastically reduce the branching factor.

Lifting example: blocks world

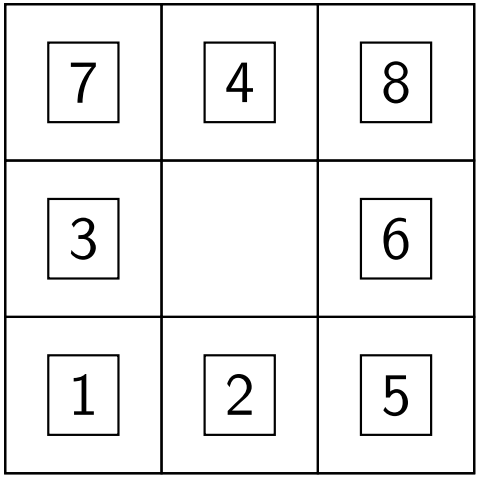
Consider that $g = \mathit{holding}(b_1)$:

- reachable with $\mathit{pickup}(b_1)$, $\mathit{unstack}(b_1, b_2)$, $\mathit{unstack}(b_1, b_3)$, $\mathit{unstack}(b_1, b_4)$, etc (without lifting), or
- reachable with $\mathit{pickup}(b_1)$, $\mathit{unstack}(b_1, y)$ where *unstack* is partially grounded (with lifting).

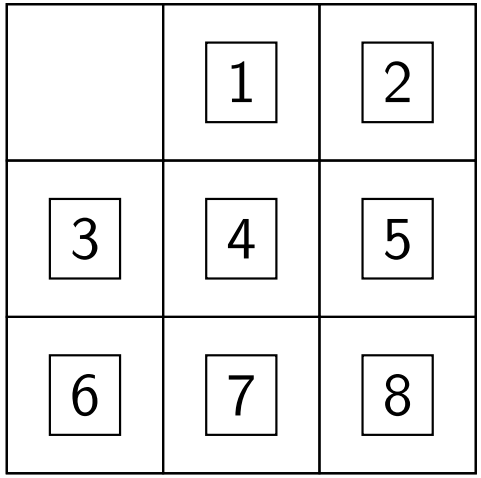
Outline

Domain specific heuristics

Standard example: the 8-puzzle



Random initial position



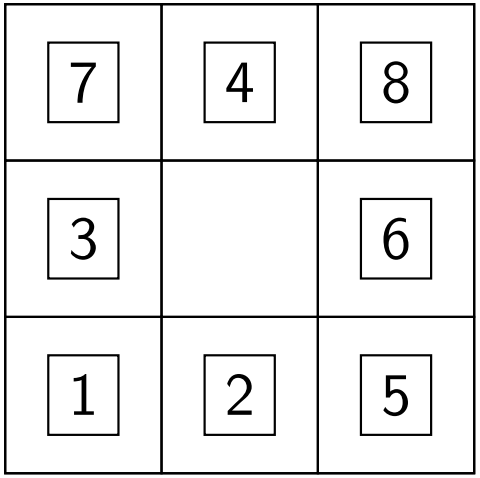
Goal position

About 8-puzzle

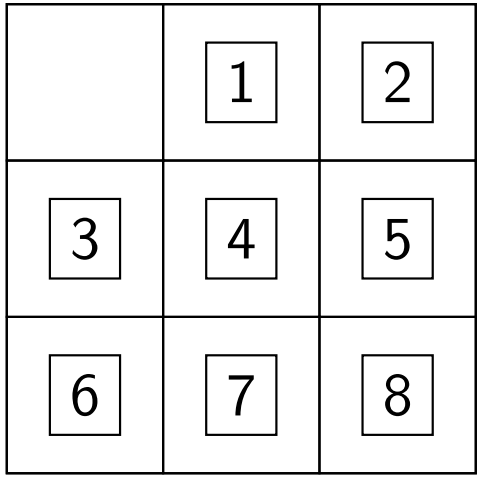
- Maximal branching: 4,
- States: 362880 (9!).

15-puzzle? 24-puzzle?

Standard example: the 8-puzzle



Random initial position



Goal position

About 8-puzzle

- Maximal branching: 4,
- States: 362880 (9!).

15-puzzle? 24-puzzle?

Heuristics

- Number of misplaced tiles (admissible?),
- Sum of distances of tiles from their goal position (admissible?).

Standard example: shortest road

Many roads in the world, travelling salesman problem, as the crow flies.

A few words on domain independent heuristics

Required properties of heuristics

- Computable at each state from the input of the planner (model of a planning problem),
- cheap to compute (time/memory for searching without heuristic vs. time/memory for searching with heuristic + time/memory for computing heuristic),
- accurate (bad admissible heuristic vs. precise non-admissible heuristic).

General idea for computing and using a heuristic

Try to automatically get domain-specific knowledge of the problem considered. Use it to drive the search.

Delete relaxation heuristics

Delete relaxation heuristics: principle

Delete relaxation heuristic computation

Remove all the negative effects of the operators to build a relaxed problem. Find exact costs in this relaxed problem to use as heuristic.

Abstraction

Compute the relaxed problem for blocks world.

Heuristic

Compute the value of the delete relaxation heuristic for the initial state of blocks world.

Admissible heuristic

Justify that this heuristic is admissible (1) for blocks world, (2) in general.

Delete relaxation heuristics: estimates

Computing delete relaxation heuristics is complex

PLAN-EXISTENCE is NP-Complete in relaxed problems.

A possible estimate of the delete relaxation heuristic at state s

- For a predicate $p \in G$, $g_s(p)$ estimates the cost to achieve p from s:
  - initialize $g_s(p)$ to 0 for each $p \in s$ and to $+\infty$ for any other p,
  - while possible, choose o s.t. $\mathit{precond}(o) \subseteq s$, $\forall p \in \mathit{effects}^+(o)$ update $g_s(p)$ to $\min(g_s(p), c(o) + g_s(\mathit{precond}(o)))$ and add p to s.
- For a set P of predicates, $g_s(P)$ is:
  - $\sum_{p \in P} g_s(p)$ (leads to non-admissible heuristics), or
  - $\max_{p \in P} g_s(p)$.
- Use $g_s(G)$ as an estimate (called h_{add} or h_{max}) for the heuristic.

Back to previous example

Compute this estimate at the initial state of blocks world.

More on delete relaxation heuristics

B. Bonet and H. Geffner. *Planning as Heuristic Search*. Artificial Intelligence, 2001.

Critical path heuristics

Critical path heuristics: principle

A family of admissible heuristics

h^1, h^2, ...

- $h^m(s) \geq h^{m-1}(s)$: more and more accurate,
- computing h^m is exponential in m: more and more difficult to build

Definition

$h^m(s)$ is the maximum cost over all subsets of m predicates of achieving one of them from s.

Remark

h^1 is exactly the h_{max} heuristic used as an estimate of the delete relaxation heuristic.

Back to previous example

Compute h^2 at the initial state of blocks world.

More on critical path heuristics

P. Haslum. *Admissible Heuristics for Automated Planning*. Phd thesis, 2006. Chapter 3.

Abstraction heuristics

Abstraction heuristics: principle

Abstracting a planning problem

Map each state s to an abstract state $\alpha(s)$. In general, the function alpha will map several states to the same abstract state. This induces an abstract planning problem:

- initial state: $\alpha(s_0)$,
- goal: $\alpha(G) = \{\alpha(s) \ : \ s \supseteq G\}$,
- state-transitions: if $\gamma(s, a) = s'$ then $\gamma_\alpha(\alpha(s), a) \supseteq \alpha(s')$.

Abstracting a problem can bring non-determinism.

Heuristic computation

Optimal costs in the abstract problem are used as cost estimates.

Difficulty

Choose abstractions so that heuristics are fast to compute and accurate.

Abstraction heuristics: merge and shrink

Principle

Starting from atomic abstractions, iteratively build an abstraction heuristic by applying two operations:

- merge, to reduce the number of different abstractions by combining them, and
- shrink to reduce the size of individual abstractions.

Atomic abstractions

One abstraction per predicate p: $\alpha(s) = \alpha(s')$ if and only if

- $p \in s \cap s'$, or
- $p \notin s \cup s'$.

Abstraction heuristics: merge and shrink continued

Merge (combination of abstractions)

Synchronized product of the abstractions $(S_i, T_i, s_{0,i}, G_i)$ for $i \in \{1, 2\}$:

- $S_1 \times S_2$ the states,
- $((s_1, s_2), a, (s'_1, s'_2)) \in T$ iff $(s_1, a, s'_1) \in T_1$ and $(s_2, a, s'_2) \in T_2$,
- $(s_{0,1}, s_{0,2})$ the initial state,
- $G_1 \times G_2$ the goal states.

Shrink (size reduction of abstractions)

- To reduce the size by one: merge two states,
- to reduce the size by M: apply the above M times.

The choice of states to merge can be done for example: arbitrarily, or by preserving costs (value of g, h, or f).

Abstraction heuristics: merge and shrink algorithm

procedure MergeAndShrink((P, N))
    $absSet \leftarrow \{\alpha \ : \ \alpha$ is an atomic abstraction$\}$
    select abs in $absSet$; $absSet \leftarrow absSet \setminus \{abs\}$
    while $absSet \neq \emptyset$ **do**
        select abs' in $absSet$; $absSet \leftarrow absSet \setminus \{abs'\}$
        Shrink abs to size $N/size(abs')$
        Merge abs and abs' as abs
    end whilereturn abs
end procedure

Abstraction heuristics: merge and shrink algorithm

procedure MergeAndShrink((P, N))
  $absSet \leftarrow \{\alpha \ : \ \alpha \text{ is an atomic abstraction}\}$
  select abs in $absSet$; $absSet \leftarrow absSet \setminus \{abs\}$
  while $absSet \neq \emptyset$ **do**
    select abs' in $absSet$; $absSet \leftarrow absSet \setminus \{abs'\}$
    Shrink abs to size $N/size(abs')$
    Merge abs and abs' as abs
  end whilereturn abs
end procedure

Back to previous example

Compute the value of a merge and shrink heuristic at the initial state of blocks world.

Abstraction heuristics: pattern databases

Starting point: groups of predicates

Choose a partition of the predicates so that predicates in a given set are mutually exclusives:

- $P_1, P_2, \ldots, P_n$ sets of predicates,
- $P_1 \cup P_2 \cup \cdots \cup P_n = P \cup \{true\}$ (P the set of all predicates),
- $\forall i, j, P_i \cap P_j = \emptyset$ (or $= \{true\}$),
- $\forall i, \forall p_1, p_2 \in P_i$, one has that p_1 and p_2 are never true together in a reachable state,
- *true* is a special element expressing the fact that no predicate from P_i may be true in a given state.

Remark: this partition is not unique.

Abstraction heuristics: pattern databases continued

Pattern databases

- For each element P_i of a partition, get an abstraction of the planning problem by projecting on the predicates of P_i.
- Shortest paths to goal in this abstraction directly give an admissible heuristic h_i.
- Under certain conditions, the heuristics $h_1, h_2, \ldots, h_n$ can be combined (by summing them) into a single admissible heuristic.

Abstraction heuristics: pattern databases continued

Pattern databases

- For each element P_i of a partition, get an abstraction of the planning problem by projecting on the predicates of P_i.
- Shortest paths to goal in this abstraction directly give an admissible heuristic h_i.
- Under certain conditions, the heuristics $h_1, h_2, \ldots, h_n$ can be combined (by summing them) into a single admissible heuristic.

Independent abstraction set

A set $I = \{P_{k_1}, P_{k_2}, \ldots, P_{k_m}\}$ of groups is an independent abstraction set if no operator affects both predicates in groups in I and in groups not in I. A partition of the groups into independent abstraction sets yields heuristics that can be safely combined.

More on abstraction heuristics

Merge and shrink

M. Helmert, P. Haslum, J. Hoffmann. *Flexible Abstraction Heuristics for Optimal Sequential Planning*. ICAPS 2007.

Pattern databases

S. Edelkamp. *Planning with Pattern Databases*. ICAPS 2001.

Landmarks heuristics

Landmarks heuristics

Landmarks

We call landmarks propositional formulas that must necessarily be true at some point in any plan.

Ordered landmarks

Landmarks can be ordered if they have to be true in a certain order in any plan.

Path dependent heuristic

The number of landmarks yet to achieve given a partial plan (a prefix of a plan) can be used as a heuristic. Notice that it does not depend on the current state only, but also on the way it was reached.

More on landmarks heuristics

Non-admissible heuristics

S. Richter, M. Helmert, and M. Westphal. *Landmarks Revisited*. AAAI, 2008.

Admissible heuristics

E. Karpas and C. Domshlak. *Cost-Optimal Planning With Landmarks*. IJCAI, 2009.

Outline

Partial order techniques

The basic idea

Plan (as we defined it)

A sequence of actions, i.e. a totally ordered set of actions.

Concurrent actions

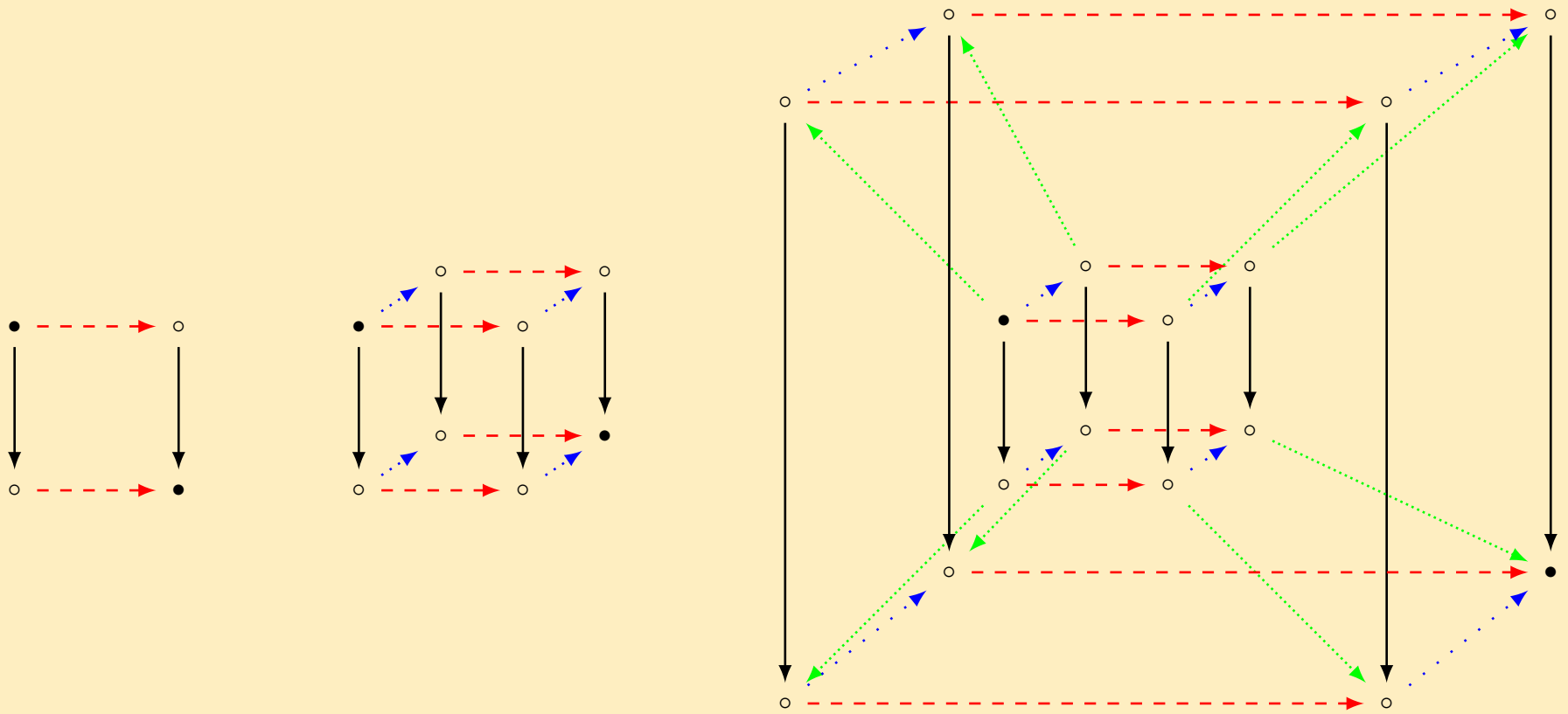
- Independant,
- executable in any order from any state which fulfils their preconditions,
- or even executable in parallel

Example of concurrent actions

- $(precond_1 \cup effects_1) \cap (precond_2 \cup effects_2) = \emptyset$
- $\forall i, j, (precond_i \cup effects_i) \cap effects_j = \emptyset$

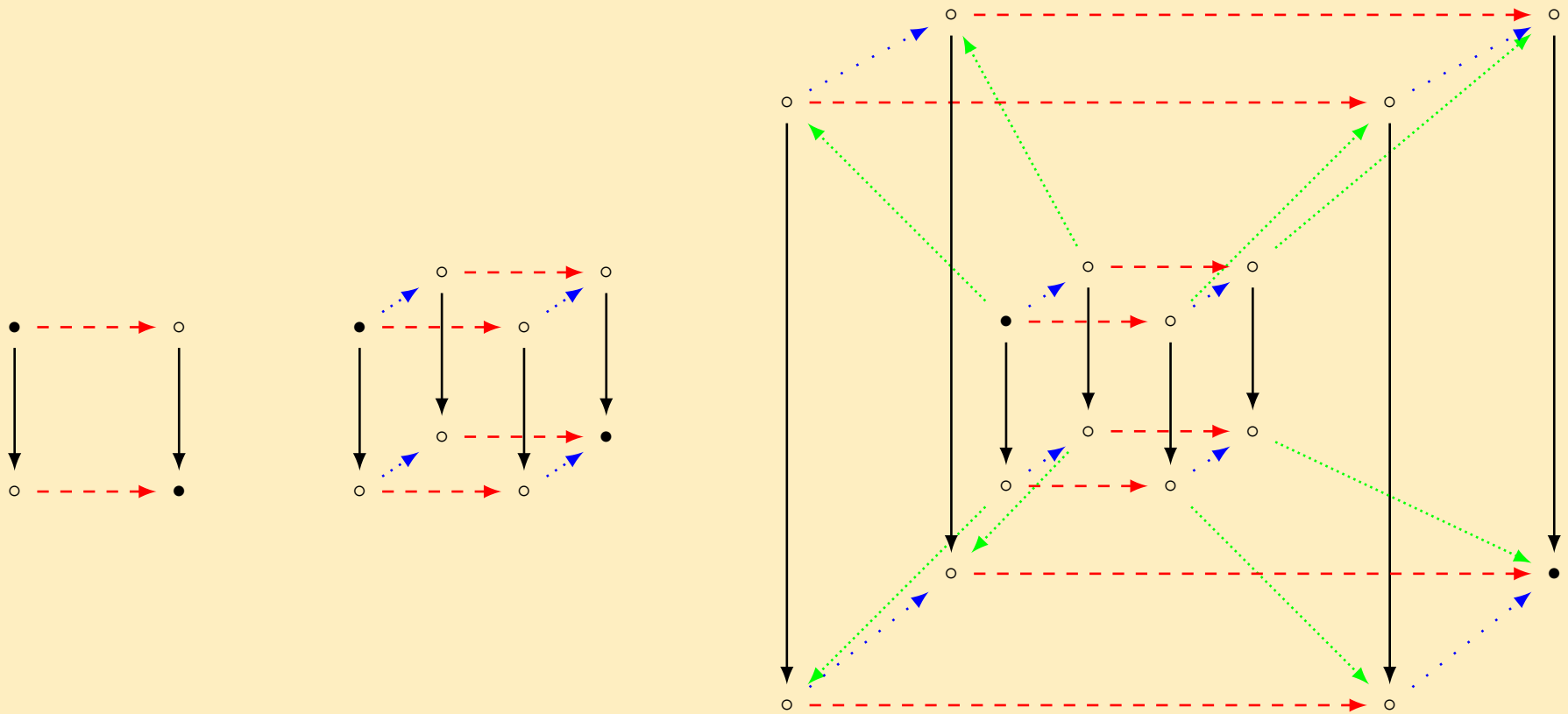
The basic idea continued

2, 3, 4 concurrent actions in a sequential plan



The basic idea continued

2, 3, 4 concurrent actions in a sequential plan



⇒ **Represent plans as partial orders**

Two partial order approaches to planning

Several approaches based on partial orders exist to solve planning problems. In this part of the course we focus on two of them.

Petri net unfolding[3]

A planning problem can be represented as a Petri net. Unfolding techniques allow to search for plans.

Graphplan[4]

Planning problems can be represented as special graphs called *planning graphs* where concurrency is taken into account. Plans can be found directly in these graphs.

[3] S. Hickmott, J. Rintanen, S. Thiébaux, and L. White. *Planning Via Petri Net Unfolding*. IJCAI 2007.

[4] A. L. Blum and M. L. Furst. *Fast Planning Through Planning Graph Analysis.* Artificial Intelligence 1995.

Planning with Petri nets

Petri nets: formalism

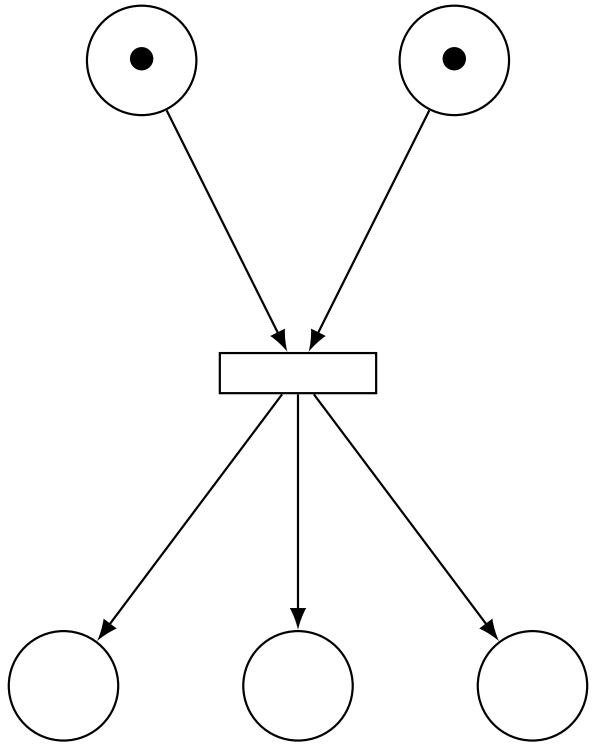
We use a restricted class of Petri nets called safe PT-nets.

Syntax

A *PT-net* is a tuple (P, T, F, M_0) where P and T are disjoint finite sets of places and transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, and $M_0 \subseteq P$ is an initial marking.

Semantics

In a PT-net, given a marking M a transition t is *fireable* if $\forall p \in {}^\bullet t$ one has $p \in M$. In this case, firing t leads to the marking $M' = (M \setminus {}^\bullet t) \cup t^\bullet$. With ${}^\bullet t = \{p \ : \ (p, t) \in T\}$ the preset of t and $t^\bullet = \{p \ : \ (t, p) \in T\}$ the postset of t.



From planning problems to Petri nets: safe operators

Initial remark

Transitions of Petri nets will represent actions of planning problems. Places will represent atoms. A net is safe if any token added by a transition is added in an empty place.
$\Rightarrow$ An action should never add an atom if it is already there.

Safe operator

An operator is safe if $\overline{effects} \subseteq precond$ (all effects appear negatively in the precondition).

Ensuring that operators are safe

For each operator create several variants of it, that are all safe and all together are equivalent to the original operator.

From planning problems to Petri nets: safe operators 2

Making safe operators, formally

Replace each operator $o = (p, e)$ with $2^{e \setminus \overline{p}}$ operators, defined from the subsets e' of $e \setminus \overline{p}$.

- New operator precondition: $p \cup \overline{e'} \cup (e \setminus \overline{p}) \setminus e'$.
- New operator effects: $e' \cup e \cap \overline{p}$.

Idea: the operator built from e' corresponds to the case where o modifies exactly the atoms from e' (everything other atom is left unchanged by o).

Example

The operator with $p = \{a, \neg b, c\}$ and $e = \{\neg a, b, d, \neg e\}$ is replaced by:

From planning problems to Petri nets: safe operators 2

Making safe operators, formally

Replace each operator $o = (p, e)$ with $2^{e \setminus \overline{p}}$ operators, defined from the subsets e' of $e \setminus \overline{p}$.

- New operator precondition: $p \cup \overline{e'} \cup (e \setminus \overline{p}) \setminus e'$.
- New operator effects: $e' \cup e \cap \overline{p}$.

Idea: the operator built from e' corresponds to the case where o modifies exactly the atoms from e' (everything other atom is left unchanged by o).

Example

The operator with $p = \{a, \neg b, c\}$ and $e = \{\neg a, b, d, \neg e\}$ is replaced by:

- $p = \{a, \neg b, c, d, \neg e\}$, $e = \{\neg a, b\}$,
- $p = \{a, \neg b, c, \neg d, \neg e\}$, $e = \{\neg a, b, d\}$,
- $p = \{a, \neg b, c, d, e\}$, $e = \{\neg a, b, \neg e\}$, and
- $p = \{a, \neg b, c, \neg d, e\}$, $e = \{\neg a, b, d, \neg e\}$.

From planning problems to PN: negative preconditions

Initial remark

In Petri nets there are no transitions with negative preconditions.
Rightarrow An action should never have negative preconditions.

Emulating negative preconditions

Each atom is duplicated: a and *nota*. Then, each operator is modified to ensure that a is true exactly when *nota* is false. Each $\neg a$ is replaced by *nota* in preconditions.

Example

Eliminate negative preconditions from the four operators of the previous example.

From planning problems to PN

Principle of the mapping

Simply remark that safe operators with no negative preconditions directly correspond to transitions in Petri nets:

- places : new atoms (a and $nota$),
- transitions : new operators ($o = (p, e)$),
- flow relation : $\{(a, o) \ : \ a \in p\} \cup \{(o, a) \ : \ a \in e \vee (a \in p \wedge \neg a \notin e)\}$,
- initial marking : $a \in M_0$ iff $a \in s_0$, $nota \in M_0$ iff $a \notin s_0$,

Example

Build the (sub) Petri net corresponding to the operator $o = (p, e)$ with $p = \{a, \neg b, c\}$ and $e = \{\neg a, b, d, \neg e\}$.

Solving planning problems represented as PN

Unfolding

Any reachability technique in Petri nets can be used to solve the planning problems from the Petri nets representing them. For example, *unfolding*.

Unfolding and heuristics

Unfolding can be directed by heuristics, exactly as Dijkstra's algorithm is directed to give A* algorithm[5].

[5] B. Bonet, P. Haslum, S. Hickmott, and S. Thiébaux. *Directed Unfolding of Petri Nets*. Transactions on Petri Nets and other Models of Concurrency, 2008.

Graphplan

Planning graph

Definition

Given a planning problem (A, O, I, G), a *planning graph* is a graph (V, E) such that:

- $V = V_A \cup V_0$ with:
  - $V_A \subseteq A \times \mathbb{N}_+^*$ a set of proposition vertices, and
  - $V_O \subseteq O \times \mathbb{N}_+^*$ a set of action vertices, and
- $E = E_{pre} \cup E_{del} \cup E_{add}$ with:
  - $E_{pre} \subseteq \{((a, i), (o, i)) \ : \ a \in \mathit{precond}(o)\}$ a set of precondition edges,
  - $E_{del} \subseteq \{((o, i), (a, i + 1) \ : \ a \in \mathit{effects}^-(o))$ a set of delete edges, and
  - $E_{add} \subseteq \{((o, i), (a, i + 1)) \ : \ a \in \mathit{effects}^+(o)\}$ a set of add edges.

Example

Draw one possible planning graph for the blocks world example.

Graphplan

Preliminary remark

From now on we consider that for each atom a of any planning problem there is a special operator $noop_a$ with precondition $\{a\}$ and effect $\{a\}$. Adding this to a planning problem does not change it.

General principle of Graphplan

- Iteratively build a particular planning graph.
- Start from a representation of the initial state of a planning problem.
- Do this level by level.
- After building each level, search for a plan in the planning graph.

Graphplan: iterative planning graph construction

Building proposition level 1

For each atom $a \in I$, add a vertex $(a, 1)$. None of them are *mutually exclusive*.

Building action level i

For each operator o so that $\forall a, b \in \mathit{precond}(o)$:

- $(a, i) \in V$ (a appears at proposition level i),
- (a, i) and (b, i) are not mutually exclusive,

add a vertex (o, i) and add an arc $((a, i), (o, i))$ for each $a \in \mathit{precond}(o)$.

Mutually exclusive vertices at action level i

Two vertices (o_1, i) and (o_2, i) are said to be mutually exclusive if o_1 deletes a precondition or positive effect of o_2.

Graphplan: iterative planning graph construction

Building proposition level $i+1$

For each atom a so that there is $(o, i) \in V$ (o appears at action level i) with $a \in \mathit{effects}^+(o)$, add a vertex $(a, i+1)$ and an arc $((o, i), (a, i+1))$ for each such o.

Mutually exclusive vertices at proposition level $i+1$

Two vertices $(a_1, i+1)$ and $(a_2, i+1)$ are not mutually exclusive if:

- there exist (o_1, i) and (o_2, i) that are not mutually exclusive and so that o_1 adds a_1, o_2 adds a_2, or
- there exist (o, i) so that o adds both a_1 and a_2.

Else, $(a_1, i+1)$ and $(a_2, i+1)$ are mutually exclusive.

Example

Draw the first levels of the planning graph built by graphplan for the blocks world example.

Graphplan: plan search

When do we search for a plan?

After the construction of each proposition level.

Backward search from proposition level i

1. Is the goal achievable at proposition level i? (by non-mutually-exclusive vertices)
2. If yes, choose non-mutually exclusive operators at action level $i-1$ that achieve the goal together.

4. Set the union of their preconditions as a new goal for proposition level $i-1$ and repeat from the first step. If $i-1=1$, a plan has been found.

Graphplan: plan search

When do we search for a plan?

After the construction of each proposition level.

Backward search from proposition level i

1. Is the goal achievable at proposition level i? (by non-mutually-exclusive vertices)
2. If yes, choose non-mutually exclusive operators at action level $i-1$ that achieve the goal together.
3. If no, backtrack to level i and choose a different set of operators at step 2 (if this is not possible backtrack to level $i+1$ and so on and so forth until maximum level of the graph, in this case no plan can be found).
4. Set the union of their preconditions as a new goal for proposition level $i-1$ and repeat from the first step. If $i-1=1$, a plan has been found.

Satisfiability techniques

Idea of sat based techniques

Definitions

- Propositional variable v: can take two values (true or false).
- Negation: $\neg v$ is true if v is false (and conversely).
- Clause: disjunction ($\vee$) of (negations of) propositional variables.

SAT problem

A set V of propositional variables, a set C of clauses over these variables. Is there a truth assignation for the variables of V so that the conjunction ($\wedge$) of the clauses in C is a true formula? If so, give such an assignation.

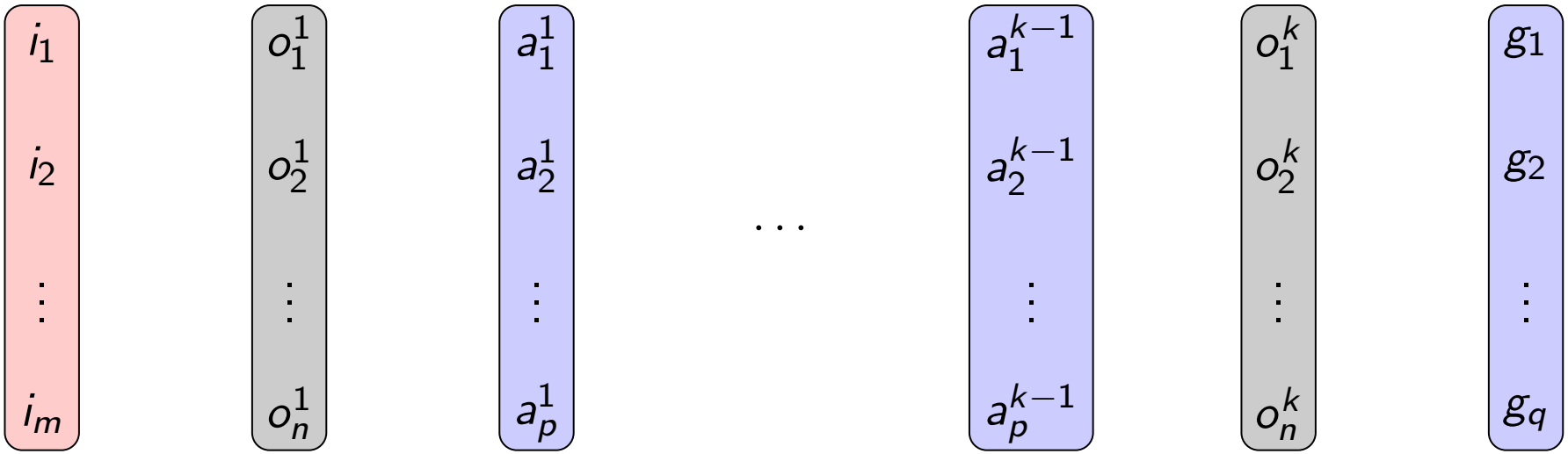
SAT Solvers and planning problems

SAT solvers are very efficient (they were widely studied). Modeling planning problems as SAT allow to solve them efficiently in practice .

From planning to SAT

Basic idea

- Assume that there exists a plan of length n (fixed before the search).
- Create one variable per atom per time step (stating if each atom is (or is not) in the state reached after k actions), and
- create one variable per action per time step (stating which action is used at each time step, only one will be true at each time step).



From planning to SAT, continued

Link between variables

The variable o^p representing action o at step p can only be true if

- all the variables representing its preconditions at step $p-1$ are true.

If o^p is true, then

- all its positive effects are true at step p,
- all its negative effects are false at step p, and
- all the other variables representing atoms are equal at step p and at step $p-1$.

SAT variables and plans

If the SAT problem has a solution then the planning problem has a plan. Any solution has one action variable true per time step. This sequence of action variables directly gives a plan to the planning problem.

Formalization: initial and goal states

Initial state

When the initial state is $s_0 = \{a_1, \ldots, a_n\}$, the clauses $\{a_1^0\}, \{a_2^0\}, \ldots \{a_n^0\}$ are added to the SAT problem. And for any $a \notin s_0$ the clause $\{\neg a\}$ is added to the SAT problem.

Goal states (for a plan of length k)

If the goal states are described by the set $\{a_1, \ldots, a_m\}$, the clauses $\{a_1^k\}, \{a_2^k\}, \ldots, \{a_m^k\}$ are added to the SAT problem.

Example

For a simple version of the blocks world example with three blocks (A, B, C) initially stacked in alphabetical order and so that the goal is to put them all on the table, give the initial state and goal states representations in a SAT problem (plan length of 4).

Formalization: preconditions and effects of actions

Action preconditions

An action $o = (p, e)$ can be used at time step i only if its preconditions are true at time step $i - 1$:

$$o^i \Rightarrow \bigwedge_{a \in p} a^{i-1}$$

Action effects

If an action $o = (p, e)$ is used at time step i its effects must be true at time step i:

$$o^i \Rightarrow \bigwedge_{a \in e} a^i$$

Exercise

Reformulate these statements as sets of clauses.

Formalization: ensuring progress

Some action occurs at each time step

For a time step i, an action must occur:

$$o_1^i \vee o_2^i \vee \cdots \vee o_p^i$$

here the set of actions is $\{o_1, \ldots, o_p\}$.

No more than one action occurs at each time step

For a time step i, each two actions are mutually exclusives:

$$\forall t \neq u \in \{1, \ldots, p\}, \neg o_t^i \vee \neg o_u^i,$$

here the set of actions is still $\{o_1, \ldots, o_p\}$.

Exercise

How many clauses represent the operators for time step i in blocks world?

Planning and SAT: example

Step 1

Represent one operator (of your choice) of the blocks world example at time step i.

Planning and SAT: example

Step 1

Represent one operator (of your choice) of the blocks world example at time step i.

Step2

Represent the full blocks world example with 4 time steps.

Planning and SAT: example

Step 1

Represent one operator (of your choice) of the blocks world example at time step i.

Step2

Represent the full blocks world example with 4 time steps.

Step3

Find a plan of length 4 in the blocks world example. Find the corresponding model for the SAT instance built at the previous step.

More on planning with SAT solvers

Original paper

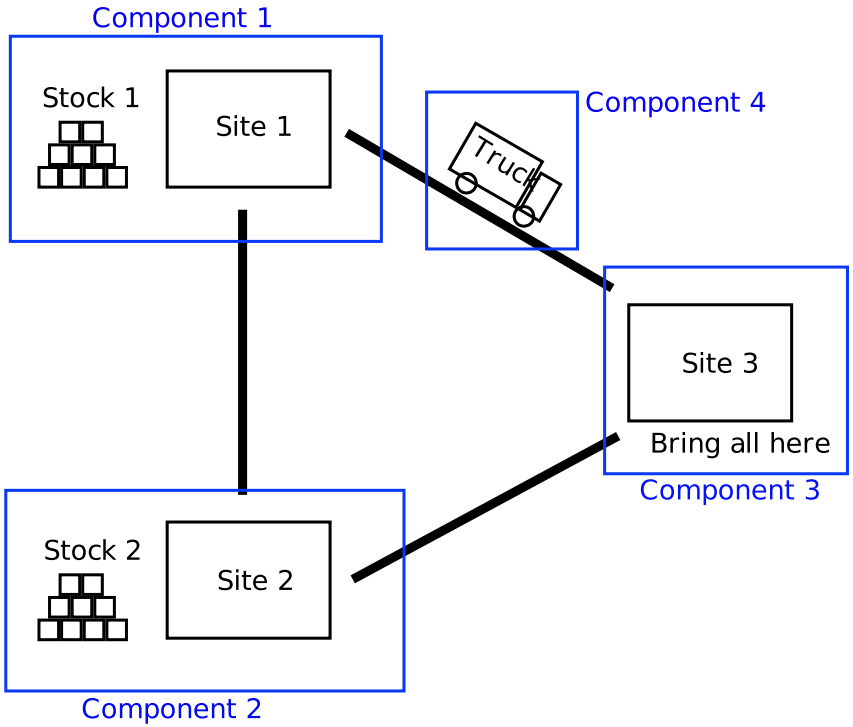
Henry Kautz and Bart Selman. *Planning as Satisfiability*. ECAI 1992.

Overview and bibliography

`https://users.aalto.fi/~rintanj1/jussi/satplan.html`

Factored planning

Factored planning: principles

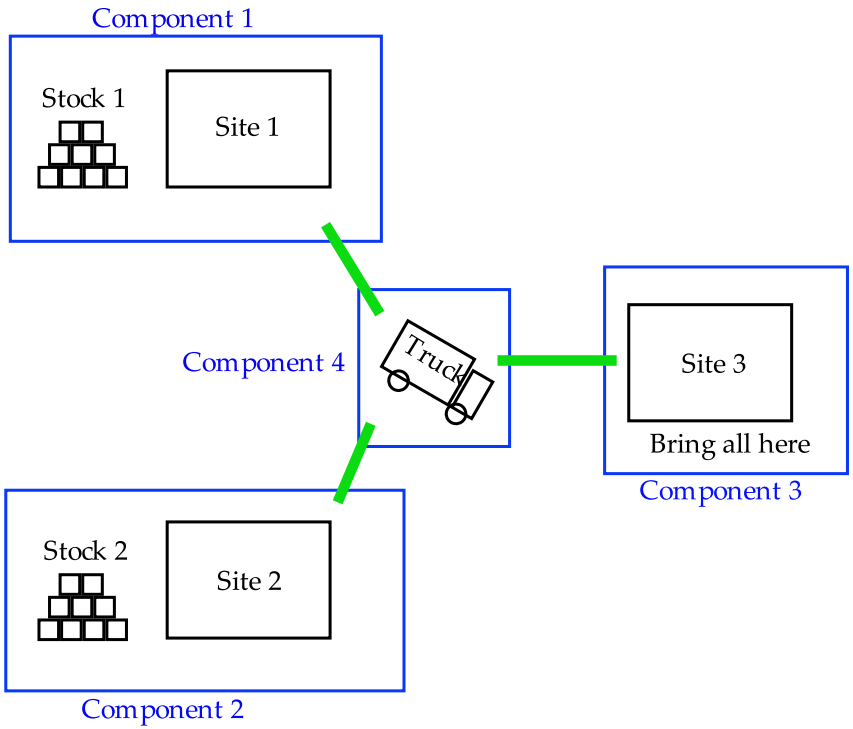


Each component is a planning problem with its own resources and actions

Goal

Find a set of *compatible* local plans: they can be *interleaved* into a global plan

Factored planning: principles



Each component is a planning problem with its own resources and actions

The components interact by resources and/or actions

Goal

Find a set of *compatible* local plans: they can be *interleaved* into a global plan

A few examples of factored planning approaches

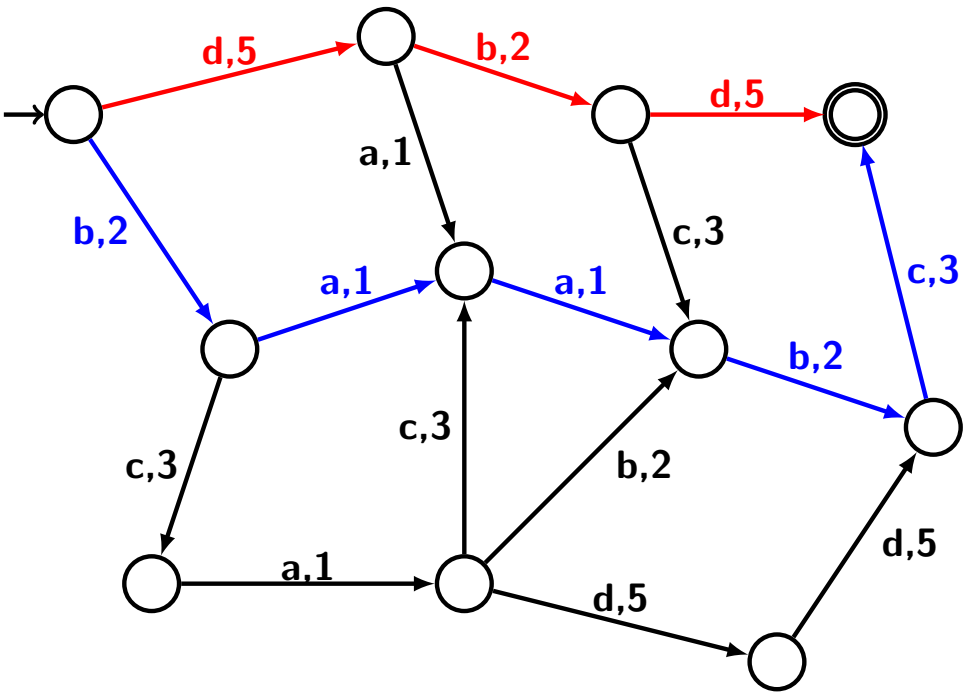
Eyal Amir and Barbara Engelhardt. *Factored Planning*. IJCAI 2003.

Ronen Brafman and Carmel Domshlak. *Factored Planning: How, When, and When Not*. AAAI 2006.

Éric Fabre, Loïg Jezequel, Patrik Haslum, and Sylvie Thiébaux. *Cost-Optimal Factored Planning: Promises and Pitfalls*. ICAPS 2010.

Cost-optimal factored planning using message-passing algorithms

Centralized planning problem = weighted automaton



Set of actions Σ

The words are the plans

The words with minimal cost are the cost-optimal plans

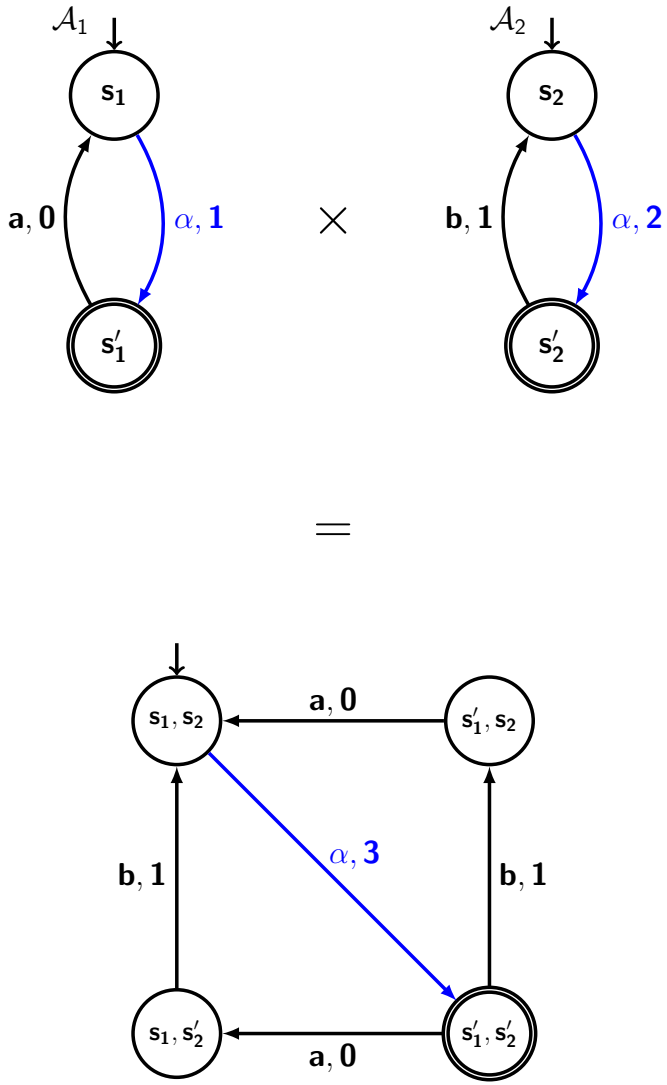
Goal

Find a minimal cost word in a weighted automaton

Factored planning problem

Components are weighted automata

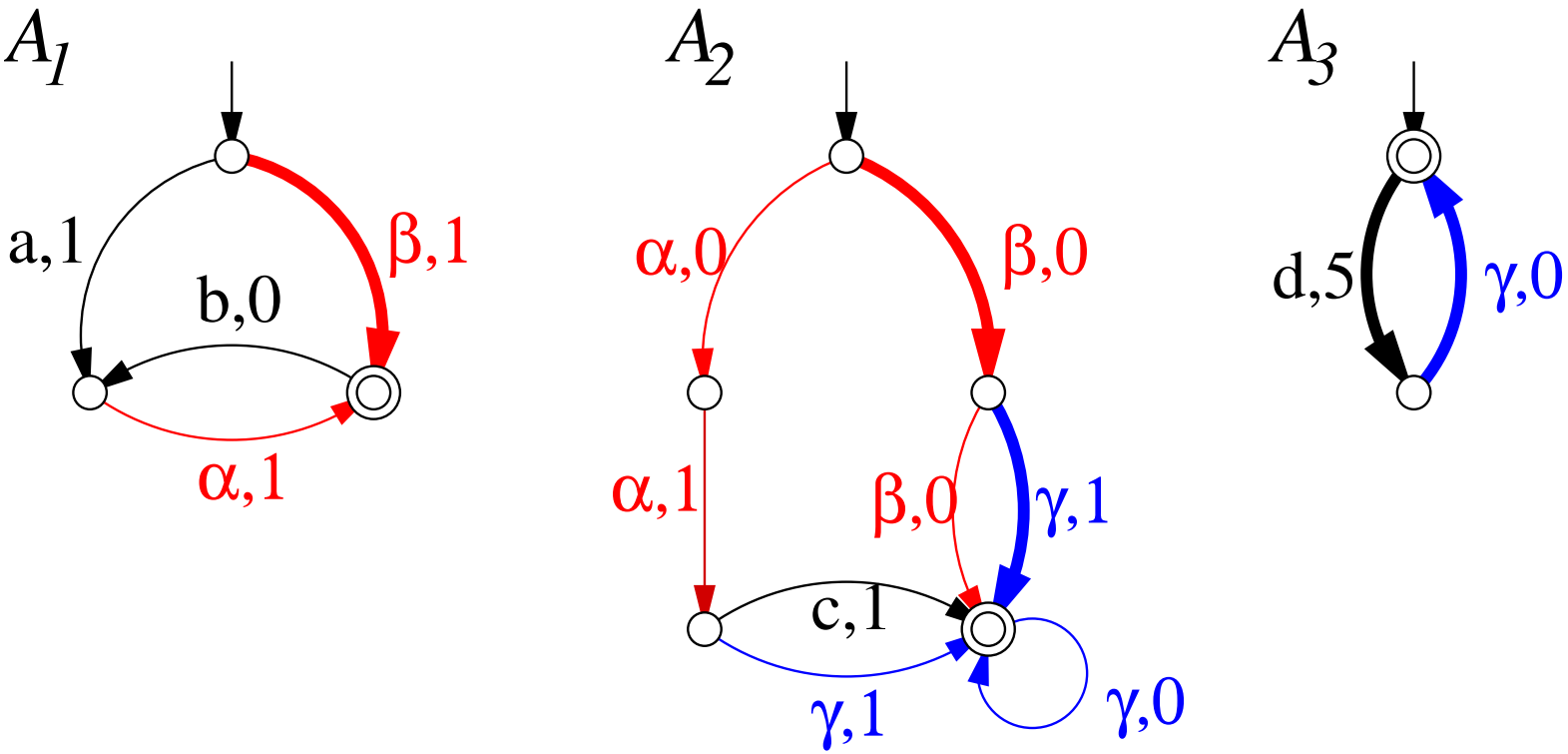
They interact by their shared actions: formalization using the notion of *synchronous product*



Goal

In $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, find a tuple $(w_1, \ldots, w_n)$ of words which are all *compatible* and *minimize* the sum of their cost, *without computing* $\mathcal{A}$

Factored planning problem: example



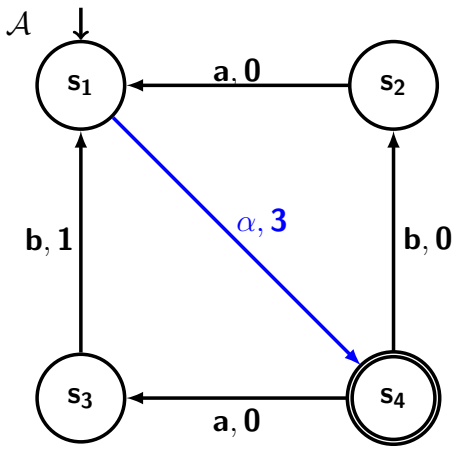
Centralized plans: $\beta d \gamma$ and $d \beta \gamma$
Factored/distributed/concurrent plan: $(\beta, \beta\gamma, d\gamma)$

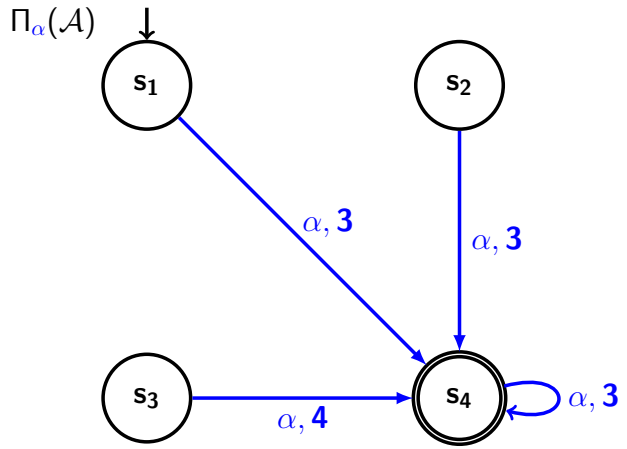
Projection: from global plans to local plans

Projection reduces a *global plan* to the actions of *a particular component*

$\Pi_{\Sigma'}$ corresponds to:

1. Replace each action not in Σ' by ε
2. Perform ε-reduction (to the left)
3. (Minimize)





MPA: computing the summaries $\Pi_{\Sigma_i}(\mathcal{A})$

Central properties of the projection of $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$

1. any *cost-optimal* word w of $\mathcal{A}$ can be projected into a *cost-optimal* word w_i of $\Pi_{\Sigma_i}(\mathcal{A})$, moreover $c(w) = c_i(w_i)$
2. any *cost-optimal* word w_i of $\Pi_{\Sigma_i}(\mathcal{A})$ is the projection of a *cost-optimal* word w of $\mathcal{A}$, moreover $c_i(w_i) = c(w)$

Consequence

Taking the minimal cost word in each $\Pi_{\Sigma_i}(\mathcal{A})$ gives a cost-optimal global plan (hypothesis: it is unique)

Building the $\Pi_{\Sigma_i}(\mathcal{A})$ by local computations

Successive refinements of the $\mathcal{A}_i$ from the constraints imposed by their neighbours

How to get the $\Pi_{\Sigma_i}(\mathcal{A})$: the message passing algorithms

Fundamental property (conditional independence)

$$\Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1 \times \mathcal{A}_2) \equiv_{\mathcal{L}} \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2)$$

Application:

$$\mathcal{A}_1 \overset{\Sigma_1 \cap \Sigma_2}{\text{———}} \mathcal{A}_2 \overset{\Sigma_2 \cap \Sigma_3}{\text{———}} \mathcal{A}_3$$

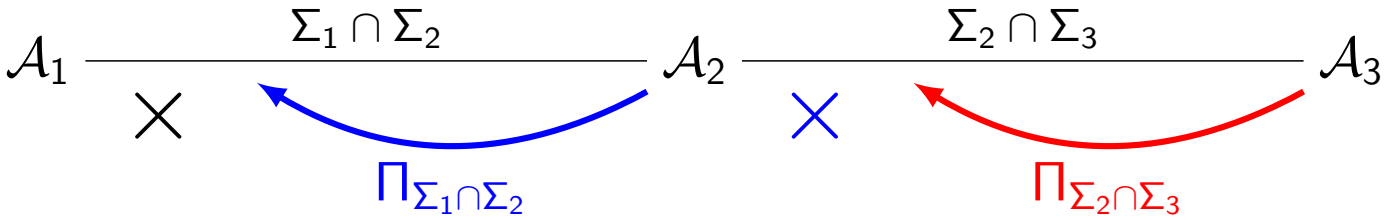
$$\begin{aligned}
\Pi_{\Sigma_1}(\mathcal{A}) &= \Pi_{\Sigma_1}(\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \Pi_{\Sigma_1}(\mathcal{A}_1) \times \Pi_{\Sigma_1}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \Pi_{\Sigma_2 \cap \Sigma_3}(\mathcal{A}_3))
\end{aligned}$$

How to get the $\Pi_{\Sigma_i}(\mathcal{A})$: the message passing algorithms

Fundamental property (conditional independence)

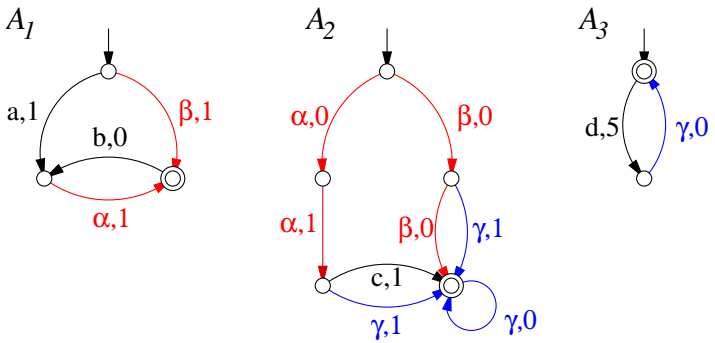
$$\Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1 \times \mathcal{A}_2) \equiv_{\mathcal{L}} \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2)$$

Application:

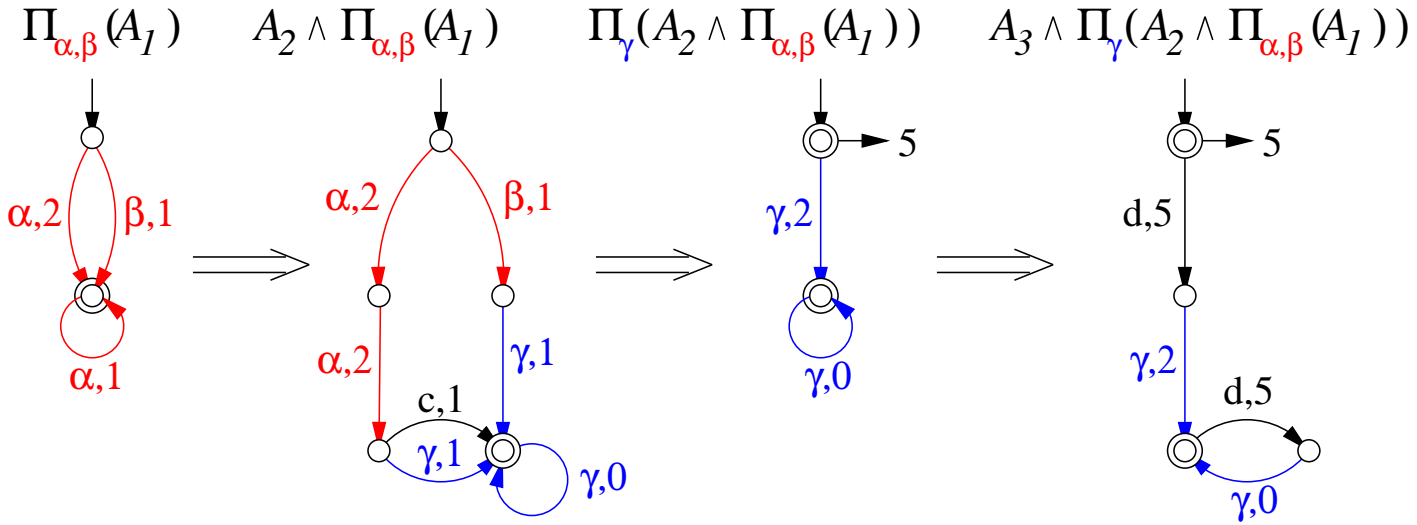


$$\begin{aligned}
\Pi_{\Sigma_1}(\mathcal{A}) &= \Pi_{\Sigma_1}(\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \Pi_{\Sigma_1}(\mathcal{A}_1) \times \Pi_{\Sigma_1}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \Pi_{\Sigma_2 \cap \Sigma_3}(\mathcal{A}_3))
\end{aligned}$$

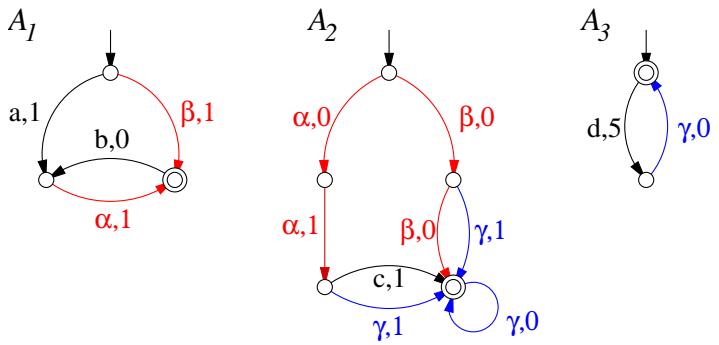
Example



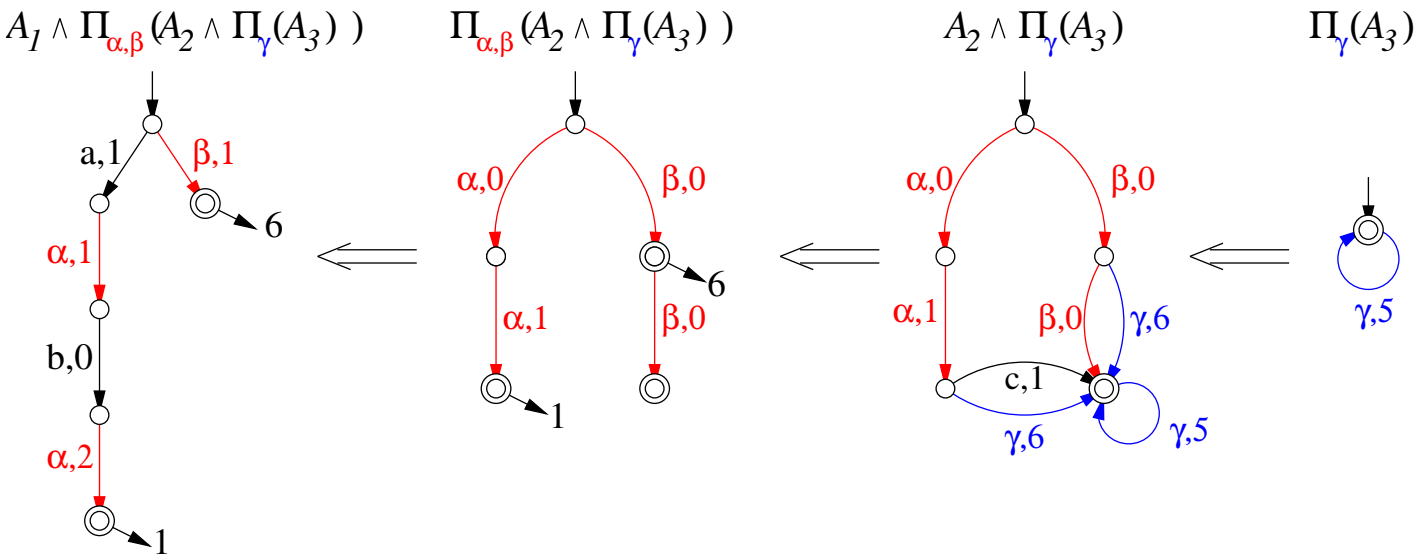
Messages from $\mathcal{A}_1$ to $\mathcal{A}_3$:



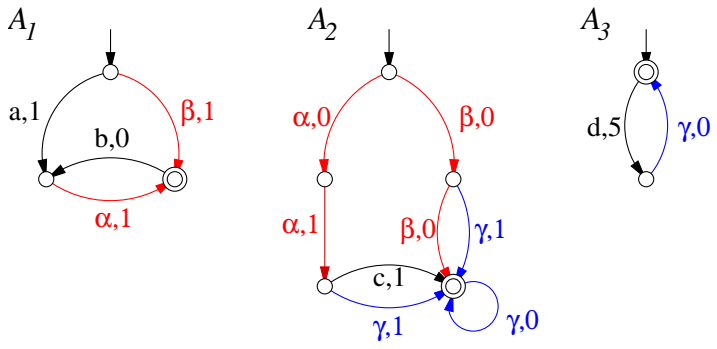
Example



Messages from $\mathcal{A}_3$ to $\mathcal{A}_1$:

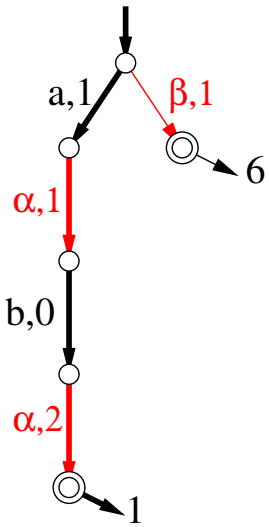


Example

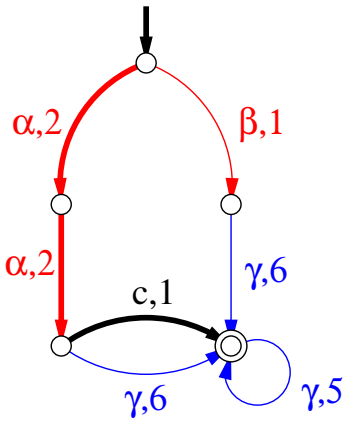


Updated components:

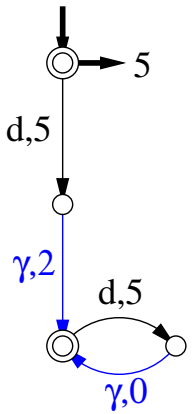
$\Pi_{\alpha,\beta,a,b}(A)$



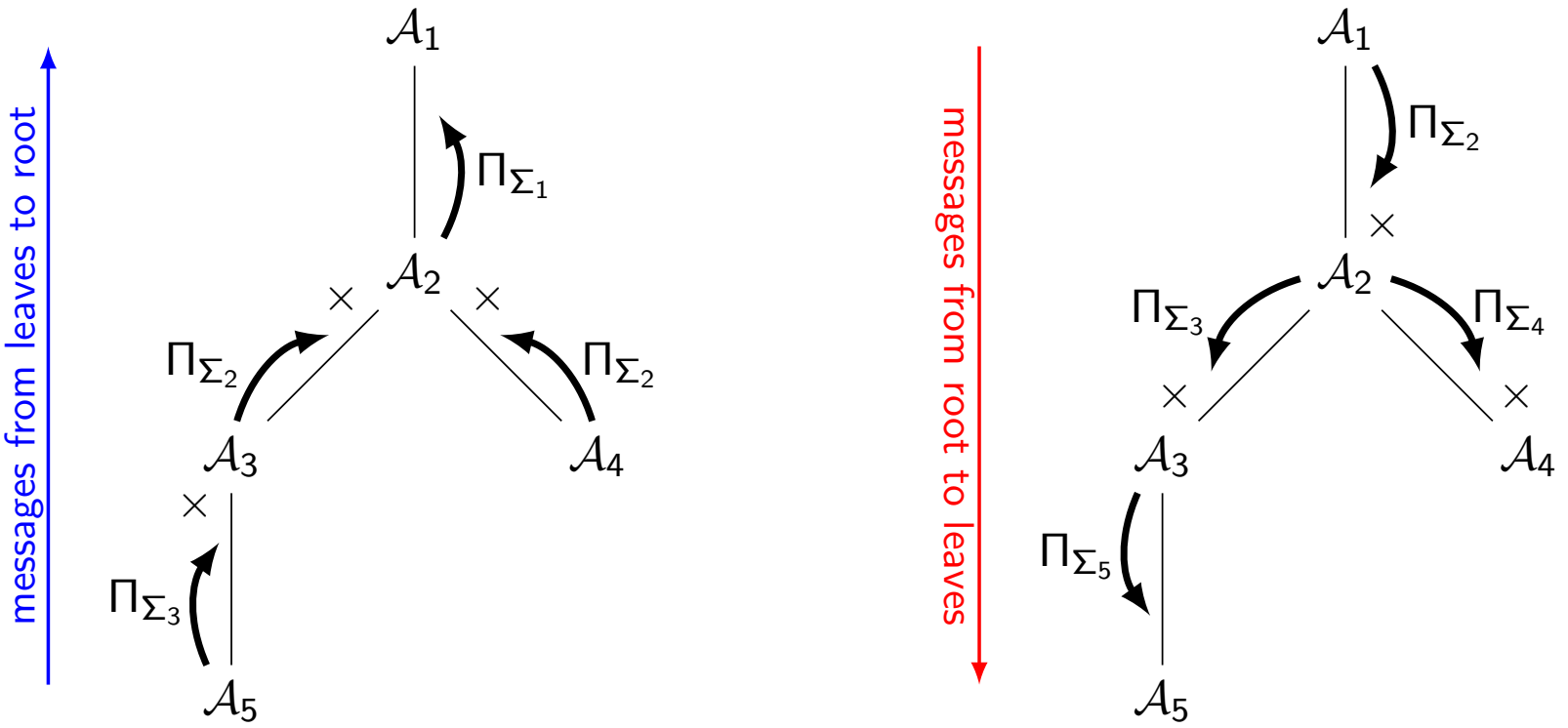
$\Pi_{\alpha,\beta,\gamma,c}(A)$



$\Pi_{\gamma,d}(A)$



Message passing algorithms: main results generalized



Theorem

If $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ has a *tree shaped interaction* graph, the message passing algorithm *converges* and returns $\mathcal{A}'_i \equiv_{\mathcal{L}} \Pi_{\Sigma_i}(\mathcal{A})$ for each $\mathcal{A}_i$