Automated Planning Tools for Intelligent Decision Making

3. AI Planning, Part I: Framework

Álvaro Torralba

AALBORG UNIVERSITET

Spring 2021

Thanks to Jörg Hoffmann for slide sources

Agenda

Problem Solving

Shakey the Robot (1966 - 1972)

→the first general-purpose mobile robot to be able to reason about its own actions

AI Planning

"Planning is the art and practice of thinking before acting." — Patrik Haslum

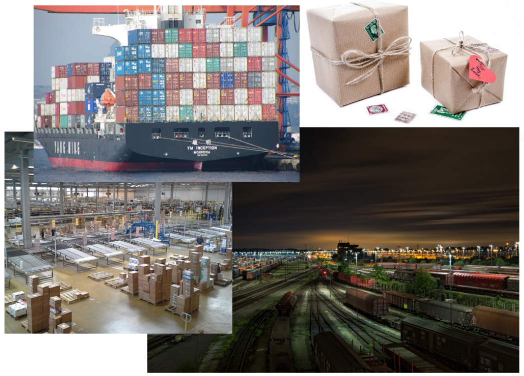
1. Model-based: Given a description of the environment, and the goals of the agent
2. Sequential decision making: decide which actions to perform to achieve the goals
3. Domain independent: Develop a general tool that can solve all problems of this kind

Classical Planning

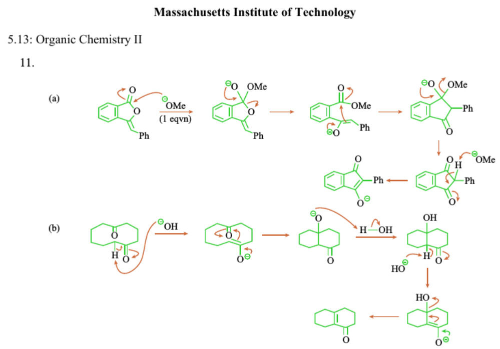
- Deterministic
- Fully observable
- Static
- Single-agent
- Discrete
- Sequential

So, What is (Classical) Planning Good For?

Solitaire Puzzles



Logistics



Molecule Synthesis

→and many others!

Planning

Ambition:

Write one program (planner) that can solve all sequential decision-making problems.

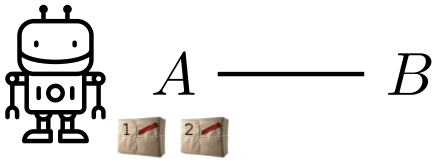
How do we describe our problem to the planner?

- A *logical description* of the possible states
- A *logical description* of the initial state I
- A *logical description* of the goal condition G
- *logical description* of the set A of actions in terms of preconditions and effects

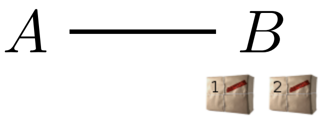
$\rightarrow$ Solution (plan) $=$ sequence of actions from A, transforming I into a state that satisfies G.

Classical Planning Task

- Initial state



- Goal



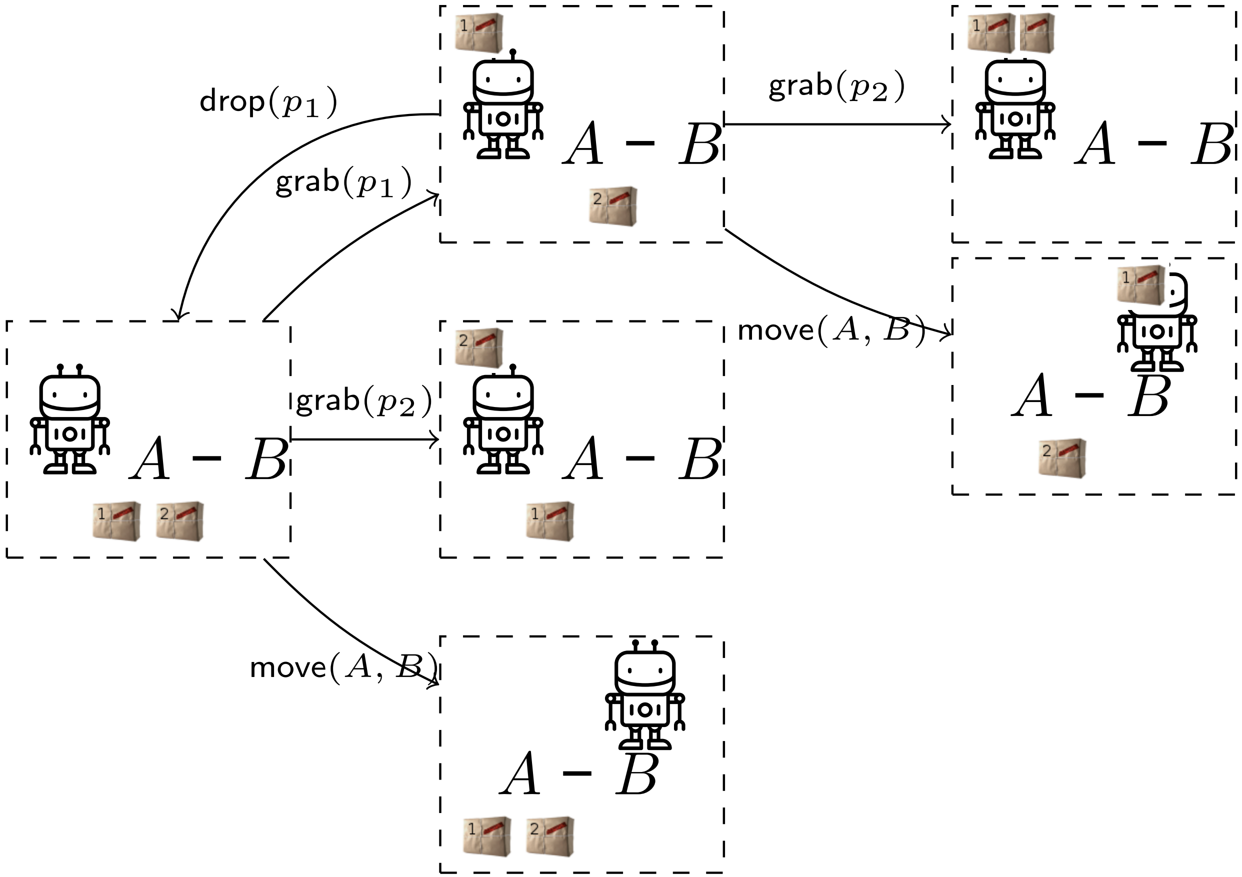
- Actions: $grab(p_1), grab(p_2), drop(p_1), drop(p_2), move(A, B)$
  →For each action we specify its preconditions and effect

→Find a plan: action sequence from the initial state to another where the goal holds

Satisficing planning: Find a plan as cheap as possible (no guarantees)

Optimal planning: Find a plan of minimum cost (guaranteed)

How to solve planning tasks? Search!



There is a lot of research on how to solve planning problems, a lot of algorithms (search is just one option) and tools.

Today, we will focus on how to phrase our problems as planning tasks so that we can use an existing planner to solve them!

Our Agenda for This Chapter

- **The STRIPS Planning Formalism:** Which concrete planning formalism will we be using?

  → Lays the framework we'll be looking at.

- **Planning Complexity:** How complex is planning?

  → The price of generality is complexity. Here's what that "price" is.

- **Planning Domain Definition Language: How to Use a Planner?**

  → A language to rule them all.

- **Applications: What are you planning for?**

  → A few problems we can solve (and which some people care about).

"STRIPS" Planning

- STRIPS = Stanford Research Institute Problem Solver.

  STRIPS is the simplest possible (reasonably expressive) logics-based planning language.
- STRIPS has only Boolean variables: propositional logic atoms.
- Its preconditions/effects/goals are as canonical as imaginable:
  - Preconditions, goals: conjunctions of positive atoms.
  - Effects: conjunctions of literals (positive or negated atoms).
- We use the common set-based notation for this simple formalism.

→ Historical note: STRIPS [**?**] was originally a planner, whose language actually wasn't quite that simple.

STRIPS Planning: Syntax

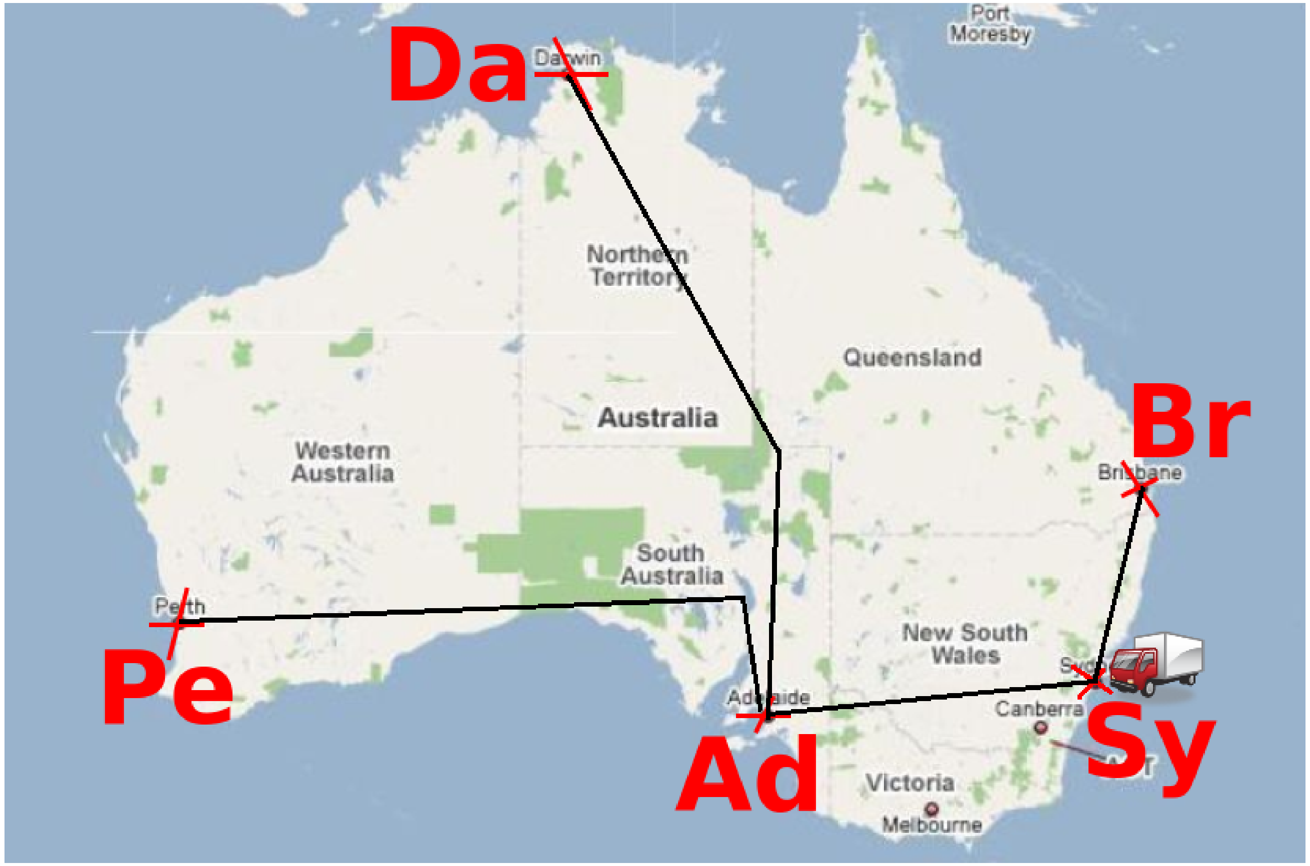
Definition (STRIPS Planning Task). *A STRIPS planning task, short planning task, is a 4-tuple* $\Pi = (P, A, I, G)$ *where:*

- P *is a finite set of facts (aka propositions).*
- A *is a finite set of actions; each* $a \in A$ *is a triple* $a = (pre_a, add_a, del_a)$ *of subsets of* P *referred to as the action's precondition, add list, and delete list respectively; we require that* $add_a \cap del_a = \emptyset$*.*
- $I \subseteq P$ *is the initial state.*
- $G \subseteq P$ *is the goal.*

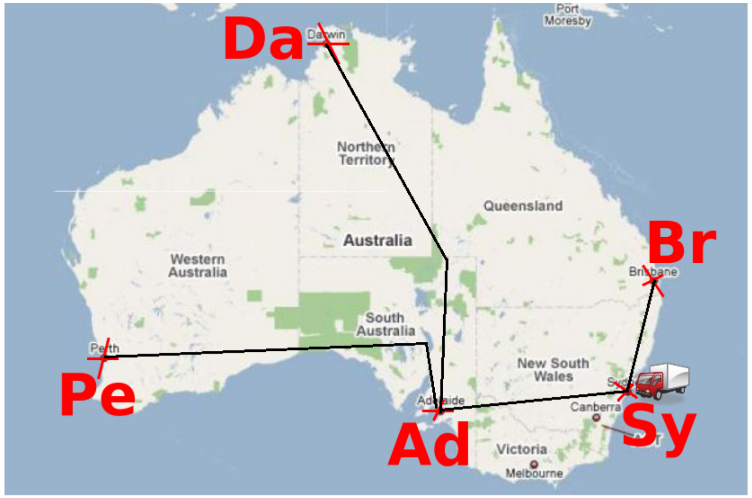
We will often give each action $a \in A$ *a name (a string), and identify* a *with that name.*

Note: We assume unit costs for simplicity: every action has cost 1.

"TSP" in Australia



STRIPS Encoding of "TSP"



- Facts P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Initial state I:
- Goal G:

- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  Precondition pre_a:
  Add list add_a:
  Delete list del_a:
- Plan:

STRIPS Planning: Semantics

Definition (STRIPS State Space). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. The state space of Π is $\Theta_\Pi = (S, A, T, I, S^G)$ where:*

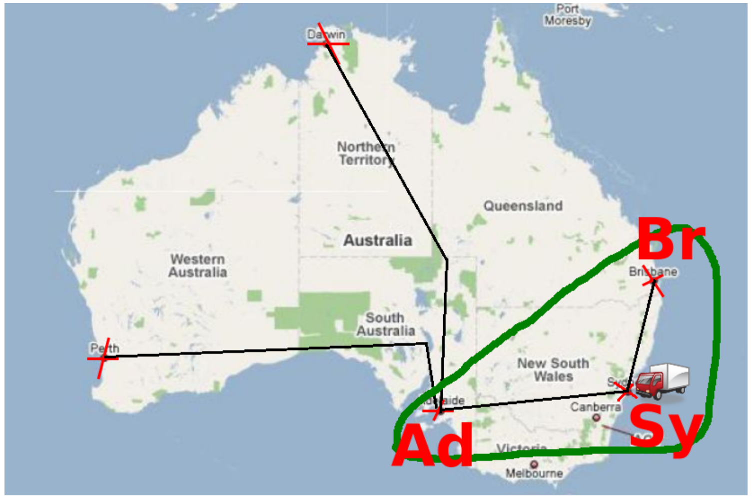
- *The states (also world states) $S = 2^P$ are the subsets of P.*
- *A is Π's action set.*
- *The transitions are $T = \{s \xrightarrow{a} s' \mid pre_a \subseteq s, s' = appl(s, a)\}$.*

  If $pre_a \subseteq s$, then a is applicable in s and $appl(s, a) := (s \cup add_a) \setminus del_a$.
  If $pre_a \not\subseteq s$, then $appl(s, a)$ is undefined.
- *I is Π's initial state.*
- *The goal states $S^G = \{s \in S \mid G \subseteq s\}$ are those that satisfy Π's goal.*

An (optimal) plan for $s \in S$ is an (optimal) solution for s in Θ_Π, i.e., a path from s to some $s' \in S^G$. A solution for I is called a plan for Π. Π is solvable if a plan for Π exists.

For $\vec{a} = \langle a_1, \dots, a_n \rangle$, $appl(s, \vec{a}) := appl(\dots appl(appl(s, a_1), a_2) \dots, a_n)$ if each a_i is applicable in the respective state; else, $appl(s, \vec{a})$ is undefined.

STRIPS Encoding of Simplified "TSP"

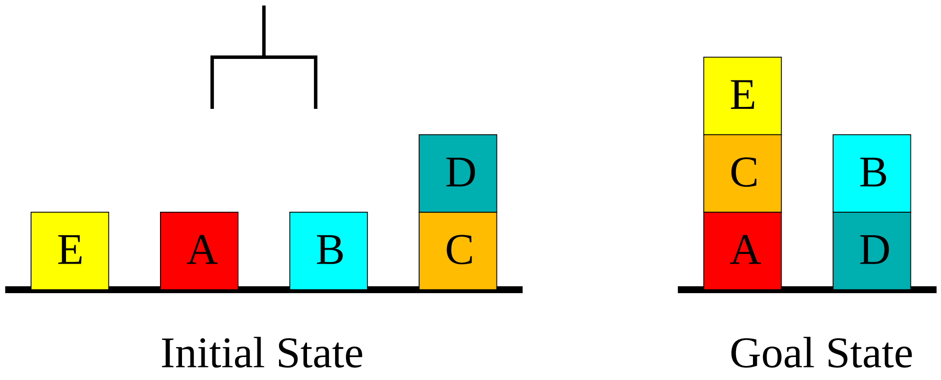


- Facts P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$.
- Initial state I:
- Goal G: $\{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$. (Note: no "$at(Sydney)$".)
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  - Precondition pre_a:
  - Add list add_a:
  - Delete list del_a:

STRIPS Encoding of Simplified "TSP": State Space

→ Is this actually the state space?

(Oh no it's) The Blocksworld



- Facts: $on(x,y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- Initial state: $\{onTable(E)$, $clear(E)$, $\dots$, $onTable(C)$, $on(D,C)$, $clear(D)$, $armEmpty()\}$.
- Goal: $\{on(E,C)$, $on(C,A)$, $on(B,D)\}$.
- Actions: $stack(x,y)$, $unstack(x,y)$, $putdown(x)$, $pickup(x)$.
- $stack(x,y)$?

Questionnaire

Question!

Which are correct encodings (part of <u>some</u> correct overall encoding) of the STRIPS Blocksworld $pickup(x)$ action schema?

(A): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x)\})$.

(B): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{armEmpty()\})$.

(C): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty(), clear(x)\})$.

(D): $(\{onTable(x), clear(x), armEmpty()\}, \{holding(x)\}, \{onTable(x), armEmpty()\})$.

Exercises

Exercise 1: STRIPS Modelling

Algorithmic Problems in Planning

Satisficing Planning

Input: A planning task Π.
Output: A plan for Π, or "unsolvable" if no plan for Π exists.

Optimal Planning

Input: A planning task Π.
Output: An *optimal* plan for Π, or "unsolvable" if no plan for Π exists.

$\rightarrow$ The techniques successful for either one of these are almost disjoint. And satisficing planning is *much* more effective in practice.

$\rightarrow$ Programs solving these problems are called (optimal) planners, planning systems, or planning tools.

Decision Problems in (STRIPS) Planning

Definition (PlanEx). *Given a STRIPS task Π, does there exists a plan for Π?*
$\rightarrow$ Corresponds to satisficing planning.

Theorem. *PlanEx is* **PSPACE***-complete.*

Definition (PlanLen). *Given a STRIPS task Π and an integer K, does there exists a plan for Π of length at most K?* $\rightarrow$ Corresponds to optimal planning.

Theorem. *PlanLen is* **PSPACE***-complete.*

Definition (PolyPlanLen). *Given a STRIPS planning task Π and an integer K bounded by a polynomial in the size of Π, does there exists a plan for Π of length at most K?* $\rightarrow$ Corresponds to optimal planning with "small" plans.

Theorem. *PolyPlanLen is* **NP***-complete.*

Example of a planning domain with exponentially long plans?

$\rightarrow$Classical Planning is as hard as SAT if plans are of polynomial length, harder if plans are exponentially long

Domain-Specific PlanEx vs. PlanLen . . .

. . . is more interesting than the general case.

- *In general*, both have the same complexity.
- *Within particular applications*, bounded length plan existence (optimal planning) is often harder than plan existence (satisficing planning).
- This happens in many planning competition benchmark domains: PlanLen is **NP**-complete while PlanEx is in **P**.
- For example: Blocksworld and Logistics.

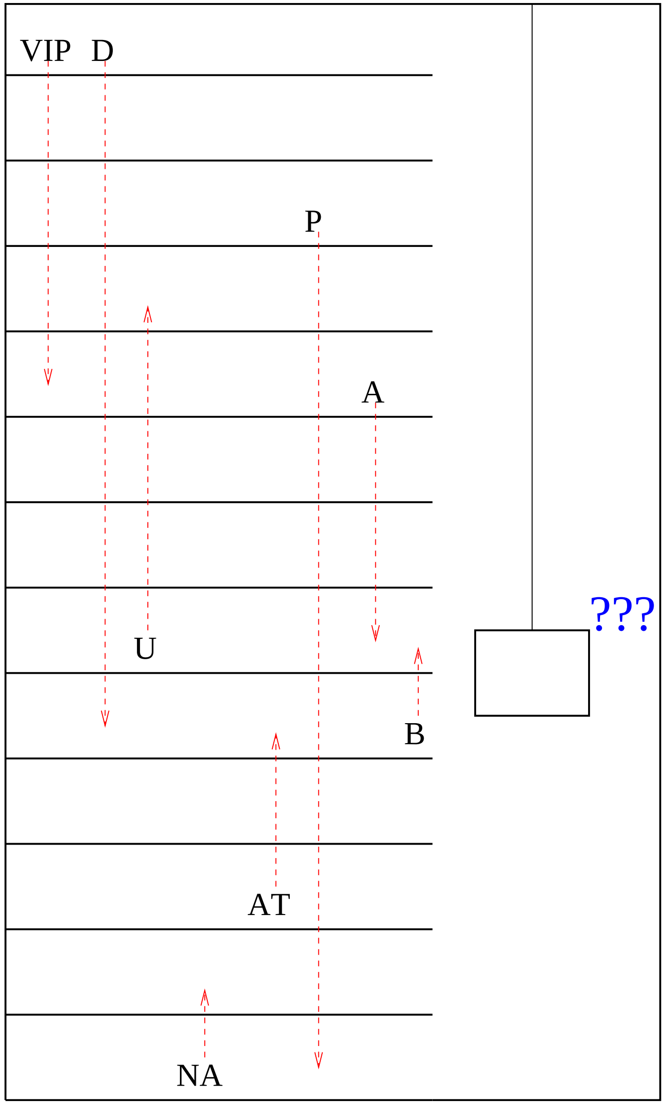
→ In practice, optimal planning is (almost) never easy.

The Blocksworld is Hard?

The Blocksworld is Hard!

Miconic-ADL: PlanEx is Hard

- VIP: Served first.
- D: Lift may only go *down* when inside; similar for U.
- NA: Never-alone; AT: Attendant.
- A, B: Never together in the same elevator (!)
- P: Normal passenger :-)



PDDL History

Planning Domain Description Language:

- A description language for planning in the STRIPS formalism and various extensions.
- Used in the International Planning Competition (IPC).

- 1998: PDDL [**?**].
- 2000: "PDDL subset for the 2000 competition" [**?**].
- 2002: PDDL2.1, Levels 1-3 [**?**].
- 2004: PDDL2.2 [**?**].
- 2006: PDDL3 [**?**].

PDDL Quick Facts

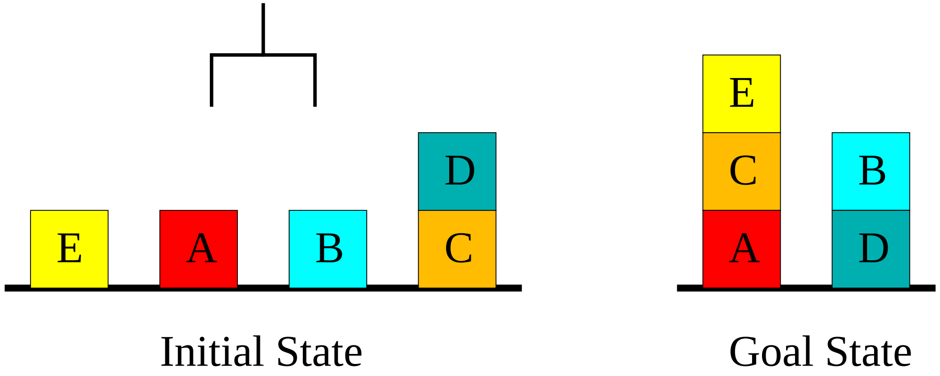
PDDL is not a propositional language:

- Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic)
- Action schemas parameterized by objects.
- Predicates to be instantiated with objects.

A PDDL planning task comes in two pieces:

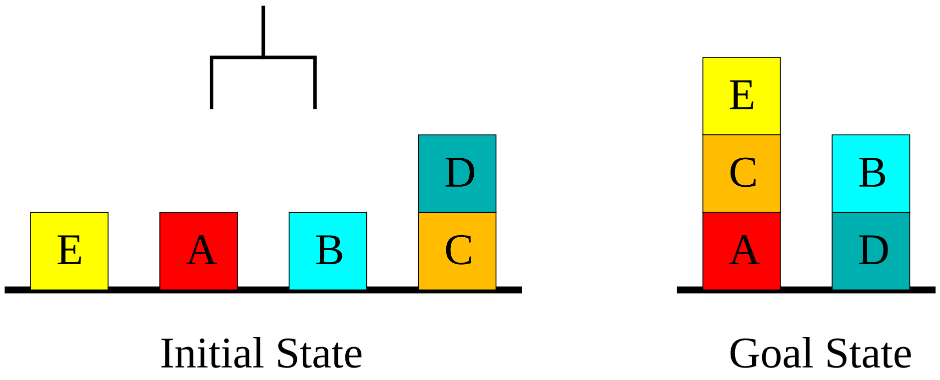
- The domain file and the problem file.
- The problem file gives the objects, the initial state, and the goal state.
- The domain file gives the predicates and the action schemas; each benchmark domain has *one* domain file.

The Blocksworld in PDDL (STRIPS): Domain File




```
(define (domain blocksworld)
(:predicates (clear ?x) (holding ?x) (on ?x ?y)
             (on-table ?x) (arm-empty))
(:action stack
 :parameters (?x ?y)
 :precondition (and (clear ?y) (holding ?x))
 :effect (and (arm-empty) (on ?x ?y)
              (not (clear ?y)) (not (holding ?x)))
)
...
```

The Blocksworld in PDDL (STRIPS): Problem File




```
(define (problem bw-abcde)
(:domain blocksworld)
(:objects a b c d e)
(:init (on-table a) (clear a)
       (on-table b) (clear b)
       (on-table e) (clear e)
       (on-table c) (on d c) (clear d)
       (arm-empty))
(:goal (and (on e c) (on c a) (on b d))))
```

Fast Downward

Fast Downward is a planning system featuring a lot of algorithms. When you run it you need to select which configuration to use:

```
./fast-downward.py (<domain>) <instance> --search "config"
```

```
./fast-downward.py --alias "config-alias" (<domain>) <instance>
```

There are A LOT of configurations. Here I list a few convenient ones:

Satisficing Planning:

- What we see in the lecture:
  `--evaluator "hff=ff(transform=adapt_costs(one))" --search "eager_greedy([hff], preferred=[hff], cost_type=one)"`
- `--alias lama-first`: Good configuration
- `--alias lama`: Good configuration (anytime)

Optimal Planning:

- `--search "astar(blind)"`: Dijkstra search
- `--search "astar(lmcut)"`: Ok configuration (though not best)

Action Description Language (ADL)

STRIPS + ADL (Action Description Language):

- Arbitrary first-order logic formulas in action preconditions and the goal: forall, exists, or, imply, not
- Conditional effects, i.e., effects that occur only if their separate effect condition holds: when

→A useful construct is effects of the form forall-when:

```
(forall (?x) (when (condition) (effect))
```

ADL is a real headache to implement:

- Most planners that do handle ADL *compile* it down [**?**]
- Example FF: 7000 C lines for compilation, 2000 lines core planner.

Action Costs

`(:requirements :action-costs)`

Domain file:

- Declare cost function

```
(:functions
    (road-length ?l1 ?l2 - location) - number ; optional
    (total-cost) - number ; The cost function must have this name
)
```

- Declare action cost as effect:
  `(increase (total-cost) (road-length ?l1 ?l2))`

Problem file:

- (optional) Declare costs in the initial state:
  `(= (total-cost) 0)`
  `(= (road-length city-3-loc-2 city-2-loc-3) 186)`
- Optimization criteria: `(:metric minimize (total-cost))`

PDDL Extensions

- PDDL 2.1: numeric and temporal planning
- PDDL 2.2: derived predicates (e.g., flow of current in an electricity network) and timed initial literals (e.g., sunrise and sunset, shop closing times).
- PDDL 3: soft goals (e.g.goals that have a reward) and preferences (e.g.temporal goals)

In practice, most planners only support a subset of PDDL. In this project, you should consider:

- STRIPS
- Negative Preconditions
- Forall-when effects
- Action costs

Questionnaire

Question!

What is PDDL good for?

(A): Nothing.

(B): Free beer.

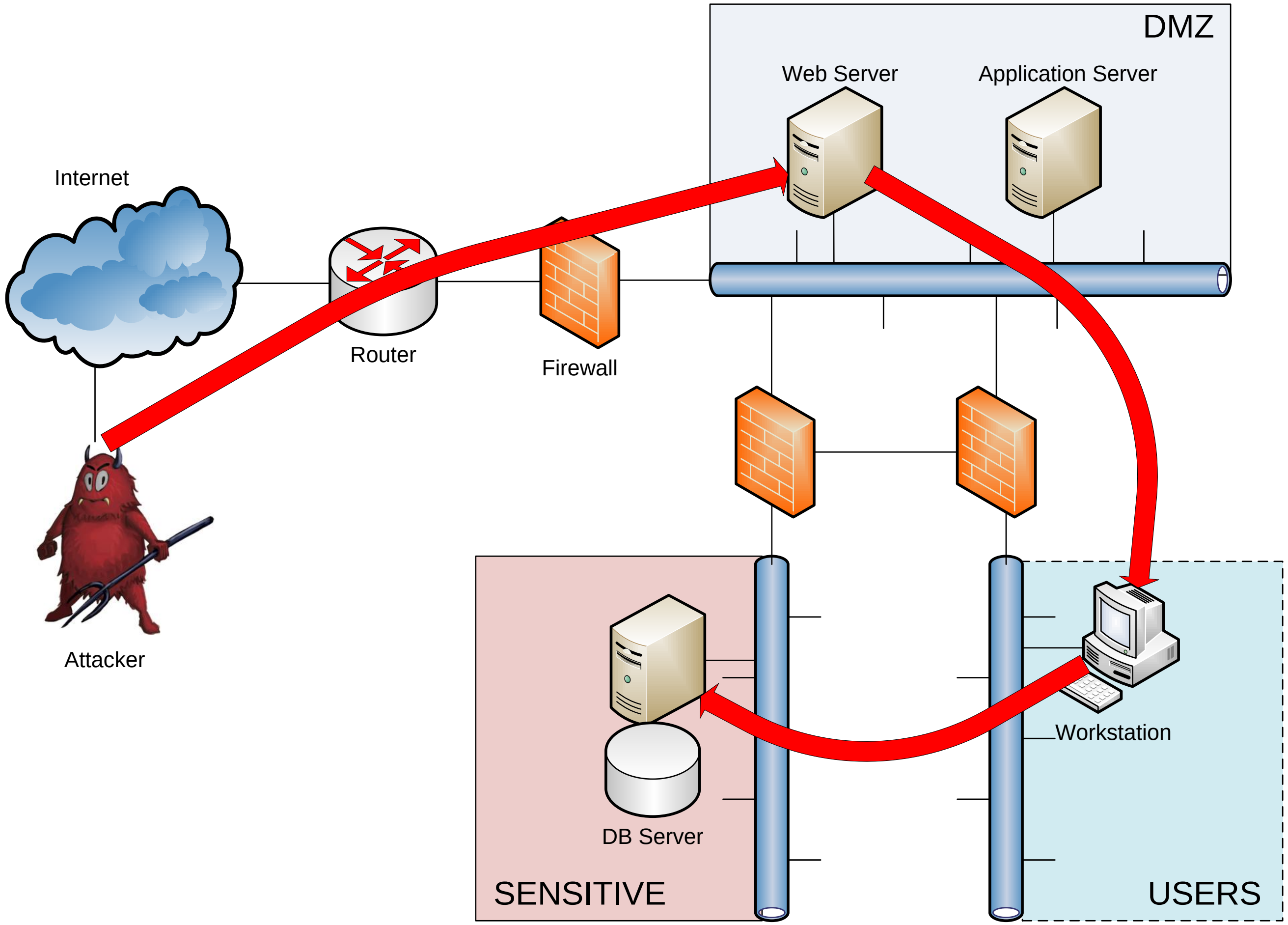
(C): Those AI planning guys.

(D): Being lazy at work.

Exercises

Exercise 2: PDDL Modelling

Network Hacking



Penetration Testing (Pentesting)

Pentesting

Actively verifying network defenses by conducting an intrusion in the same way an attacker would.

- Well-established industry (roots back to the 60s).
- Points out specific dangerous attacks (as opposed to vulnerability scanners).
- Pentesting tools sold by security companies, like Core Security.

  $\rightarrow$ Core IMPACT (since 2001); Immunity Canvas (since 2002); Metasploit (since 2003).
- Run security checks launching exploits.
- Core IMPACT uses FF for automation since 2010.

Motivation for Automation

Motivation for Automation: Wrap-Up

Simulated penetration testing serves to:

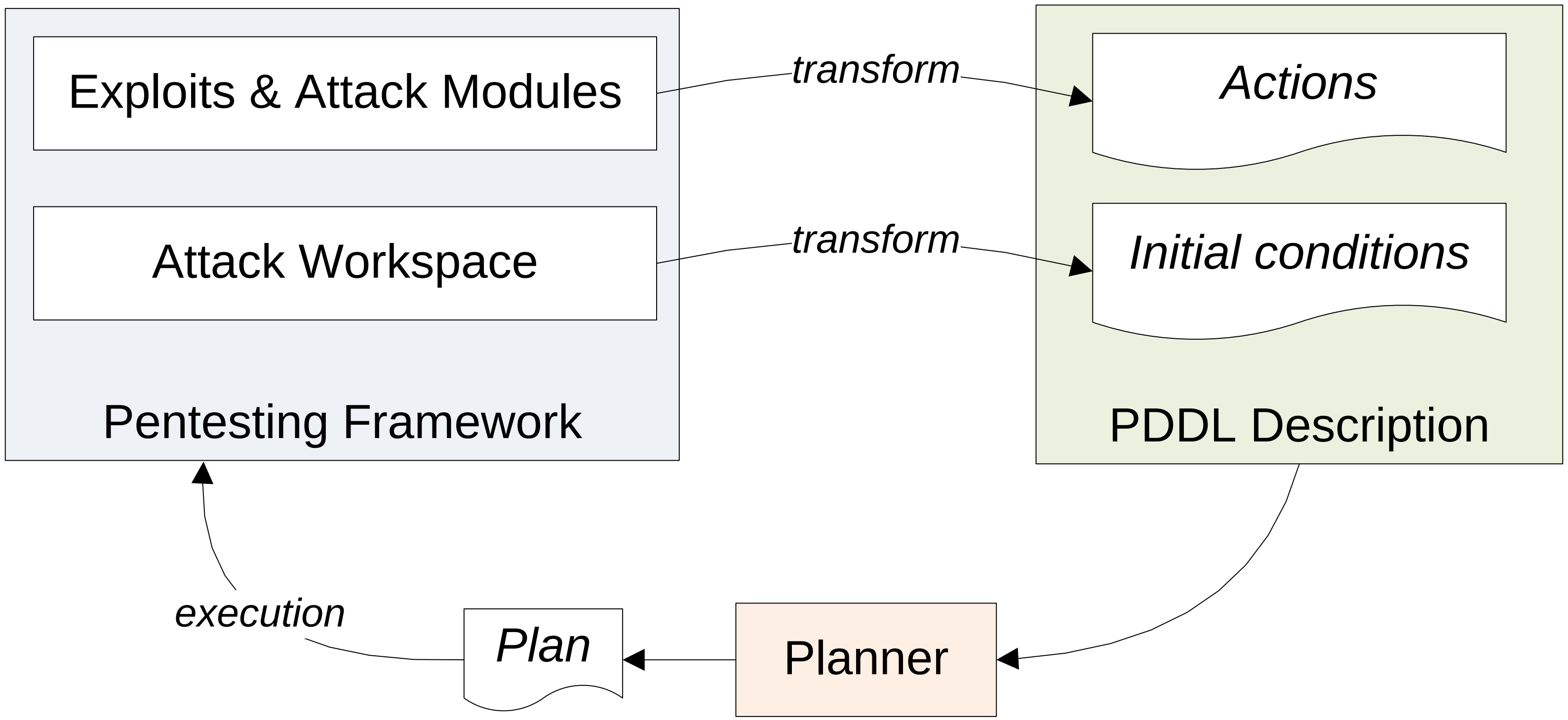
- Reduce human labor.
- Increase testing coverage:
  - Higher testing frequency.
  - Broader tests trying more possibilities.
- Deal with the dynamics of pentesting:
  - More exploits.
  - New tools used in attacks (Client-Side, WiFi, WebApps, . . .).

$\rightarrow$ The aim is to automate pentesting, so that the attacks can continuously be run in the background, thus decreasing human labor while allowing broad coverage of complex attack possibilities.

The Turing Test, Revisited

Simulated Pentesting at Core Security

Core IMPACT system architecture:



$\rightarrow$ In practice, the attack plans are being used to point out to the security team where to look.

Core Security PDDL

Object Types:

network	operating_system
host	OS_version
port	OS_edition
port_set	OS_build
application	OS_servicepack
agent	OS_distro
privileges	kernel_version

Core Security PDDL, ctd.

Predicates expressing connectivity:

```
(connected_to_network ?s - host ?n - network)
(IP_connectivity ?s - host ?t - host)
(TCP_connectivity ?s - host ?t - host ?p - port)
(TCP_listen_port ?h - host ?p - port)
(UDP_listen_port ?h - host ?p - port)
```

Core Security PDDL, ctd.

Predicates expressing configurations:

```
(has_OS ?h - host ?os - operating_system)
(has_OS_version ?h - host ?osv - OS_version)
(has_OS_edition ?h - host ?ose - OS_edition)
(has_OS_build ?h - host ?osb - OS_build)
(has_OS_servicepack ?h - host ?ossp - OS_servicepack)
(has_OS_distro ?h - host ?osd - OS_distro)
(has_kernel_version ?h - host ?kv - kernel_version)
(has_architecture ?h - host ?a - OS_architecture)
(has_application ?h - host ?p - application)
```

Core Security PDDL, ctd.

Actions modeling exploits:

```
(:action HP_OpenView_Remote_Buffer_Overflow_Exploit
:parameters (?s - host ?t - host)
:precondition (and (compromised ?s)
  (and (has_OS ?t Windows)
    (has_OS_edition ?t Professional)
    (has_OS_servicepack ?t Sp2)
    (has_OS_version ?t WinXp)
    (has_architecture ?t I386))
  (has_service ?t ovtrcd)
  (TCP_connectivity ?s ?t port5053)
)
:effect(and (installed_agent ?t high_privileges)
  (increase (time) 10)
))
```

Core Security PDDL, ctd.

Actions allowing to reap benefits of exploits:

```
(:action Mark_as_compromised
:parameters (?a - agent ?h - host)
:precondition (installed ?a ?h)
:effect (compromised ?h)
)

(:action IP_connect
:parameters (?s - host ?t - host)
:precondition (and (compromised ?s)
  (exists (?n - network)
    (and (connected_to_network ?s ?n)
      (connected_to_network ?t ?n))))
:effect (IP_connectivity ?s ?t)
)
```

Core Security PDDL, ctd.

An attack plan:

```
 0: Mark_as_compromised localagent localhost
 1: IP_connect localhost 10.0.1.1
 2: TCP_connect localhost 10.0.1.1 port80
 3: Phpmyadmin Server_databases Remote Code Execution
          localhost 10.0.1.1
 4: Mark_as_compromised 10.0.1.1 high_privileges
 ...
14: Mark_as_compromised 10.0.4.2 high_privileges
15: IP_connect 10.0.4.2 10.0.5.12
16: TCP_connect 10.0.4.2 10.0.5.12 port445
17: Novell Client NetIdentity Agent Buffer Overflow
          10.0.4.2 10.0.5.12
18: Mark_as_compromised 10.0.5.12 high_privileges
```

Simulated Pentesting@Core Security: Remarks

History:

- Planning domain "of this kind" (less IT-level, including also physical actions like talking to somebody) first proposed by [**?**]; used as benchmark in IPC'08 and IPC'11.
- Presented encoding proposed by [**?**].
- Used commercially by Core Security in Core INSIGHT since 2010.

Do Core Security's customers like this?

- I am told they do.
- In fact, they like it so much already that Core Security is very reluctant to invest money in making this better ...

Questionnaire

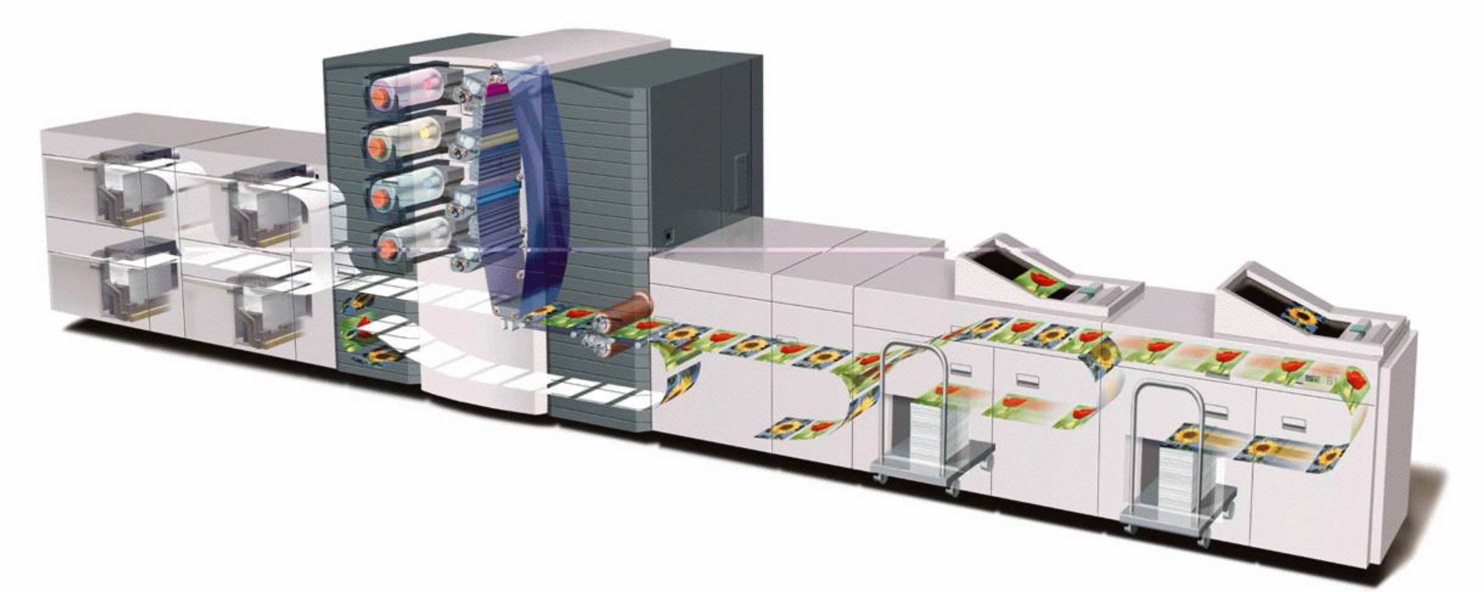
Question!

Is the current realization @Core Security really a simulation of what human hackers do?

(A): Yes. (B): No.

FAI Research

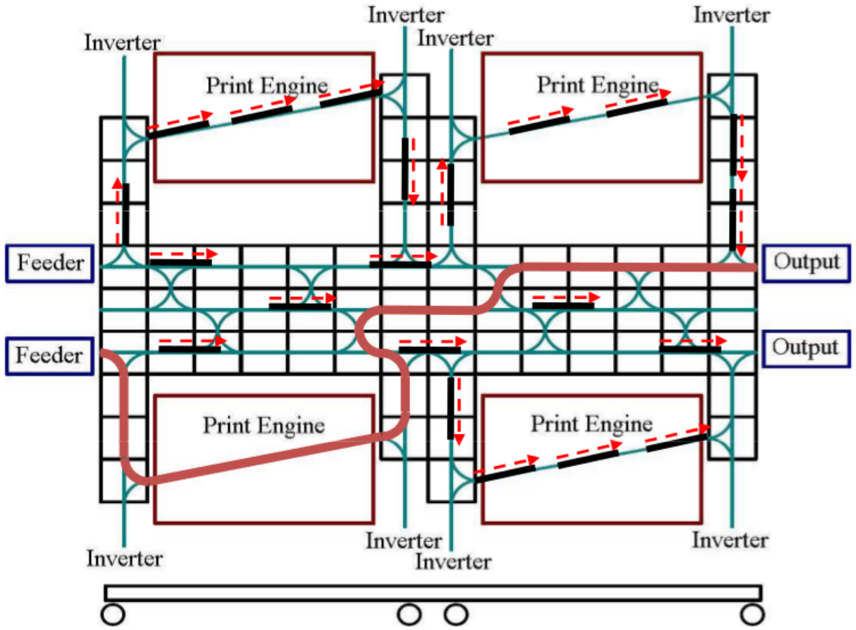
Large-Scale Printing Systems: Complex stuff already . . .



- Process blank sheets of paper into anything (book/bill in folded envelope, . . .).
- Hundreds of independently controlled processing components.
- Dozens of different processes active at any one time.
- Online problem, new jobs come in as we go.

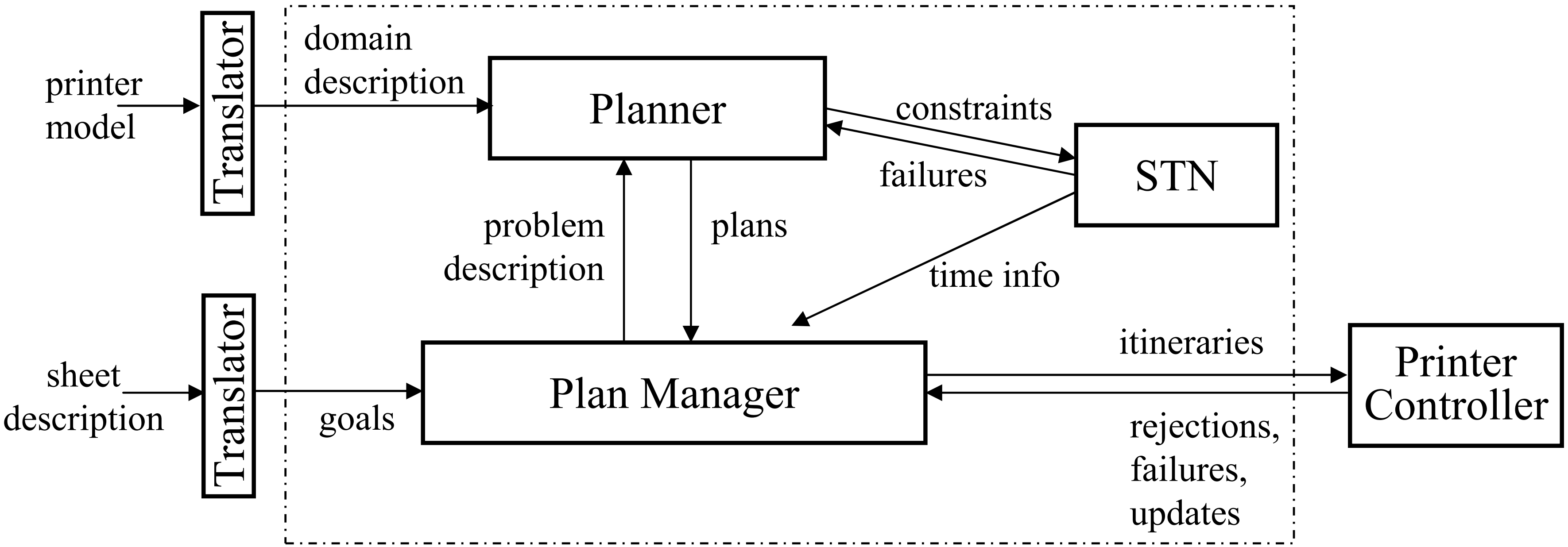
. . . and now we're making it MUCH worse!

MODULAR Large-Scale Printing Systems:



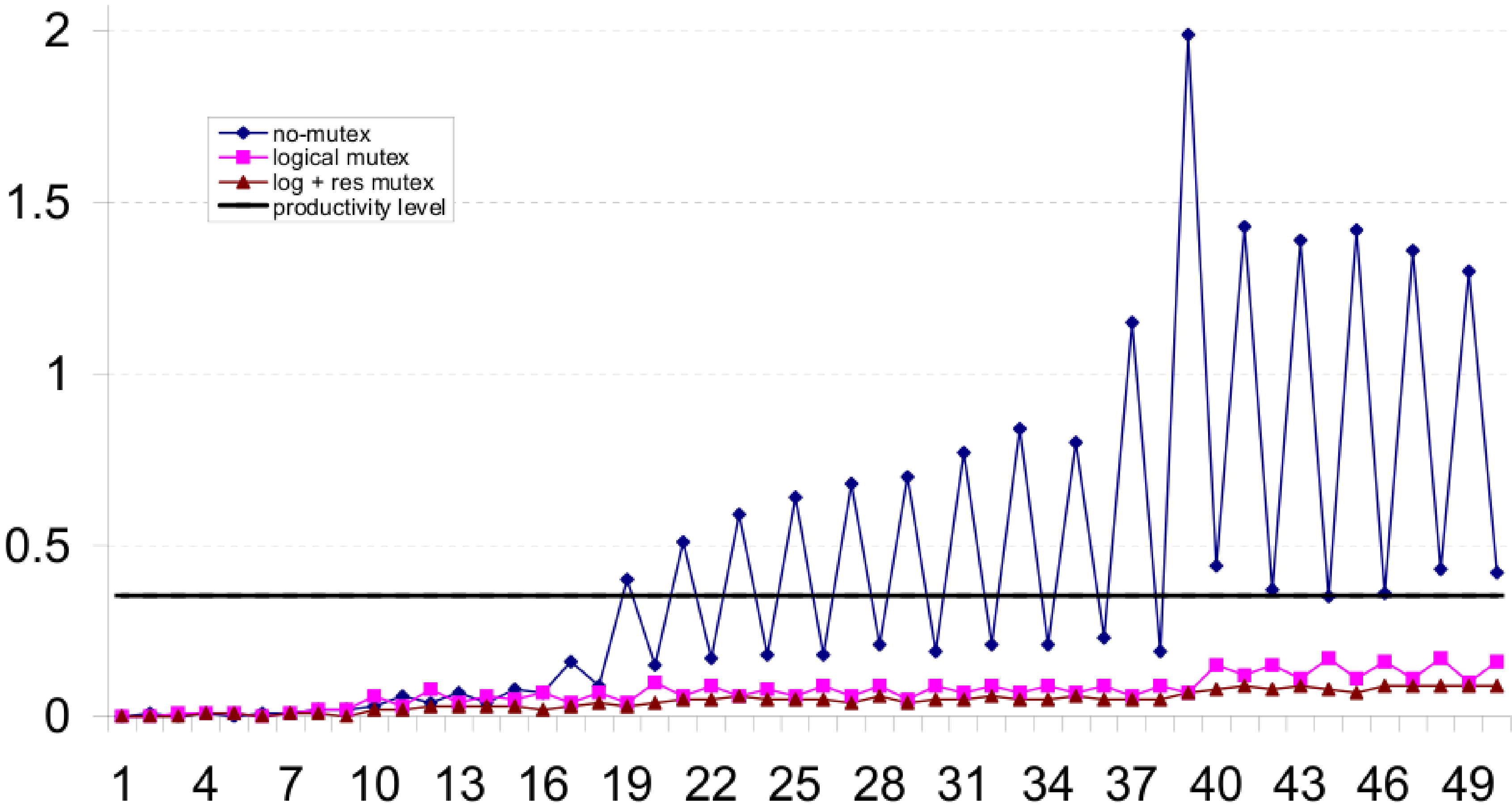
- Assemble and configure components as required by customer.
- No need to buy stuff you don't want, easy to adapt as needed.
- Control can no longer be pre-programmed/configured for a particular machine.
- Requires flexible software that can control anything we could build!

Planning To the Rescue!



- "Planner" as opposed to "Plan Manager": Finding a solution for the task at any given point in time, vs. managing the updates to the task (new jobs arriving, job cancelled due to paper jam, ...).
- "STN": Simple Temporal Network. A constraint-based representation of action durations and precedence constraints, identifying unresolvable conflicts.
- The rest should be self-explanatory ...

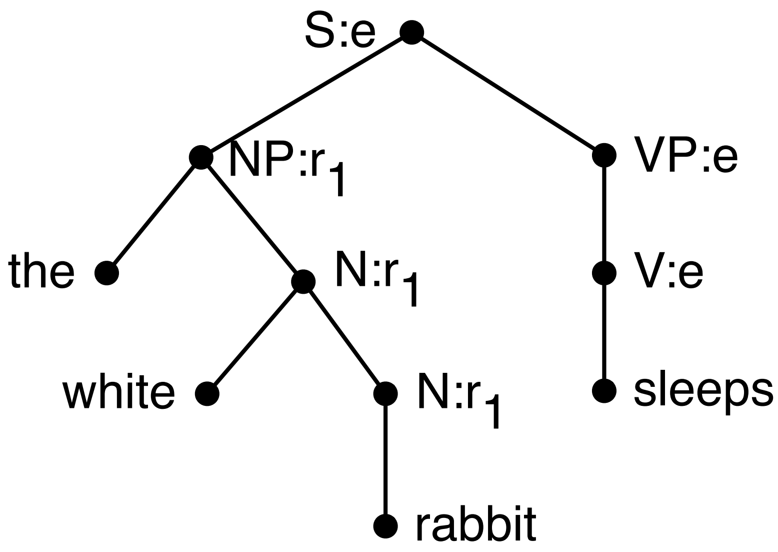
Empirical Performance



- x-axis: jobs come in during online processing; y-axis: runtime (seconds) for planning the new job; productivity level: runtime needed for practicability.
- "no mutex": without h^2 heuristic function.
  $\rightarrow$ h^2 is the key element making this work!

Current/Future FAI Research

Natural Language Generation (NLG)



- **Input:** Grammar, intended meaning.
- **Output:** Sentence implementing meaning.

NLG as Planning, Remarks

Historical:

- Long-standing historical connection between NLG and Planning (first mentioned in early 80s).
- Resurrected in 2007, after long silence, thanks to efficiency of heuristic search planners like FF [**?**].
- Encoding below proposed by [**?**].

Main advantages of planning in this application:

- Rapid development (try to develop a language generator yourself . . .).
- Flexibility (grammar/knowledge changes handled automatically).
- Seamless combination with other tasks (like text planning).

NLG with TAG

(Model-based) NLG in General:

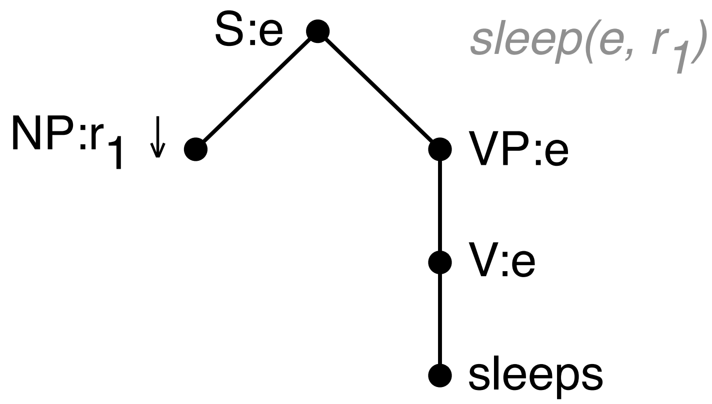
- Given semantic representation (formula) and grammar, compute sentence that expresses this semantics.
- Standard problem in natural language processing, many different approaches exist.

NLG here:

- NLG with tree-adjoining grammars (TAG) [**?**].
- Grammar given in form of finite set of elementary trees.
- Problem instance given by grammar, knowledge base, and a set of ground atoms which the sentence should express.

NLG with TAG: Example

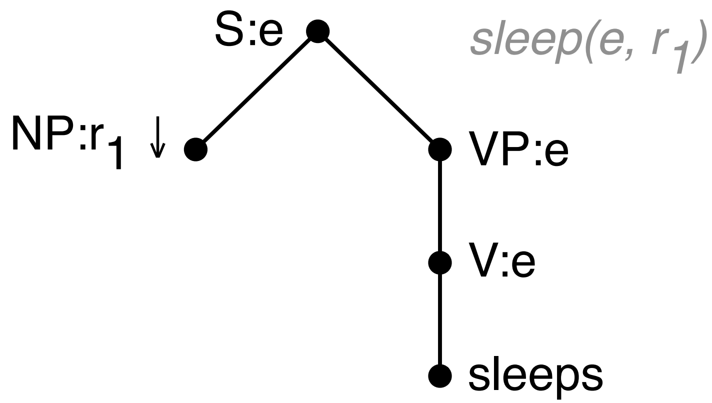
Task: Express ground atom $\{sleep(e, r_1)\}$.
Knowledge Base: $\{sleep(e, r_1), \; rabbit(r_1), \; white(r_1), \; rabbit(r_2)\}$.



- "S:e" stands for sentence referring to event e.
- "NP:$r_1 \downarrow$" stands for a noun phrase referring to r_1, which must be substituted here.
- ["VP:e" and "V:e" stand for a verb phrase referring to e, and can be used to adjoin further trees (not detailed here).]

NLG with TAG: Example, ctd.

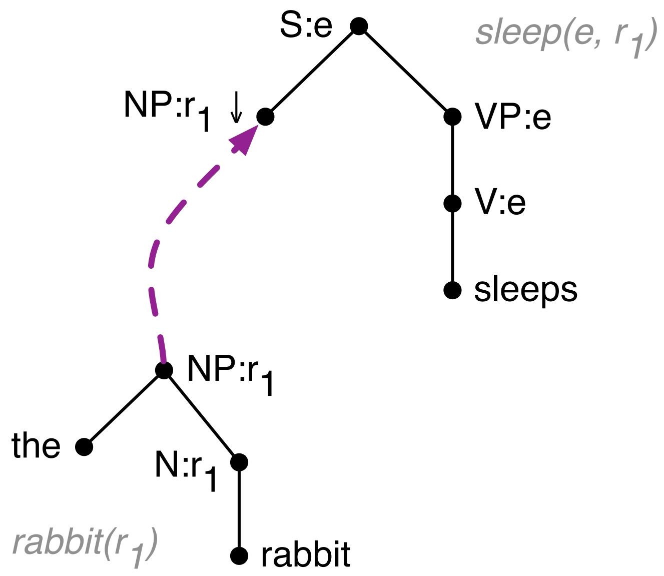
Task: Express ground atom $\{sleep(e, r_1)\}$.
Knowledge Base: $\{sleep(e, r_1),\ rabbit(r_1),\ white(r_1),\ rabbit(r_2)\}$.



- Is this a complete sentence derivation?

NLG with TAG: Example, ctd.

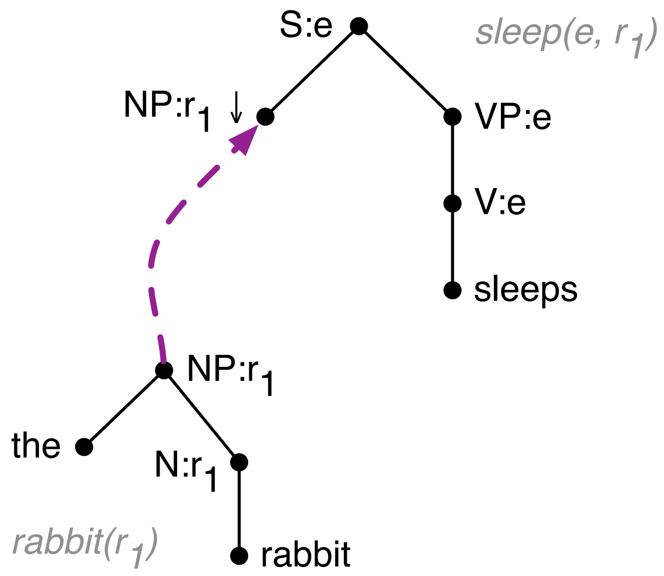
Task: Express ground atom $\{sleep(e, r_1)\}$.
Knowledge Base: $\{sleep(e, r_1), rabbit(r_1), white(r_1), rabbit(r_2)\}$.



- This is a substitution operation (purple dashed arrow in our illustration).
- "N:r_1" stands for a noun-phrase element referring to r_1, and can be used to adjoin further trees.

NLG with TAG: Example, ctd.

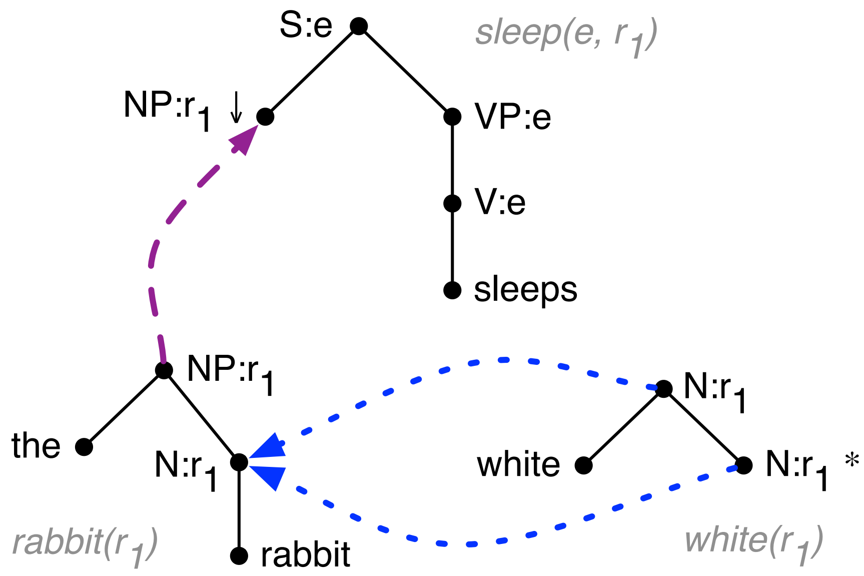
Task: Express ground atom $\{sleep(e, r_1)\}$.
Knowledge Base: $\{sleep(e, r_1),\ rabbit(r_1),\ white(r_1),\ rabbit(r_2)\}$.



- Is this a complete sentence derivation?
- Does the sentence express the desired meaning?

NLG with TAG: Example, ctd.

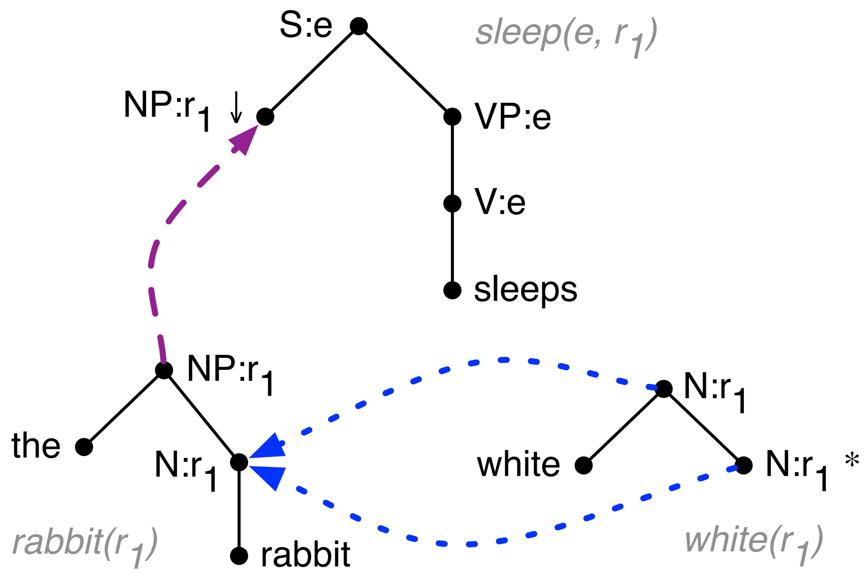
Task: Express ground atom $\{sleep(e, r_1)\}$.
Knowledge Base: $\{sleep(e, r_1), rabbit(r_1), white(r_1), rabbit(r_2)\}$.



- This is an adjunction operation (blue dotted arrow in our illustration).
- "N:r_1" stands for a noun-phrase element referring to r_1, and can be used to adjoin further trees.

NLG with TAG: Example, ctd.

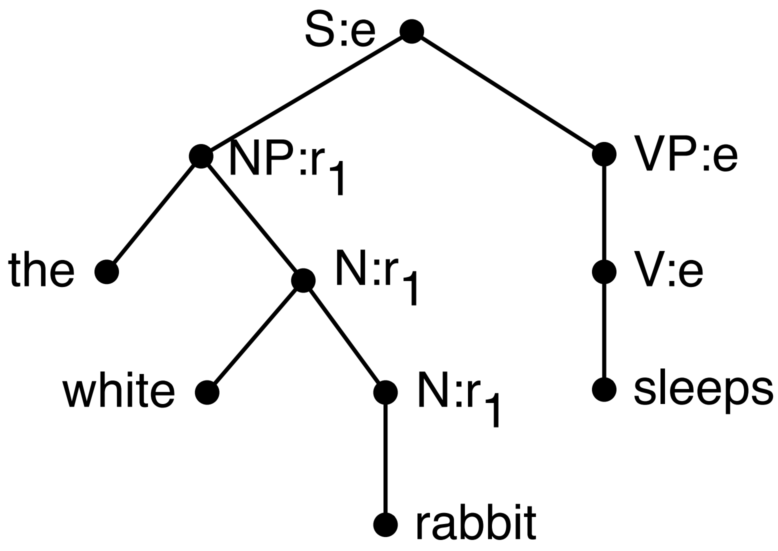
Task: Express ground atom $\{sleep(e, r_1)\}$.
Knowledge Base: $\{sleep(e, r_1),\ rabbit(r_1),\ white(r_1),\ rabbit(r_2)\}$.



- Is this a complete sentence derivation?
- Does the sentence express the desired meaning?

NLG with TAG: Example, ctd.

Task: Express ground atom $\{sleep(e, r_1)\}$.
Knowledge Base: $\{sleep(e, r_1), rabbit(r_1), white(r_1), rabbit(r_2)\}$.



- The outcome of our substitution and adjunction operations here.
- To obtain the desired sentence, read off the leaves from left to right.

. . . and now in PDDL!

From [?], slightly simplified:

sleeps(u, u', x, x'):

pre: $subst(S, u)$, $ref(u, x)$, $sleep(x, x')$

eff: $expressed(sleep, x, x')$, $\neg subst(S, u)$, $subst(NP, u')$, $ref(u', x')$, $\forall y. y \neq x' \rightarrow distractor(u', y)$

"u, u'": nodes in grammar trees
"x": event
"x'": sentence subject

rabbit(u', x'):

pre: $subst(NP, u')$, $ref(u', x')$, $rabbit(x')$

eff: $\neg subst(NP, u')$, $canadjoin(N, u')$, $\forall y. \neg rabbit(y) \rightarrow \neg distractor(u', y)$

white(u', x'):

pre: $canadjoin(N, u')$, $ref(u', x')$, $white(x')$

eff: $\forall y. \neg white(y) \rightarrow \neg distractor(u', y)$

Initial state: $subst(S, u_0)$, $ref(u_0, e)$, $sleep(e, r_1)$, $rabbit(r_1), \dots$

Goal: $expressed(sleep, e, r_1)$
$\forall u \forall x. \neg subst(u, x)$
$\forall u \forall x. \neg distractor(u, x)$

Plan: $\langle$**sleeps**(u_0, u_1, e, r_1), **rabbit**(u_1, r_1), **white**$(u_1, r_1)\rangle$.

Questionnaire

Question!

In the action "sleeps(u, u', x, x')", what for do we need the effect literal "$\neg subst(S, u)$"?

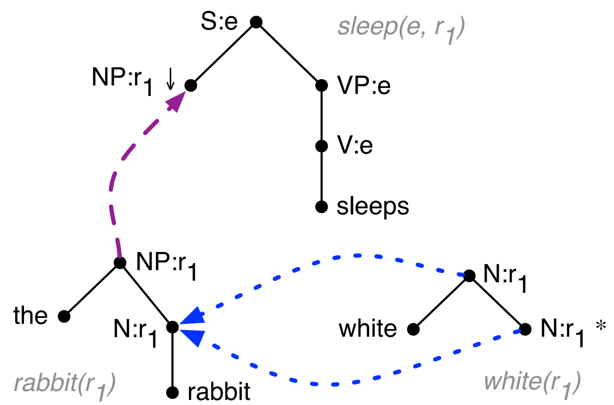
(A): So we don't fall asleep.

(B): So the rabbit does not fall asleep.

(C): To mark the subject of S as being open.

(D): To mark S itself as closed.

sleeps(u, u', x, x'):
pre: $subst(S, u)$, $ref(u, x)$, $sleep(x, x')$
eff: $expressed(sleep, x, x')$, $\color{red}{\neg subst(S, u)}$,
$subst(NP, u')$, $ref(u', x')$,
$\forall y.y \neq x' \rightarrow distractor(u', y)$



Questionnaire, ctd.

Question!

When we apply the action "sleeps(u_0, u_1, e, r_1)" in our plan, what does "u_1" stand for?

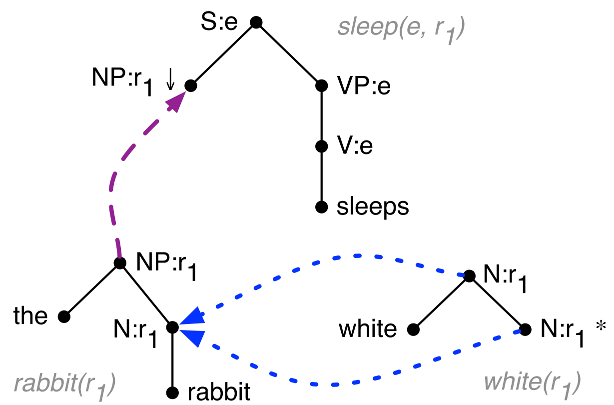
(A): The verb phrase.

(B): The noun phrase.

(C): The node "NP:$r_1 \downarrow$" in the verb-phrase tree.

(D): The tree representing the noun phrase.

sleeps(u, u', x, x'):
pre: $subst(S, u)$, $ref(u, x)$, $sleep(x, x')$
eff: $expressed(sleep, x, x')$, $\neg subst(S, u)$, $subst(NP, u')$, $ref(u', x')$, $\forall y.y \neq x' \rightarrow distractor(u', y)$



FAI Research

Summary

Ambition:

Write one program (planner) that can solve all sequential decision-making problems.

Trade-off: generality vs. efficiency

We have seen the simplest form of planning:

- Underlying formalism: STRIPS
- Language for Tools: PDDL

Summary

- Thanks to the efficiency of heuristic search planning techniques, planning is being applied in a broad variety of applications today.
- Simulated penetration testing is used for regular network security checks, and is commercially employed with FF as the underlying planner. http://fai.cs.uni-saarland.de/hoffmann/papers/icaps15inv.pdf
- Flexible printer system control is required for large-scale configurable printing systems, and can be successfully tackled using a temporal variant of the planning heuristic h^2. https://jair.org/index.php/jair/article/view/10693
- Natural language generation involves constructing sentences, and can be successfully encoded into PDDL using FF. http://fai.cs.uni-saarland.de/hoffmann/papers/icaps10.pdf

Further Reading

There is a book on PDDL:

An Introduction to the Planning Domain Definition Language

`http://www.morganclaypoolpublishers.com/catalog_Orig/product_info.php?products_id=1384`

Remarks

There's quite a range of **further application areas:**

- Greenhouse logistics involves moving a series of conveyor belts to cater for the needs of all the plants [**?**].
- Plan recognition involves observing (some of) the actions of an agent, and inferring what the goal is [**?**].
- Business process management involves creating, maintaining, and executing complex processes across large enterprises; planning can be used to automatically generate process templates [**?**].
- Software model checking involves (amongst others) finding bugs; this can be formulated as finding a plan to an error state [**?**].

References I

Fahiem Bacchus. *Subset of PDDL for the AIPS2000 Planning Competition*. The AIPS-00 Planning Competition Comitee, 2000.

Mark Boddy, Jonathan Gohde, Tom Haigh, and Steven Harp. Course of action generation for cyber security using classical planning. In Susanne Biundo, Karen Myers, and Kanna Rajan, editors, *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 12–21, Monterey, CA, USA, 2005. AAAI Press.

Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In S. Steel and R. Alami, editors, *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 221–233. Springer-Verlag, 1997.

References II

Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

Malte Helmert and Hauke Lasinger. The Scanalyzer domain: Greenhouse logistics as a planning problem. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 234–237. AAAI Press, 2010.

Jörg Hoffmann and Stefan Edelkamp. The deterministic part of ipc-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Jörg Hoffmann, Ingo Weber, and Frank Michael Kraft. SAP speaks PDDL: Exploiting a software-engineering model for planning in business process management. *Journal of Artificial Intelligence Research*, 44:587–632, 2012.

References III

Alexander Koller and Jörg Hoffmann. Waking up a sleeping rabbit: On natural-language sentence generation with FF. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 238–241. AAAI Press, 2010.

Alexander Koller and Matthew Stone. Sentence generation as a planning problem. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007)*, pages 336–343. The Association for Computational Linguistics, 2007.

Sebastian Kupferschmid, Jörg Hoffmann, Henning Dierks, and Gerd Behrmann. Adapting an AI planning heuristic for directed model checking. In Antti Valmari, editor, *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, pages 35–52. Springer-Verlag, 2006.

Jorge Lucangeli, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. In *Proceedings of the 2nd Workshop on Intelligent Security (SecArt'10)*, 2010.

References IV

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee, 1998.

Miquel Ramírez and Hector Geffner. Plan recognition as planning. In Craig Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 1778–1783, Pasadena, California, USA, July 2009. Morgan Kaufmann.