## Principles of AI Planning

November 3rd, 2006 - Transition systems

Transition systems
Definition
Example
Matrices
Reachability
Algorithm
Succinct transition systems
State variables
Propositional logic
Operators
Schematic operators

# Principles of AI Planning <br> Transition systems 

Malte Helmert Bernhard Nebel

Albert-Ludwigs-Universität Freiburg
November 3rd, 2006

## Transition systems



## Transition systems

Formalization of the dynamics of the world/application

## Definition

A transition system is $\left\langle S, I,\left\{a_{1}, \ldots, a_{n}\right\}, G\right\rangle$ where

- $S$ is a finite set of states (the state space),
- $I \subseteq S$ is a finite set of initial states,
- every action $a_{i} \subseteq S \times S$ is a binary relation on $S$,
- $G \subseteq S$ is a finite set of goal states.


## Definition

An action $a$ is applicable in a state $s$ if $s a s^{\prime}$ for at least one state $s^{\prime}$.

## Transition systems

Deterministic transition systems
A transition system is deterministic if there is only one initial state and all actions are deterministic. Hence all future states of the world are completely predictable.

## Definition

A deterministic transition system is $\langle S, I, O, G\rangle$ where

- $S$ is a finite set of states (the state space),
- $I \in S$ is a state,
- actions $a \in O$ (with $a \subseteq S \times S$ ) are partial functions,
- $G \subseteq S$ is a finite set of goal states.

Successor state wrt. an action
Given a state $s$ and an action $A$ so that $a$ is applicable in $s$, the successor state of $s$ with respect to $a$ is $s^{\prime}$ such that sas', denoted by $s^{\prime}=a p p_{a}(s)$.

## Blocks world

The rules of the game

Location on the table does not matter.


Location on a block does not matter.


## Blocks world

The rules of the game
At most one block may be below a block.


At most one block may be on top of a block.

## Blocks world

The transition graph for three blocks


## Blocks world

Properties

| blocks | states |
| ---: | ---: |
| 1 | 1 |
| 2 | 3 |
| 3 | 13 |
| 4 | 73 |
| 5 | 501 |
| 6 | 4051 |
| 7 | 37633 |
| 8 | 394353 |
| 9 | 4596553 |
| 10 | 58941091 |

1. Finding a solution is polynomial time in the number of blocks (move everything onto the table and then construct the goal configuration).
2. Finding a shortest solution is NP-complete (for a compact description of the problem).

## Deterministic planning: plans

## Definition

A plan for $\langle S, I, O, G\rangle$ is a sequence $\pi=o_{1}, \ldots, o_{n}$ of operators such that $o_{1}, \ldots, o_{n} \in O$ and $s_{0}, \ldots, s_{n}$ is a sequence of states (the execution of $\pi$ ) so that

1. $s_{0}=l$,
2. $s_{i}=a p p_{o_{i}}\left(s_{i-1}\right)$ for every $i \in\{1, \ldots, n\}$, and
3. $s_{n} \in G$.

This can be equivalently expressed as

$$
\operatorname{app}_{o_{n}}\left(\operatorname{app}_{o_{n-1}}\left(\cdots a p p_{o_{1}}(I) \cdots\right)\right) \in G
$$

## Transition relations as matrices

1. If there are $n$ states, each action (a binary relation) corresponds to an $n \times n$ matrix: The element at row $i$ and column $j$ is 1 if the action maps state $i$ to state $j$, and 0 otherwise.
For deterministic actions there is at most one non-zero element in each row.
2. Matrix multiplication corresponds to sequential composition: taking action $M_{1}$ followed by action $M_{2}$ is the product $M_{1} M_{2}$. (This also corresponds to the join of the relations.)
3. The unit matrix $I_{n \times n}$ is the NO-OP action: every state is mapped to itself.

## Example



## Example



## Example

Sum matrix $M_{R}+M_{G}+M_{B}$
Representing one-step reachability by any of the component actions


|  | $A$ | $B$ | $C$ | $D$ | $E$ | $F$ |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| $A$ | 0 | 1 | 0 | 0 | 0 | 0 |
| $B$ | 0 | 0 | 0 | 0 | 0 | 1 |
| $C$ | 0 | 1 | 1 | 0 | 0 | 1 |
| $D$ | 1 | 0 | 1 | 0 | 1 | 0 |
| $E$ | 0 | 1 | 0 | 1 | 0 | 0 |
| $F$ | 1 | 0 | 0 | 0 | 1 | 0 |

We use addition $0+0=0$ and $b+b^{\prime}=1$ if $b=1$ or $b^{\prime}=1$.

## Sequential composition as matrix multiplication

$$
\left(\begin{array}{llllll}
0 & 1 & 0 & 0 & 0 & 0 \\
\hline 0 & 0 & 0 & 0 & 0 & 1 \\
\hline 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0
\end{array}\right) \times\left(\begin{array}{lllll|ll}
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0
\end{array}\right)=\left(\begin{array}{llllll}
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 0
\end{array}\right)
$$

E is reachable from B by two actions because
$F$ is reachable from $B$ by one action and
$E$ is reachable from $F$ by one action.

## Reachability

Let $M$ be the $n \times n$ matrix that is the (Boolean) sum of the matrices of the individual actions. Define

$$
\begin{aligned}
& R_{0}=I_{n \times n} \\
& R_{1}=I_{n \times n}+M \\
& R_{2}=I_{n \times n}+M+M^{2} \\
& R_{3}=I_{n \times n}+M+M^{2}+M^{3}
\end{aligned}
$$

$R_{i}$ represents reachability by $i$ actions or less. If $s^{\prime}$ is reachable from $s$, then it is reachable with $\leq n-1$ actions: $R_{n-1}=R_{n}$.

## Reachability: example, $M_{R}$



Reachability: example, $M_{R}+M_{R}^{2}$


Reachability: example, $M_{R}+M_{R}^{2}+M_{R}^{3}$


Reachability: example, $M_{R}+M_{R}^{2}+M_{R}^{3}+I_{6 \times 6}$


## Relations and sets as matrices

Row vectors as sets of states

Row vectors $S$ represent sets. $S M$ is the set of states reachable from $S$ by $M$.

$$
\left(\begin{array}{l}
1 \\
0 \\
1 \\
0 \\
0 \\
0
\end{array}\right)^{T} \times\left(\begin{array}{llllll}
1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1
\end{array}\right)=\left(\begin{array}{l}
1 \\
1 \\
1 \\
0 \\
1 \\
1
\end{array}\right)^{T}
$$

## A simple planning algorithm

- We next present a simple planning algorithm based on computing distances in the transition graph.
- The algorithm finds shortest paths less efficiently than Dijkstra's algorithm; we present the algorithm because we later will use it as a basis of an algorithm that is applicable to much bigger state spaces than Dijkstra's algorithm directly.


## A simple planning algorithm

 Idea

## A simple planning algorithm

1. Compute the matrices $R_{0}, R_{1}, R_{2}, \ldots, R_{n}$ representing reachability with $0,1,2, \ldots, n$ steps with all actions.
2. Find the smallest $i$ such that a goal state $s_{g}$ is reachable from the initial state according to $R_{i}$.
3. Find an action (the last action of the plan) by which $s_{g}$ is reached with one step from a state $s_{g^{\prime}}$ that is reachable from the initial state according to $R_{i-1}$.
4. Repeat the last step, now viewing $s_{g^{\prime}}$ as the goal state with distance $i-1$.

## Example



|  | A | $B$ | $C$ | D |  |  | A | $B$ | C | $D$ |  |  | A | $B$ | C | $D$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| A | 0 | 1 | 0 | 0 |  | A | 0 | 1 | 0 | 0 |  | A | 0 | 1 | 0 | 0 |
| $B$ | 0 | 0 | 0 | 0 | $+$ | $B$ | 0 | 0 | 1 | 0 | $=$ | $B$ | 0 | 0 | 1 | 0 |
| C | 0 | 0 | 0 | 1 |  | C | 1 | 0 | 0 | 0 |  | C | 1 | 0 | 0 | 1 |
| D | 0 | 0 | 0 | 0 |  | D | 0 | 0 | 0 | 0 |  | D | 0 | 0 | 0 | 0 |

## Example



## Succinct representation of transition systems

- More compact representation of actions than as relations is often

1. possible because of symmetries and other regularities,
2. unavoidable because the relations are too big.

- Represent different aspects of the world in terms of different state variables. $\Longrightarrow A$ state is a valuation of state variables.
- Represent actions in terms of changes to the state variables.


## State variables

- The state of the world is described in terms of a finite set of finite-valued state variables.


## Example

HOUR : $\{0, \ldots, 23\}=13$
MINUTE : $\{0, \ldots, 59\}=55$
LOCATION : $\{51,52,82,101,102\}=101$
WEATHER : $\{$ sunny, cloudy, rainy $\}=$ cloudy
HOLIDAY : $\{\mathrm{T}, \mathrm{F}\}=\mathrm{F}$

- Any $n$-valued state variable can be replaced by $\left\lceil\log _{2} n\right\rceil$ Boolean (2-valued) state variables.
- Actions change the values of the state variables.


## Blocks world with state variables

State variables:
LOCATIONofA : $\{B, C$, TABLE $\}$
LOCATIONofB : $\{A, C$, TABLE $\}$
LOCATIONofC : $\{A, B$, TABLE $\}$
Example
$s($ LOCATIONofA $)=$ TABLE
$s($ LOCATIONofB $)=A$
$s($ LOCATIONofC $)=$ TABLE


Not all valuations correspond to an intended blocks world state, e.g. s such that $s($ LOCATIONofA $)=B$ and $s($ LOCATIONofB $)=A$.

## Blocks world with Boolean state variables

## Example

$$
\begin{array}{lll}
s(\text { Aon } B)=0 & s(\text { Aon } C)=0 & s(\text { AonTABLE })=1 \\
s(\text { Bon } A)=1 & s(\text { Bon } C)=0 & s(\text { BonTABLE })=0 \\
s(\text { Con } A)=0 & s(\text { Con } B)=0 & s(\text { ConTABLE })=1
\end{array}
$$



## Logical representations of state sets

- $n$ state variables with $m$ values induce a state space consisting of $m^{n}$ states ( $2^{n}$ states for $n$ Boolean state variables).
- A language for talking about sets of states (valuations of state variables) is propositional logic.
- Logical connectives correspond to set-theoretical operations.
- Logical relations correspond to set-theoretical relations.


## Propositional logic

Let $A$ be a set of atomic propositions ( $\sim$ state variables.)

1. For all $a \in A, a$ is a propositional formula.
2. If $\phi$ is a propositional formula, then so is $\neg \phi$.
3. If $\phi$ and $\phi^{\prime}$ are propositional formulae, then so is $\phi \vee \phi^{\prime}$.
4. If $\phi$ and $\phi^{\prime}$ are propositional formulae, then so is $\phi \wedge \phi^{\prime}$.
5. The symbols $\perp$ and $T$ are propositional formulae.

The implication $\phi \rightarrow \phi^{\prime}$ is an abbreviation for $\neg \phi \vee \phi^{\prime}$.
The equivalence $\phi \leftrightarrow \phi^{\prime}$ is an abbreviation for $\left(\phi \rightarrow \phi^{\prime}\right) \wedge\left(\phi^{\prime} \rightarrow \phi\right)$.

## Propositional logic

## Valuations and truth

A valuation of $A$ is a function $v: A \rightarrow\{0,1\}$. Define the notation $v \models \phi$ for valuations $v$ and formulae $\phi$ by

1. $v \models a$ if and only if $v(a)=1$, for $a \in A$.
2. $v \vDash \neg \phi$ if and only if $v \not \vDash \phi$
3. $v \models \phi \vee \phi^{\prime}$ if and only if $v \models \phi$ or $v \models \phi^{\prime}$
4. $v \models \phi \wedge \phi^{\prime}$ if and only if $v \models \phi$ and $v \models \phi^{\prime}$
5. $v \vDash T$
6. $v \not \vDash \perp$

## Propositional logic

## Some terminology

- A propositional formula $\phi$ is satisfiable if there is at least one valuation $v$ so that $v \models \phi$. Otherwise it is unsatisfiable.
- A propositional formula $\phi$ is valid or a tautology if $v \models \phi$ for all valuations $v$. We write this as $\models \phi$.
- A propositional formula $\phi$ is a logical consequence of a propositional formula $\phi^{\prime}$, written $\phi^{\prime} \models \phi$, if $v \models \phi$ for all valuations $v$ such that $v \vDash \phi^{\prime}$.
- A propositional formula that is a proposition a or a negated proposition $\neg a$ for some $a \in A$ is a literal.
- A formula that is a disjunction of literals is a clause.


## Formulae vs. sets

| sets | formulae |
| :--- | :--- |
| those $\frac{2^{n}}{2}$ states in which $a$ is true | $a \in A$ |
| $E \cup F$ | $E \vee F$ |
| $E \cap F$ | $E \wedge F$ |
| $\frac{E}{} \backslash F$ | (set difference) |
| $\frac{E}{E}$ | $E \wedge \neg F$ |
|  | $\neg E$ |
| the empty set $\emptyset$ |  |
| the universal set | $\top$ |
|  |  |
| question about sets | question about formulae |
| $E \subseteq F ?$ | $E \models F ?$ |
| $E \subset F ?$ | $E \models F$ and $F \not \models E ?$ |
| $E=F ?$ | $E \models F$ and $F \models E ?$ |

## Operators

Actions are represented as operators $\langle c, e\rangle$ where

- $c$ (the precondition) is a propositional formula over $A$ describing the set of states in which the action can be taken. (States in which an arrow starts.)
- e (the effect) describes the successor states of states in which the action can be taken. (Where do the arrows go.)
The description is procedural: how do the values of the state variable change?


## Effects

For deterministic operators

## Definition

Effects are recursively defined as follows.

1. $a$ and $\neg a$ for state variables $a \in A$ are effects.
2. $e_{1} \wedge \cdots \wedge e_{n}$ is an effect if $e_{1}, \ldots, e_{n}$ are effects (the special case with $n=0$ is the empty conjunction $T$.)
3. $c \triangleright e$ is an effect if $c$ is a formula and $e$ is an effect.

Atomic effects $a$ and $\neg a$ are best understood respectively as assignments $a:=1$ and $a:=0$.

## Effects

Meaning of conditional effects $\triangleright$
$c \triangleright e$ means that change $e$ takes place if $c$ is true in the current state.

## Example

Increment 4-bit numbers $b_{3} b_{2} b_{1} b_{0}$.

$$
\begin{aligned}
&\left(\neg b_{0} \triangleright b_{0}\right) \wedge \\
&\left(\left(\neg b_{1} \wedge b_{0}\right) \triangleright\left(b_{1} \wedge \neg b_{0}\right)\right) \wedge \\
&\left(\left(\neg b_{2} \wedge b_{1} \wedge b_{0}\right) \triangleright\left(b_{2} \wedge \neg b_{1} \wedge \neg b_{0}\right)\right) \wedge \\
&\left(\left(\neg b_{3} \wedge b_{2} \wedge b_{1} \wedge b_{0}\right) \triangleright\left(b_{3} \wedge \neg b_{2} \wedge \neg b_{1} \wedge \neg b_{0}\right)\right)
\end{aligned}
$$

## Example: operators for blocks world

In addition to state variables likes $A o n T$ and $B o n C$, for convenience we also use state variables Aclear, Bclear, and Cclear to denote that there is nothing on the block in question.
$\langle$ Aclear $\wedge$ Aon $T \wedge$ Bclear, $A o n B \wedge \neg$ Aon $T \wedge \neg$ Bclear $\rangle$
$\langle$ Aclear $\wedge$ Aon $T \wedge$ Cclear, Aon $C \wedge \neg$ Aon $T \wedge \neg$ Cclear $\rangle$
$\langle$ Aclear $\wedge$ AonB, Aon $T \wedge \neg$ Aon $B \wedge$ Bclear $\rangle$
$\langle$ Aclear $\wedge$ AonC, Aon $T \wedge \neg$ AonC $\wedge$ Cclear $\rangle$
$\langle$ Bclear $\wedge$ BonA, BonT $\wedge \neg$ BonA $\wedge$ Aclear $\rangle$
$\langle$ Bclear $\wedge$ BonC, BonT $\wedge \neg$ BonC $\wedge$ Cclear $\rangle$

## Operators: meaning

Changes caused by an operator
Assign to each effect $e$ and state $s$ a set $[e]_{s}$ of literals as follows.

1. $[a]_{s}=\{a\}$ and $[\neg a]_{s}=\{\neg a\}$ for $a \in A$.
2. $\left[e_{1} \wedge \cdots \wedge e_{n}\right]_{s}=\left[e_{1}\right]_{s} \cup \ldots \cup\left[e_{n}\right]_{s}$.
3. $[c \triangleright e]_{s}=[e]_{s}$ if $s \neq c$ and $[c \triangleright e]_{s}=\emptyset$ otherwise.

Applicability of an operator
Operator $\langle c, e\rangle$ is applicable in a state $s$ iff $s \models c$ and $[e]_{s}$ is consistent.

## Operators: the successor state of a state

Definition (Successor state)
The successor state appo(s) of $s$ with respect to operator $o=\langle c, e\rangle$ is obtained from $s$ by making literals in $[e]_{s}$ true.
This is defined only if $o$ is applicable in $s$.

## Example

Consider the operator $\langle a, \neg a \wedge(\neg c \triangleright \neg b)\rangle$ and a state $s$ such that $s \neq a \wedge b \wedge c$.
The operator is applicable because $s \models a$ and $[\neg a \wedge(\neg c \triangleright \neg b)]_{s}=\{\neg a\}$ is consistent.
Hence $\operatorname{app}_{\langle a, \neg a \wedge(\neg c \triangleright \neg b)\rangle}(s) \models \neg a \wedge b \wedge c$.

## Operators

## Example

State variables: $A=\{a, b, c\}$.
An operator is

$$
\begin{aligned}
& \langle(b \wedge c) \vee(\neg a \wedge b \wedge \neg c) \vee(\neg a \wedge c), \\
& ((b \wedge c) \triangleright \neg c) \\
& \wedge(\neg b \triangleright(a \wedge b)) \\
& \wedge(\neg c \triangleright a)\rangle
\end{aligned}
$$



## Succinct transition systems

Deterministic case

## Definition

A succinct deterministic transition system is $\left\langle A, I,\left\{o_{1}, \ldots, o_{n}\right\}, G\right\rangle$ where

- $A$ is a finite set of state variables,
- $I$ is an initial state,
- every $o_{i}$ is an operator,
- $G$ is a formula describing the goal states.


## Mapping from succinct TS to TS

From every succinct transition system $\langle A, I, O, G\rangle$ we can produce a corresponding transition system $\left\langle S, I, O^{\prime}, G^{\prime}\right\rangle$.

1. $S$ is the set of all valuations of $A$,
2. $O^{\prime}=\{R(o) \mid o \in O\}$ where $R(o)=\left\{\left(s, s^{\prime}\right) \in S \times S \mid s^{\prime}=a p p_{o}(s)\right\}$, and
3. $G^{\prime}=\{s \in S \mid s \models G\}$.

## Schematic operators

- Description of state variables and operators in terms of a given finite set of objects.
- Analogy: propositional logic vs. predicate logic
- Planners take input as schematic operators, and translate them into (ground) operators. This is called grounding.


## Schematic operators: example

Schematic operator

$$
\begin{aligned}
& x \in\{\text { car1, car2 }\} \\
& y_{1} \in\{\text { Freiburg, Strassburg }\}, \\
& y_{2} \in\{\text { Freiburg, Strassburg }\}, y_{1} \neq y_{2} \\
& \left\langle\operatorname{in}\left(x, y_{1}\right), \operatorname{in}\left(x, y_{2}\right) \wedge \neg \operatorname{in}\left(x, y_{1}\right)\right\rangle
\end{aligned}
$$

corresponds to the operators
$\langle$ in(car1, Freiburg), in(car1, Strassburg) $\wedge \neg$ in(car1, Freiburg) $\rangle$, $\langle$ in(car1, Strassburg), in(car1, Freiburg) $\wedge \neg$ in(car1, Strassburg) $\rangle$, $\langle$ in(car2, Freiburg), in(car2, Strassburg) $\wedge \neg$ in(car2, Freiburg) $\rangle$, $\langle$ in(car2, Strassburg), in(car2, Freiburg) $\wedge \neg$ in(car2, Strassburg) $\rangle$

## Schematic operators: quantification

Existential quantification (for formulae only)
Finite disjunctions $\phi\left(a_{1}\right) \vee \cdots \vee \phi\left(a_{n}\right)$ represented as $\exists x \in\left\{a_{1}, \ldots, a_{n}\right\} \phi(x)$.

Universal quantification (for formulae and effects)
Finite conjunctions $\phi\left(a_{1}\right) \wedge \cdots \wedge \phi\left(a_{n}\right)$ represented as $\forall x \in\left\{a_{1}, \ldots, a_{n}\right\} \phi(x)$.

## Example

$\exists x \in\{A, B, C\}$ in $(x$, Freiburg $)$ is a short-hand for in $(A$, Freiburg $) \vee$ in $(B$, Freiburg $) \vee$ in $(C$, Freiburg $)$.

## PDDL: the Planning Domain Description Language

- Used by almost all implemented systems for deterministic planning.
- Supports a language comparable to what we have defined above (including schematic operators and quantification).
- Syntax inspired by the Lisp programming language: e.g. prefix notation for formulae.

```
(and (or (on A B) (on A C))
    (or (on B A) (on B C))
    (or (on C A) (on A B)))
```


## PDDL: domain files

A domain file consists of

- (define (domain DOMAINNAME)
- a :requirements definition (use :adl :typing by default)
- definitions of types (each parameter has a type)
- definitions of predicates
- definitions of operators


## Example: blocks world in PDDL

```
(define (domain BLOCKS)
    (:requirements :adl :typing)
    (:types block - object
        blueblock smallblock - block)
    (:predicates (on ?x - smallblock ?y - block)
        (ontable ?x - block)
        (clear ?x - block)
    )
```


## PDDL: operator definition

- (:action OPERATORNAME
- list of parameters: (?x - type1 ?y - type2 ?z - type3)
- precondition: a formula

```
<schematic-state-var>
(and <formula> ... <formula>)
(or <formula> ... <formula>)
(not <formula>)
(forall (?x1 - type1 ... ?xn - typen) <formula>)
(exists (?x1 - type1 ... ?xn - typen) <formula>)
```

- effect:

```
<schematic-state-var>
(not <schematic-state-var>)
(and <effect> ... <effect>)
(when <formula> <effect>)
(forall (?x1 - type1 ... ?xn - typen) <effect>)
```

(:action fromtable
:parameters (?x - smallblock ?y - block)
:precondition (and (not (= ?x ?y))
(clear ?x)
(ontable ?x)
(clear ?y))
: effect
(and (not (ontable ?x))
(not (clear ?y))
(on ?x ?y)))

## PDDL: problem files

A problem file consists of

- (define (problem PROBLEMNAME)
- declaration of which domain is needed for this problem
- definitions of objects belonging to each type
- definition of the initial state (list of state variables initially true)
- definition of goal states (a formula like operator precondition)

```
(define (problem example)
    (:domain BLOCKS)
    (:objects a b c - smallblock)
            d e - block
        f - blueblock)
    (:init (clear a) (clear b) (clear c)
        (clear d) (clear e) (clear f)
        (ontable a) (ontable b) (ontable c)
        (ontable d) (ontable e) (ontable f))
    (:goal (and (on a d) (on b e) (on c f)))
)
```


## Example run on the FF planner

```
edu/PSO4> ./ff -o blocks-dom.pddl -f blocks-ex.pddl
ff: parsing domain file, domain 'BLOCKS' defined
ff: parsing problem file, problem 'EXAMPLE' defined
ff: found legal plan as follows
step 0: FROMTABLE A D
    1: FROMTABLE B E
    2: FROMTABLE C F
0.01 seconds total time
```


## Example: blocks world in PDDL

```
(define (domain BLOCKS)
    (:requirements :adl :typing)
    (:types block)
    (:predicates (on ?x - block ?y - block)
        (ontable ?x - block)
        (clear ?x - block)
        )
```

```
(:action fromtable
    :parameters (?x - block ?y - block)
    :precondition (and (not (= ?x ?y))
        (clear ?x)
        (ontable ?x)
        (clear ?y))
    :effect
    (and (not (ontable ?x))
        (not (clear ?y))
        (on ?x ?y)))
```


## (:action totable

:parameters (?x - block ?y - block)
:precondition (and (clear ?x) (on ?x ?y))
:effect

```
(and (not (on ?x ?y))
    (clear ?y)
    (ontable ?x)))
```

```
(:action move
    :parameters (?x - block
        ?y - block
        ?z - block)
    :precondition (and (clear ?x) (clear ?z)
        (on ?x ?y) (not (= ?x ?z)))
    :effect
    (and (not (clear ?z))
        (clear ?y)
        (not (on ?x ?y))
        (on ?x ?z)))
```

```
(define (problem blocks-10-0)
    (:domain BLOCKS)
    (:objects d a h g b j e i f c - block)
    (: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))
    (: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)))
)
```

