LU Einführung in Wissensbasierte Systeme

Diagnosis and Planning using DLV

Wolfgang Faber, Hans Tompits

Institut für Informationssysteme

Abteilung Wissensbasierte System (184/3)

Technische Universität Wien

http://www.kr.tuwien.ac.at

Overview

- Reminder: What is planning?
- Formalising planning
- Tour of Language $\mathcal{K}$
- Example: Blocksworld
- Invocation of DLV$^{\mathcal{K}}$

Reminder: What is Planning?

- Starting from a situation
- Try to reach a goal
- By executing some actions

Given the initial situation, a goal, and a domain description, choose the actions such that the goal is achieved.

Planning – Example

- I am at Favoritenstraße at 8:30
- And have to give a lecture at Gusshausstraße at 9:00
- I can pack my lecture material in a bag, exit Favoritenstraße, walk to Gusshausstraße in order to achieve the goal.

Obviously, there could also be other plans (e.g. going by bike if one is available to me), taking a taxi, etc.

Note also that the plan need not always work (e.g. forgot the lecture material at home, Gusshausstraße is locked, etc.)

Planning – How to formalize?

We can identify 3 main elements:

- Static statements “I am at Favoritenstraße.”

  ⇒ *Fluents*

- Action statements “Walk to Gusshausstraße.”

  ⇒ *Actions*

- Time-Independent statements “Favoritenstraße is a location.”

  ⇒ *Background Knowledge*

Planning – How to formalize?

Observations:

1. In general, actions need *preconditions*.
2. Actions usually have *effects*.
3. Often, fluents remain true when unaffected by actions.
4. Sometimes fluents depend (only) on other fluents.

Planning – How to formalize?

Examples for observations:

1. Preconditions: E.g. I can leave Favoritenstraße only if I am currently at Favoritenstraße.
2. Effects: E.g. entering Gusshausstraße has the effect of me being there.
3. E.g. if I am at Favoritenstraße and pack my lecture material in a bag, I am still at Favoritenstraße afterwards.
4. If I am in EI 5, I am at Gusshausstraße.

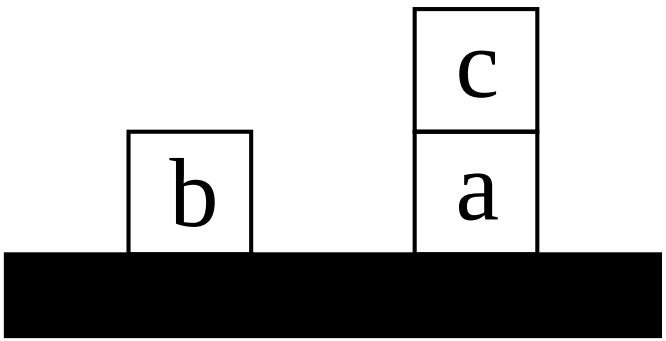
Planning – How to formalize?

Terms for these observations:

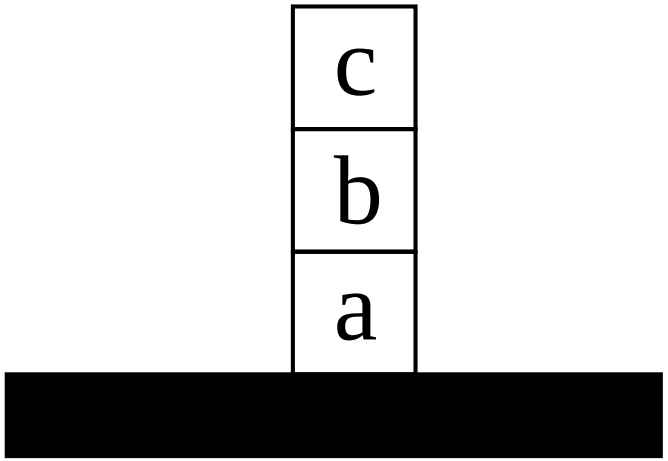
1. *Qualification Problem*
2. *Effects* (only focus for a long time)
3. *Frame Problem*, *Inertia*
4. *Ramification Problem*, *Indirect Effects*

Example: Blocksworld

initial:

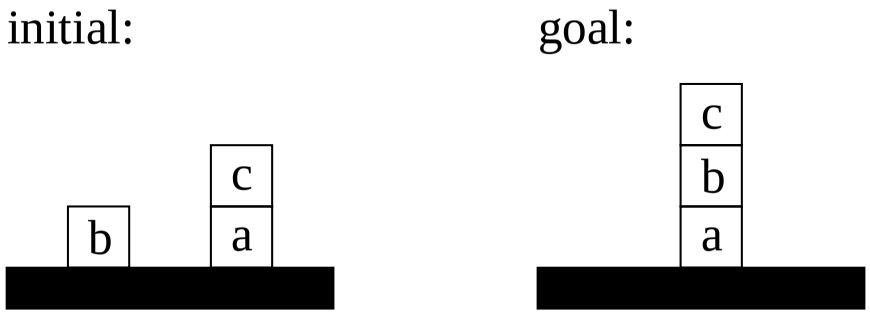


goal:



One clear block can be moved to a clear block or the table at a time.

Blocksworld: Background Knowledge



- Block names:

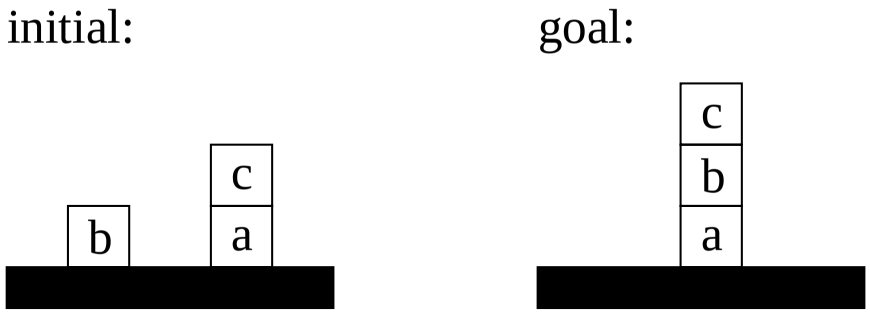
  a, b, c are blocks.

- Location names:

  table is a location.

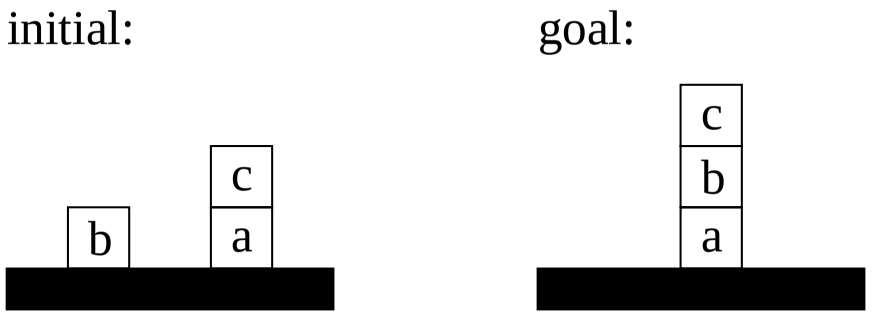
  All blocks are also locations.

Blocksworld: Fluents and Actions



Fluents:	on(B,L)	*(block B is on top of location L)*
	occupied(B)	*(something is on top of block B)*
	clear(B)	*(nothing is on top of block B)*
	supported(B)	*(B does not "float")*
Actions:	move(B,L)	*(move block B to location L).*

Blocksworld: Initial State, Goal, Domain Description



Initial state: on(b, table), on(c, a), on(a, table).

Goal: on(c,b), on(b, a), on(a, table).

System Description: source and destination locations must be clear, no concurrent actions, ...

$\mathcal{K}$– A Logic-Based Action Language

- Action and Fluent Declarations
- Action Costs
- Causation Rules
- Nonmonotonic Negation, Strong Negation
- Conditional Executability
- Initial State Constraints, Goal
- Inertia

Language $\mathcal{K}$: Background Knowledge

We assume that an Answer Set Program (the *background knowledge*) exists, which admits exactly one answer set (and is computable in polynomial time).

In the blocksworld example, it is

```
block(a).  block(b).  block(c).
location(table).
location(L) :- block(L).
```

It admits one answer set:

```
{block(a), block(b), block(c), location(table),
    location(a), location(b), location(c)}
```

Language $\mathcal{K}$: Type Declarations

Specify the ranges of the arguments of the fluents and actions, using the background knowledge.

For fluents `on` and `occupied`:

```
on(B,L) requires block(B), location(L).
occupied(B) requires block(B).
```

For action `move`:

```
move(B,L) requires block(B), location(L).
```

Action Costs

It is possible to associate costs to actions.

E.g. if moving costs five resources:

```
move(B,L) requires block(B), location(L) costs
                                    5.
```

Language $\mathcal{K}$: Causation Rules

Causation rules can be used to model action effects.

Moving a block to a location L causes the block to be on L afterwards:

```
caused on(B,L) after move(B,L).
```

Moving a block to a location L causes the block to be not on any other location L1 afterwards:

```
caused -on(B,L1) after move(B,L), L <> L1.
```

Language $\mathcal{K}$: Causation Rules

Causation rules can model indirect effects (solving the ramification problem).

If some block `B1` is on a block `B`, then `B` is occupied in the same moment:

```
caused occupied(B) if on(B1,B).
```

A block `B` is clear if it is not occupied:

```
caused clear(B) if not occupied(B).
```

One causation rule can also have both `if` and `after` conditions.

Language $\mathcal{K}$: Negation

Default negation (`not`)

```
caused clear(B) if not occupied(B).
```

Strong negation (`-`)

```
caused -on(B,L) after move(B,L1), L <> L1.
```

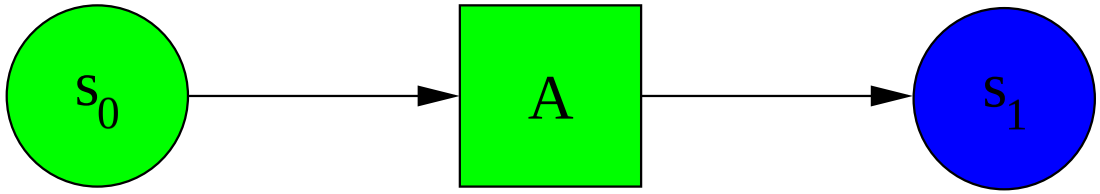
Difference between default negation and strong negation:

Default negation: `not` a holds if a is not known

Strong negation: $-a$ holds only if $-a$ is known explicitly

Semantics: Transitions

caused fl if Cond1 after Cond2



s_0 and s_1 are states, A a set of actions.

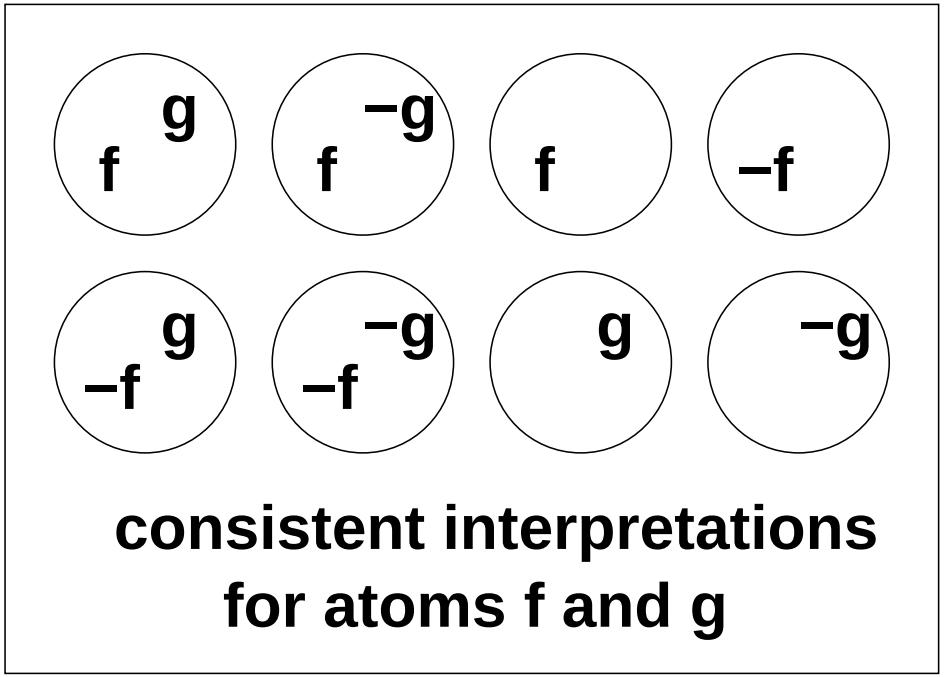
"Executing A after state s_0 takes you to s_1."

$Cond2$ refers to s_0 and A, while fl and $Cond$ refer to s_1.

Valid transitions satisfy all causation rules.

What is a State?

Consistent sets of Fluent literals ⇒ Fluents which are known to hold (positive literals) ⇒ Fluents which are known not to hold (strongly negative literals)



Constraints

Constraints are special causation rules:

```
forbidden a after b.
```

Any state satisfying `a after b` is illegal. Can also be written as

```
caused false if a after b.
```

Executability

Express preconditions actions (solving the qualification problem).

`move` is executable if neither the moved block `B` nor the destination `L` is occupied, and if `B` and `L` differ.

```
executable move(B,L) if not occupied(B),
                        not occupied(L), B <> L.
```

An executability condition can have both `if` and `after` conditions.

Nonexecutability

Specify conditions under which an action is *not* executable.

`nonexecutable` takes precedence over `executable`!

```
executable move(B,L).
nonexecutable move(B,L) if occupied(B).
nonexecutable move(B,L) if occupied(L).
nonexecutable move(B,L) if B <> L.
```

Initial State Constraints

To specify valid initial states, the scope of causation rules can be set to apply only to the initial state.

These rules cannot have an `after` condition!

They are grouped into a block preceded by `initially:`, while causation rules applying to all states are gruped into a block preceded by `always:`.

```
initially:  forbidden block(B), not
                supported(B).
```

Goal

A goal is a conjunction of fluents, plus an optional planlength.

```
on(c,b), on(b,a), on(a,table) ? (3)
```

A goal may also contain default negated fluents.

Language $\mathcal{K}$: Inertia

A fluent may be declared to be inertial:

```
inertial on(B,L).
```

The inertial fluent `on(B,L)` continues to hold until `-on(B,L)` is caused.

Language $\mathcal{K}$: Parallel vs. Sequential Plans

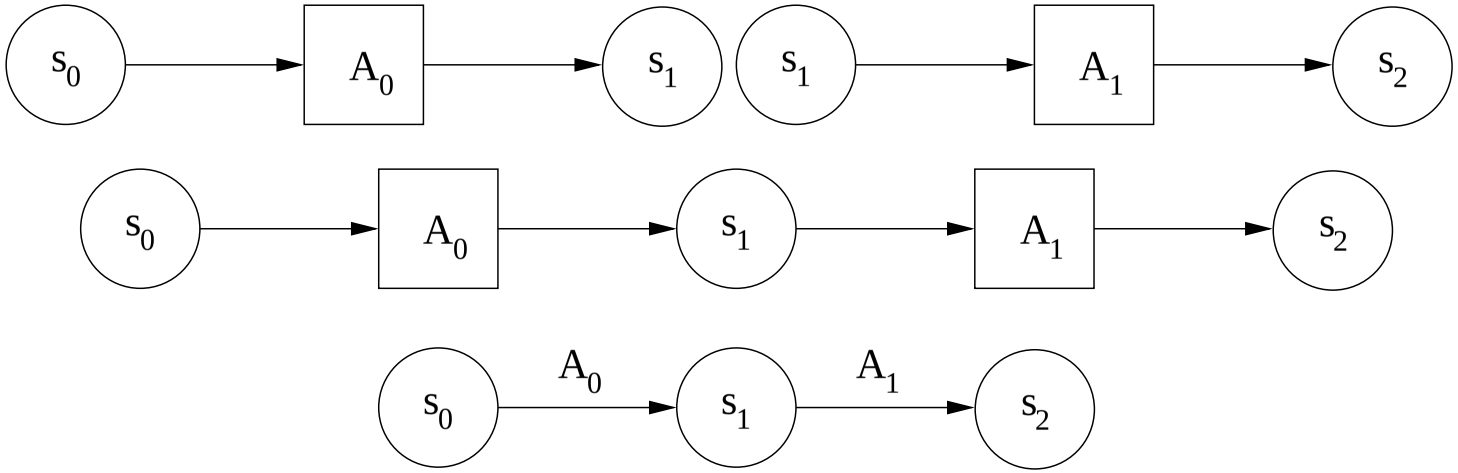
Can actions be executed in parallel or only one at a time?

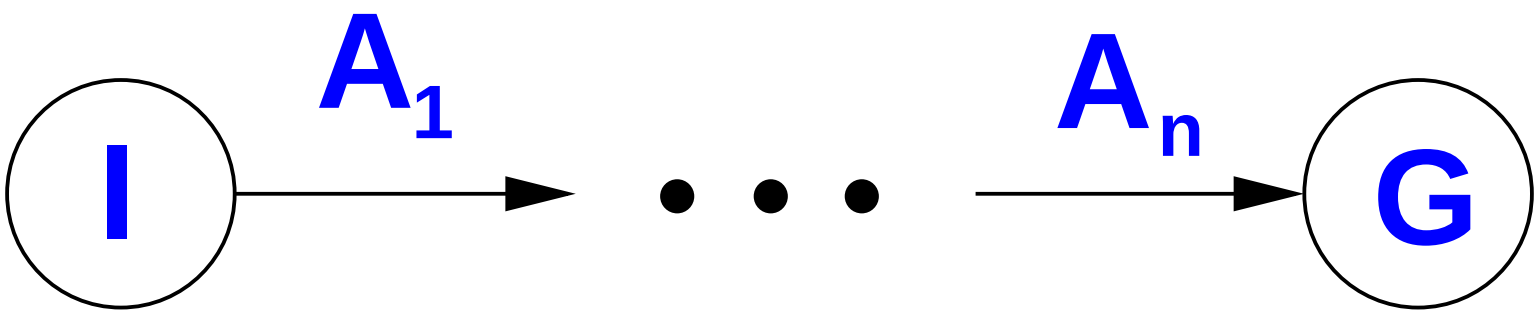
In $\mathcal{K}$, parallel actions are allowed by default. Sequential plans can be enforced by the keyword:

```
noConcurrency.
```

Plans

A chaining of transitions, starting from initial state, arriving at a goal state.





Blocksworld in $\mathcal{K}$

(Background Knowledge in file `blocksworld.dl`)

```
block(a).
block(b).
block(c).
location(table).
location(L) :- block(L).
```

Blocksworld in $\mathcal{K}$

(Domain Description in file `blocksworld.plan`)

```
fluents: on(B,L) requires block(B), location(L).
         occupied(B) requires block(B).

actions: move(B,L) requires block(B), location(L).

always:  executable move(B,L) if not occupied(B),
                                  not occupied(L), B<>L.
         inertial on(B,L).
         caused occupied(B) if on(B1,B).
         caused  on(B,L)  after move(B,L).
         caused -on(B,L1) after move(B,L), on(B,L1), L<>L1.

noConcurrency.
```

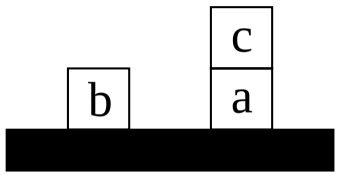
Blocksworld in $\mathcal{K}$ (cont'd)

(Planning Instance in file `sussman.plan`)

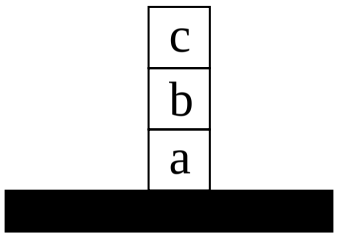
```
initially:  on(a,table). on(b,table). on(c,a).

goal:       on(c,b),on(b,a),on(a,table)? (3)
```

initial:



goal:



Invoking DLV$^{\mathcal{K}}$

```
> dlv -FPopt blocksworld.dl blocksworld.plan sussman.plan
DLV [build BEN/May 16 2003   gcc 2.95.4 20011002 (Debian prerelease)]


STATE 0: occupied(a), on(a,table), on(b,table), on(c,a)
ACTIONS: move(c,table)
STATE 1: on(a,table), on(b,table), on(c,table), -on(c,a)
ACTIONS: move(b,a)
STATE 2: on(a,table), on(b,a), on(c,table), -on(b,table), occupied(a)
ACTIONS: move(c,b)
STATE 3: on(a,table), on(b,a), on(c,b), -on(c,table),
         occupied(a), occupied(b)
PLAN: move(c,table); move(b,a); move(c,b)
```

Invoking DLV$^{\mathcal{K}}$

Viewing only the plan: Use grep (on Unix-like systems).

```
> dlv -FPopt blocksworld.dl blocksworld.plan sussman.plan | grep PLAN
PLAN: move(c,table); move(b,a); move(c,b)
```

Invoking DLV$^{\mathcal{K}}$

Setting plan length on the commandline:

```
> dlv -FPopt blocksworld.dl blocksworld.plan sussman.plan -planlength=2
DLV [build BEN/May 16 2003   gcc 2.95.4 20011002 (Debian prerelease)]
```

There is no plan of length 2.

Invoking DLV$^{\mathcal{K}}$

Assume `blocksworld_cost.dl` is equal to `blocksworld.dl`, except that moving costs 5.

```
> dlv -FPopt blocksworld_cost.dl blocksworld.plan sussman.plan
DLV [build BEN/May 16 2003   gcc 2.95.4 20011002 (Debian prerelease)]


STATE 0: occupied(a), on(a,table), on(b,table), on(c,a)
ACTIONS: move(c,table)
STATE 1: on(a,table), on(b,table), on(c,table), -on(c,a)
ACTIONS: move(b,a)
STATE 2: on(a,table), on(b,a), on(c,table), -on(b,table), occupied(a)
ACTIONS: move(c,b)
STATE 3: on(a,table), on(b,a), on(c,b), -on(c,table), occupied(a), occupied(b)
PLAN: move(c,table); move(b,a); move(c,b)  COST: 15
```

Only plans with minimal cost are computed!

Further Information

http://www.dlvsystem.com/K

Commercial Break

Praktika, Diplomarbeiten:

- **Handbuch** zu Action Language $\mathcal{K}$ und zu $\mathsf{DLV}^{\mathcal{K}}$
- **Evaluierung** und Vergleich mit anderen Systemen
- **GUI**: Weiterarbeit am Prototypen
- **Weiterentwicklung/Optimierung** von $\mathsf{DLV}^{\mathcal{K}}$
- **Nicht-Deterministisches Planen**: Sichere Pläne, Reaktive Pläne . . .
- **Anwendungen**