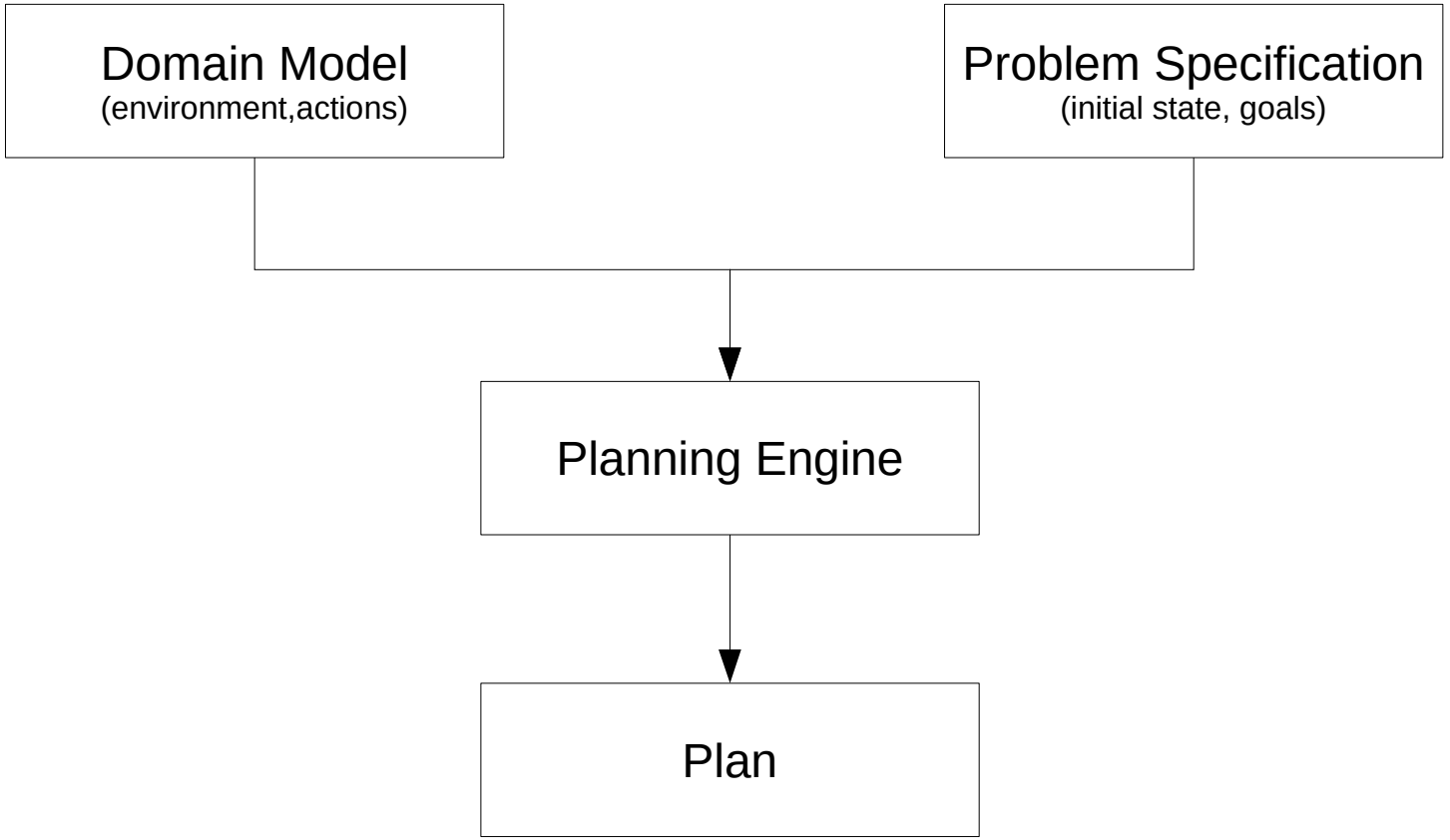
Planning for AI (B4M36PUI)

Languages, Knowledge Engineering Tools, Domain Reformulation

(based on tutorials presented on ICAPS 2017, AAAI 2018 and AAMAS 2019)

Lukáš Chrpa

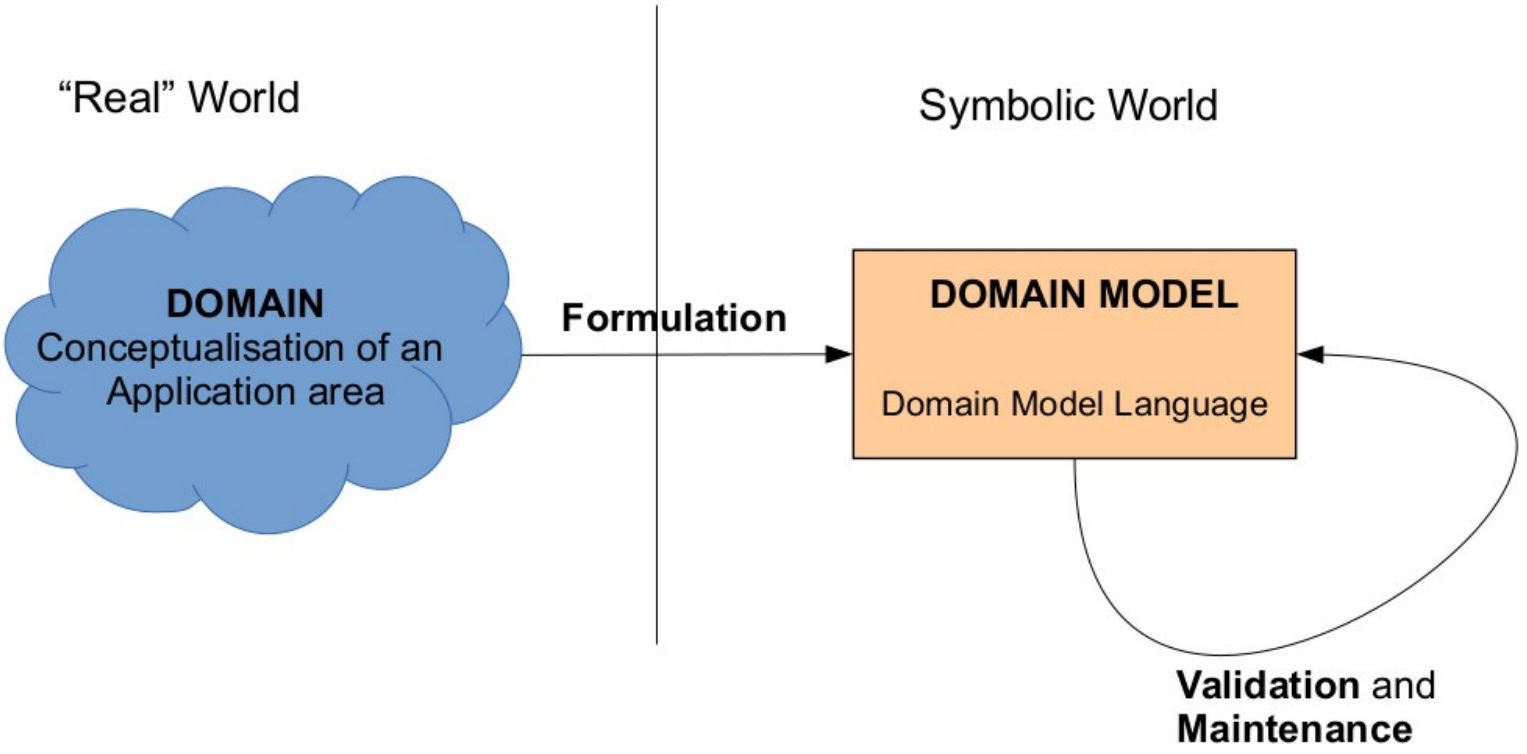
Domain-independent Planning Concept



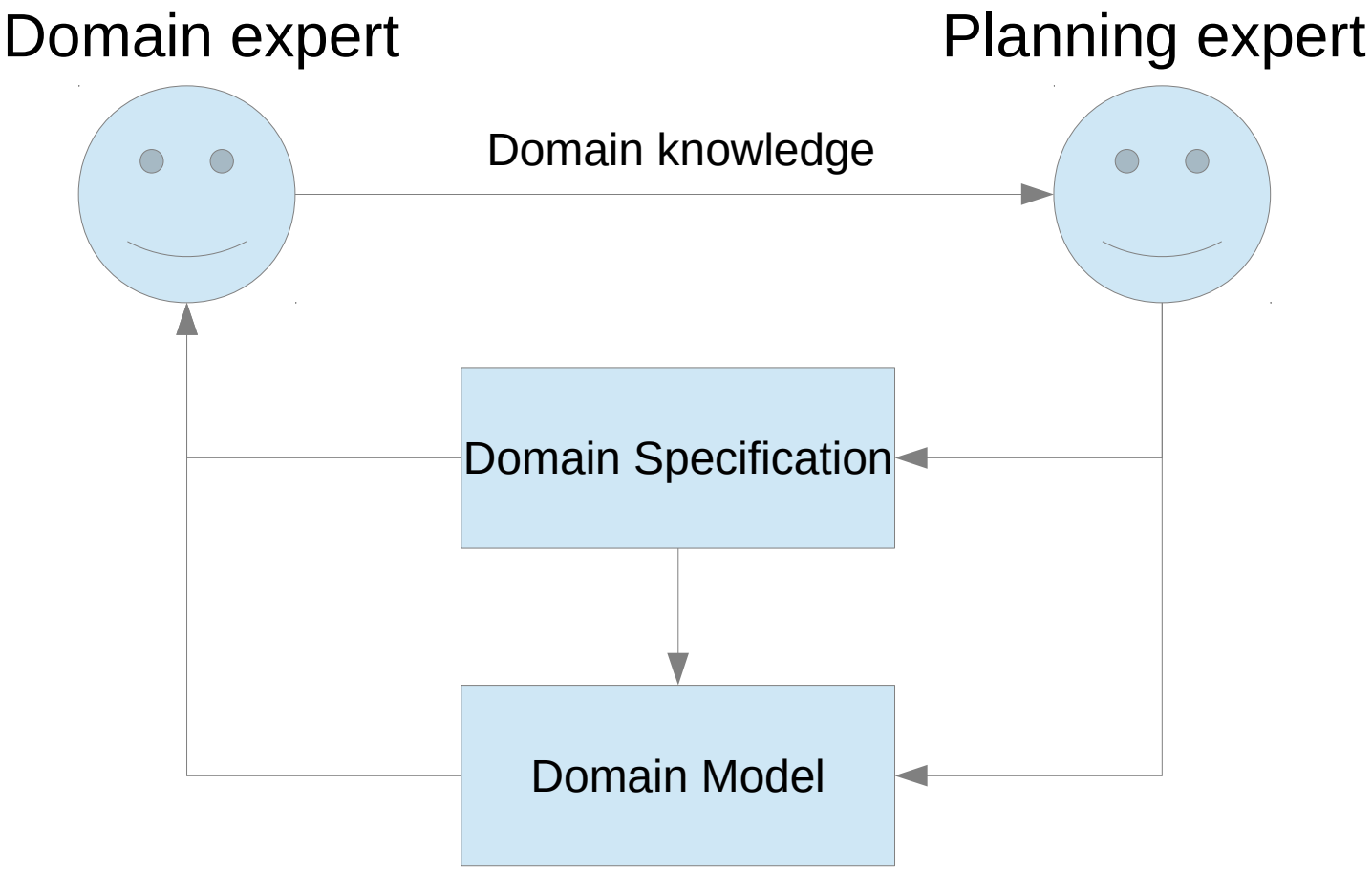
Domain-independent Planning Concept

- A (description) **language**
  - Describe domain model and problem specification
    (usually one domain model for a class of problems)
- A **planning engine**
  - must support the language
  - should be efficient for the given domain model
- **Plans** have to be interpreted (and executed)

Knowledge Engineering in Planning



Knowledge Engineering Process



An iterative process – can take a long time !

Knowledge Engineering Process

- Domain specification
  - formal conceptualization of the real-world domain
    - object types (e.g. truck, package)
    - environment description (e.g. predicates, fluents)
    - action description (precondition, effects)
  - independent on the language
- Domain model
  - the final product (to be fed into a planner)

Properties of a Domain Model [McCluskey et al. 2017]

- **Accuracy –** There is a mapping between domain requirements and a domain model
- **Consistency –** All assertions (invariants) are true
- **Completeness –** Solution plans correspond to real-world solutions
- **Adequacy –** A domain model is expressive enough to capture domain requirements
- **Operationality –** Planning engines can find solution plans in reasonable time/memory constraints

Modelling Languages

PDDL [McDermott et al, 1998]

- Planning Domain Definition Language (PDDL)
- Inspired by the STRIPS and ADL languages
- Most widespread
- Official language of International Planning Competitions (IPCs)

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types block)
  (:predicates (on ?x - block ?y - block)
       (ontable ?x - block)
       (clear ?x - block)
       (handempty)
       (holding ?x - block)
       )
  (:action pick-up
     :parameters (?x - block)
     :precondition (and (clear ?x)
                        (ontable ?x)

(handempty))
     :effect (and (not (ontable ?x))
                        (not (clear ?x))
                        (not (handempty))
                        (holding ?x))
)
…
```

Versions of PDDL

- **PDDL 1.2**
  - Predicate centric, i.e., classical (STRIPS) representation
  - Object types
  - ADL features (e.g., conditional effects, equality)
- **PDDL 2.1**
  - Numeric Fluents
  - Durative Actions
- **PDDL 2.2**
  - Timed-initial literals
  - Derived Predicates
- **PDDL 3.0**
  - State-trajectory constraints (hard constraints for the planning process)
  - Preferences (soft constraints for the planning process)
- **PDDL 3.1**
  - Object Fluents (aka FDR representation)

Extensions of PDDL

- **PDDL+**
  - Continuous processes
  - Exogenous events
- **PPDDL**
  - Probabilistic action effects
  - Reward fluents
- **MA-PDDL**
  - Multi-agent planning

NDDL [Frank & Jonsson, 2002]

- NASA's response to PDDL
- Variable representation
- Timelines/activities
- Constraints between activities

```
class Instrument
{
    Rover rover;
    InstrumentLocation location;
    InstrumentState state;

    Instrument(Rover r)
    {
            rover = r;
            location = new InstrumentLocation();
                 state = new InstrumentState();
    }

     action TakeSample{
                 Location rock;
            eq(10, duration);
     }
     …
}

Instrument::TakeSample
{
      met_by(condition object.state.Placed on);
      eq(on.rock, rock);

    contained_by(condition object.location.Unstowed);

      equals(effect object.state.Sampling sample);
      eq(sample.rock, rock);

      starts(effect object.rover.mainBattery.consume tx);
      eq(tx.quantity, 120); // consume battery power
}
```

https://github.com/nasa/europa/wiki/Example-Rover

ANML [Smith et al., 2008]

- Combines aspects from NDDL and PDDL
  - Actions and states (PDDL)
  - Variable representation (NDDL)
  - Temporal Constraints (NDDL)
- Hierarchical methods

```
action Pickup (crew ev, object item)
{
duration := 5 ;
[start] located(ev) == located(item);
[all] possesses(ev,item) == FALSE:
->TRUE ;
[end] located(item) := POSSESSED ;
}

action Putaway (crew ev, object item,
location stowage)
{
Duration := 10 ;
[start] located(ev) == stowage ;
[all] possesses(ev, item) ==  TRUE:
->FALSE ;
[end] located(item):= stowage ;
}
```

[Boddy & Bonasso, 2010]

RDDL [Sanner, 2011]

- became the official language of the probabilistic track of the IPC since 2011
- models partial observability, probabilistic effects
- efficient description of (PO)MDPs

```
domain wildfire_mdp {

types {
x_pos : object;
y_pos : object;
};

pvariables {

// Action costs and penalties
COST_CUTOUT       : {non-fluent, real, default =   -5 }; // Cost
to cut-out fuel from a cell
COST_PUTOUT       : {non-fluent, real, default =  -10 }; // Cost
to put-out a fire from a cell
PENALTY_TARGET_BURN : {non-fluent, real, default = -100 }; //
Penalty for each target cell that is burning
PENALTY_NONTARGET_BURN : {non-fluent, real, default =   -5 }; //
Penalty for each non-target cell that is burning
…..
}

cpfs{
burning'(?x, ?y) =  if ( put-out(?x, ?y) ) // Intervention to
put out fire?
                     then false
         // Modification: targets can only start to burn if at
least one neighbor is on fire
            else if (~out-of-fuel(?x, ?y) ^ ~burning(?x, ?y)) //
Ignition of a new fire? Depends on neighbors.
              then [if (TARGET(?x, ?y) ^ ~exists_{?x2: x_pos, ?
y2: y_pos} (NEIGHBOR(?x, ?y, ?x2, ?y2) ^ burning(?x2, ?y2)))
                    then false
                    else Bernoulli( 1.0 / (1.0 + exp[4.5 -
(sum_{?x2: x_pos, ?y2: y_pos} (NEIGHBOR(?x, ?y, ?x2, ?y2) ^
burning(?x2, ?y2)))]) ) ]
              else
              burning(?x, ?y); // State persists
…
}
```

https://cs.uwaterloo.ca/~mgrzes/IPPC_2014/

Domain-independent Planners

- Dozens of classical planners
  - support typed STRIPS
  - newer planners support action costs, and some ADL features
  - many of them are optimal
- Several temporal planners
  - support durative actions
  - few support numeric fluents or timed-initial literals
  - few fully support concurrency
  - very few are optimal
- Several probabilistic planners
  - (PO)MDP
  - FOND
- A few continuous planners
- ....

Language Expressiveness vs. Planning Engines

- "*It is almost a law in PDDL planning that for every language feature one adds to a domain definition, the number of planners that can solve (or even parse) it, and the efficiency of those planners, falls exponentially*" [anonymous reviewer]

- Motivate **development of more expressive** planning engines
- **Reduce** the number of **features** in models

KE Tools for Planning Domain Modelling

Purpose of KE tools

- Assist in domain developing process
  - Support development cycle (as in SW engineering)
  - Visualize (parts of) the model
  - Verification and Validation support (e.g. consistency check)
  - …
- Usable by non-experts (but with basic knowledge of planning)

GIPO [Simpson et al., 2007]

- GIPO (Graphical Interface for Planning with Objects) won the ICKEPS 2005 competition
- Based on the **OCL** (Object-Centred Language)
- Define **life histories** of **objects**
- Supports **“classical” PDDL** (limitedly also ”durative” actions)
- Supports **HTN** (HyHTN planner is integrated) [McCluskey et al., 2003]

ItSimple [Vaquero et al., 2007;2012]

- Supports development cycle
- Exploits **UML** for domain modelling
- Exploits **Petri Nets** for dynamic analysis of **state machines** (e.g. reachability analysis)
- Supports **PDDL 3.1**

https://code.google.com/archive/p/itsimple/

  - Project webpage

http://www.youtube.com/watch?feature=player_embedded&v=FGBhvBnzyvo

  - Tutorial on domain modelling it ItSimple by Chris Muise

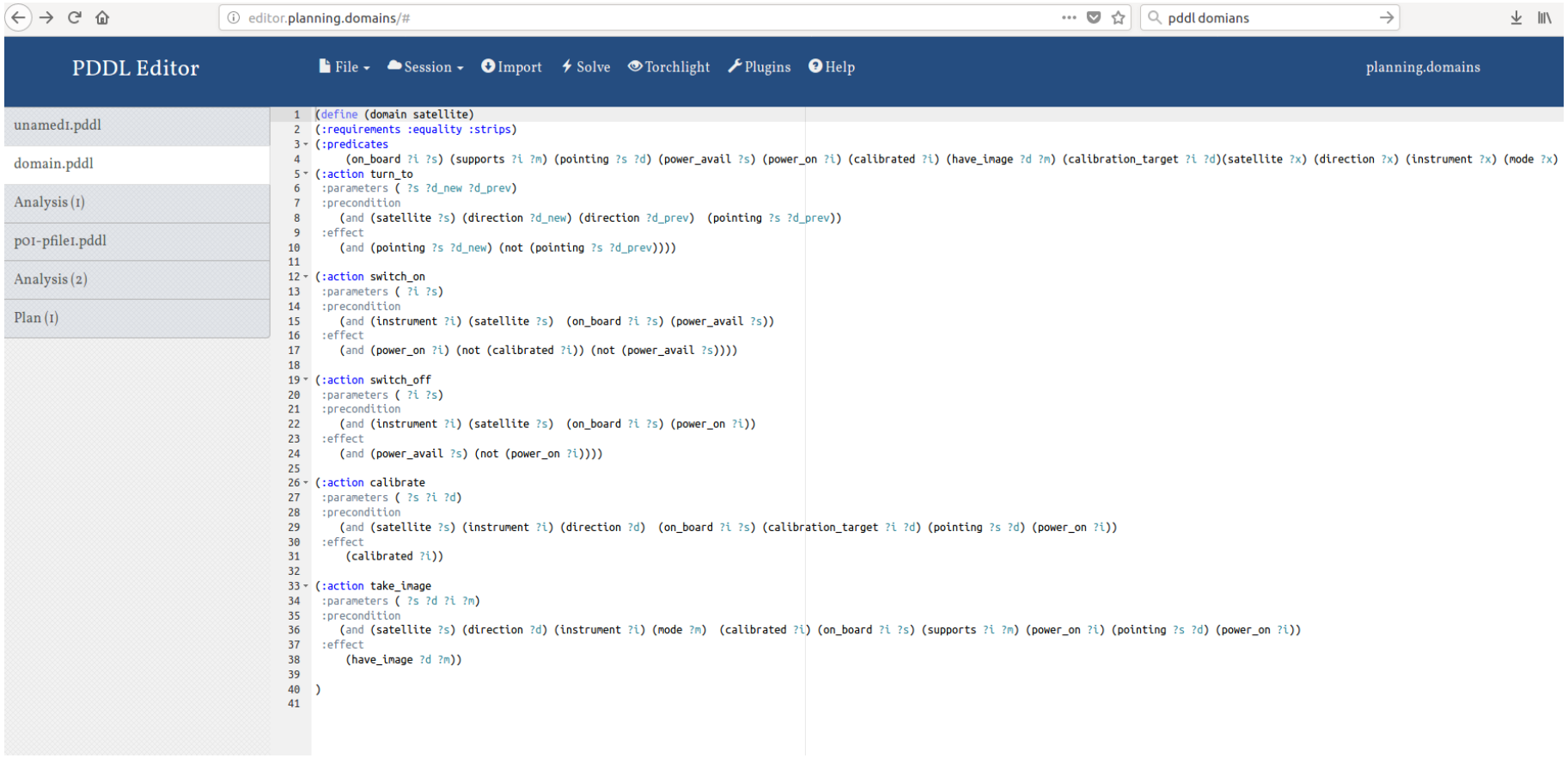
Some other KE Frameworks

- EUROPA [Barreiro et al., 2012]
  - Framework supporting NDDL and ANML
- JABBAH [Gonzalez-Ferrer et al., 2009]
  - Supports HTN
- KEWI [Wickler et al., 2014]
  - Object Centred (including inheritance)
  - Web Application (supports collaboration)
- VIZ [Vodrazka & Chrpa, 2010]
  - A “light-weight” KE tool

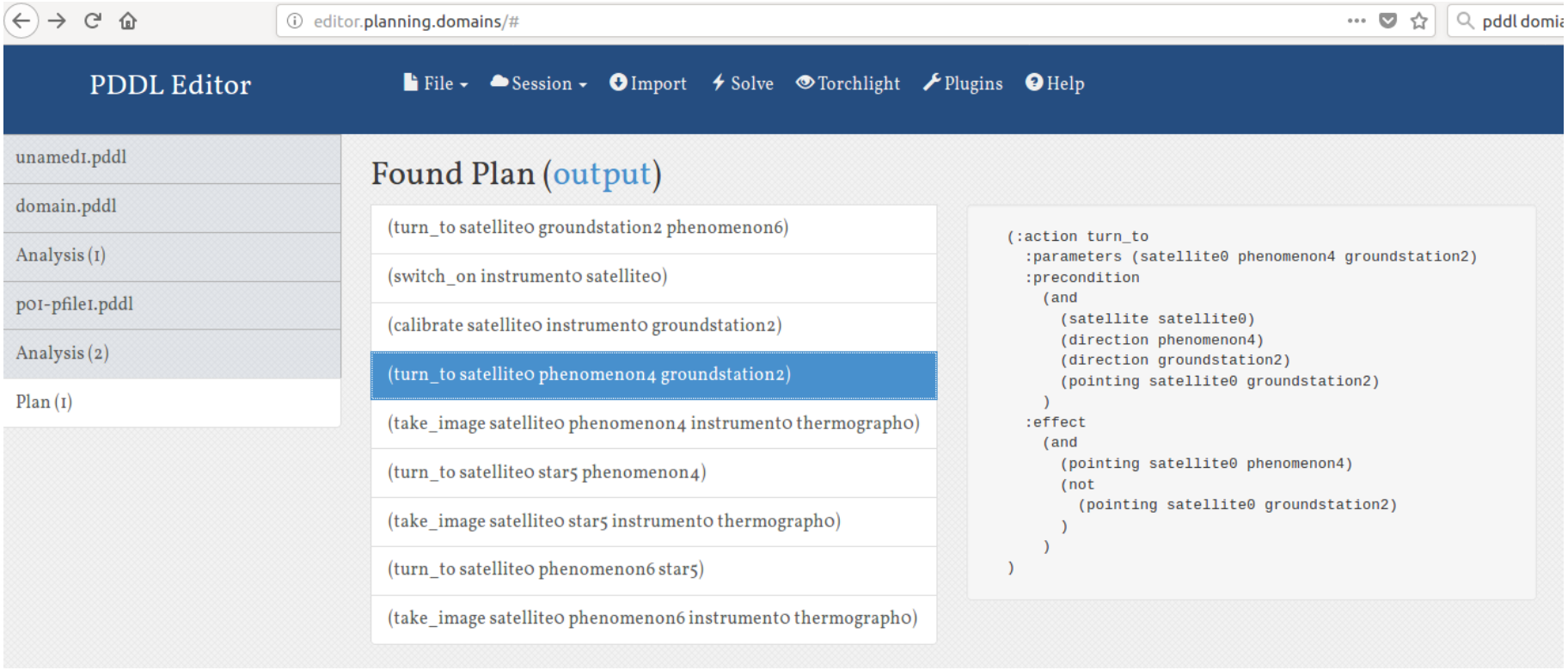
Planning.Domains

- “A Collection of Tools for Working with Planning Domains” [Muise]
- Web application
- Rich editor (syntax highlighting, autocomplete, etc.)
- Plug-in support
- Repository of all domains and problems from the IPCs

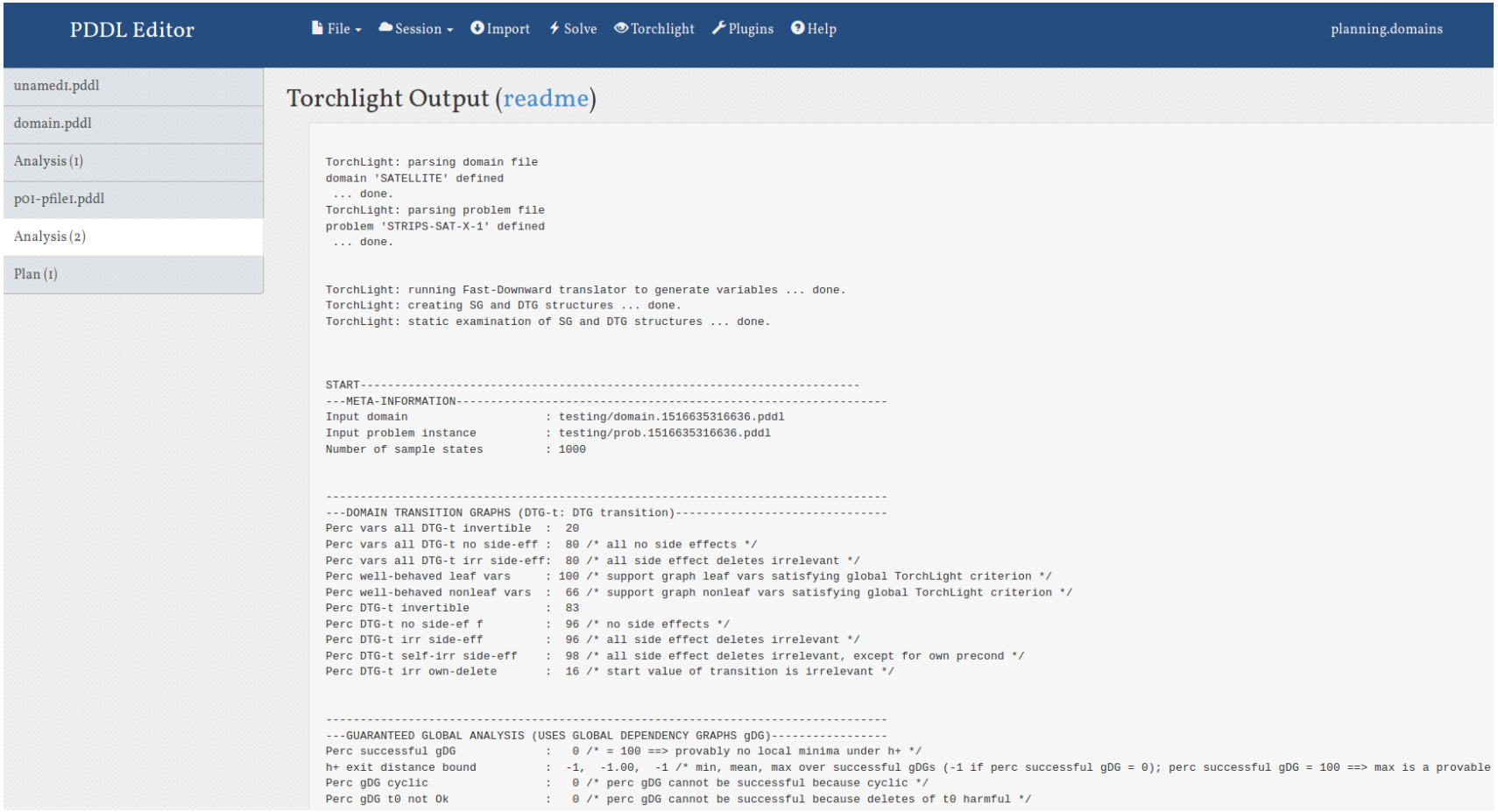
Planning.Domains – Sample Domain (Satellite)



Planning.Domains – Sample Plan (Satellite domain)



Planning.Domains – Analysis (by TorchLight)



Automated Domain Model Acquisition

Can we do domain model encoding automatically ?

Yes, to some extent…

- **OpMaker**
  - Partial Domain Model
  - Plan Traces
- **ARMS**
  - Background Knowledge,
  - Plan Traces
- **LOCM, n-LOCM**
  - Plan Traces (only)

Domain Control Knowledge and Model Reformulation

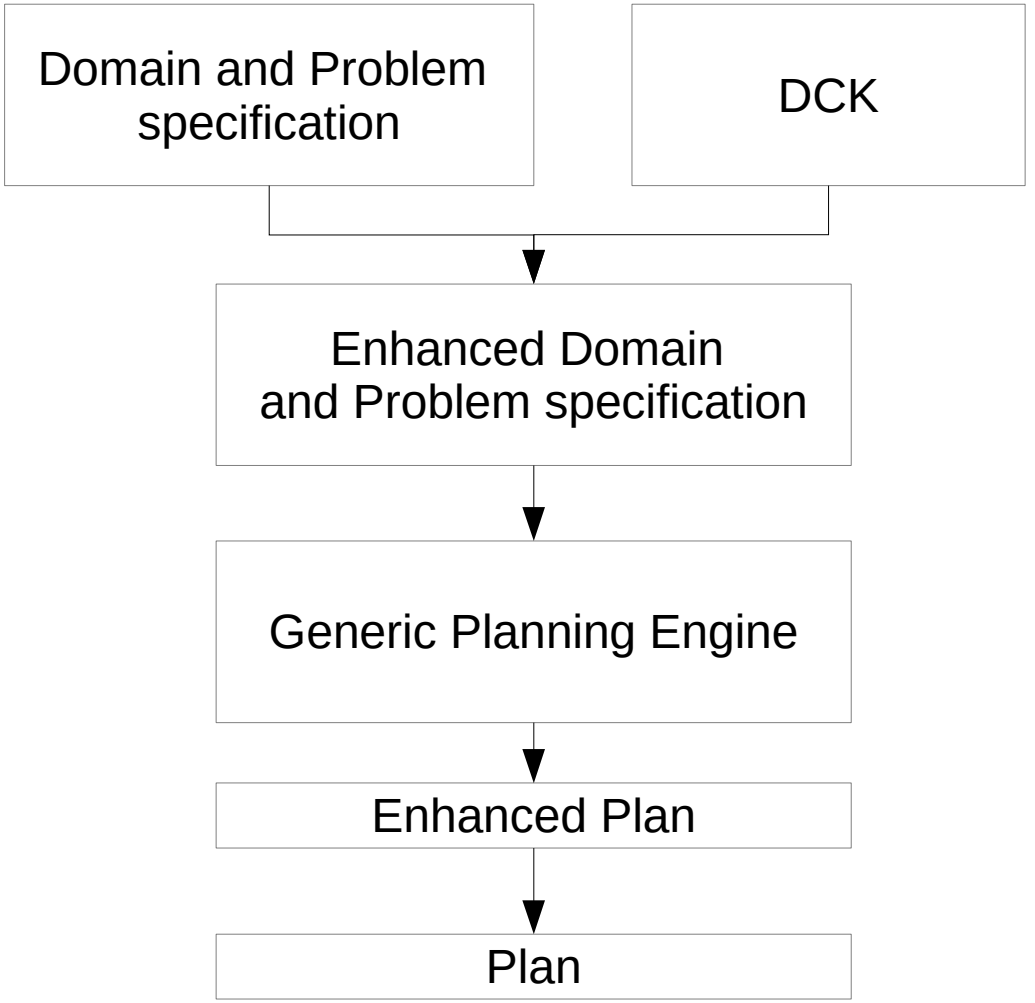
Efficiency of Domain Model

- One might introduce “accidental complexity”
  - Too large representation
  - “Deep” and undetectable dead-ends
  - Not “going along” with some classes of heuristics
  - ….
- Can we improve the planning process by making the model more efficient ?
  - bridge the gap between domain-dependent and domain-independent planning ?

Domain Control Knowledge (DCK)

- Captures useful domain-specific information
- Provides “guidance” for planning engines
- Complement “raw” domain model specification
- Two main categories of DCK
  - Planner-specific (e.g. TALPlanner, Roller)
  - Planner-independent (the rest of this lecture !)

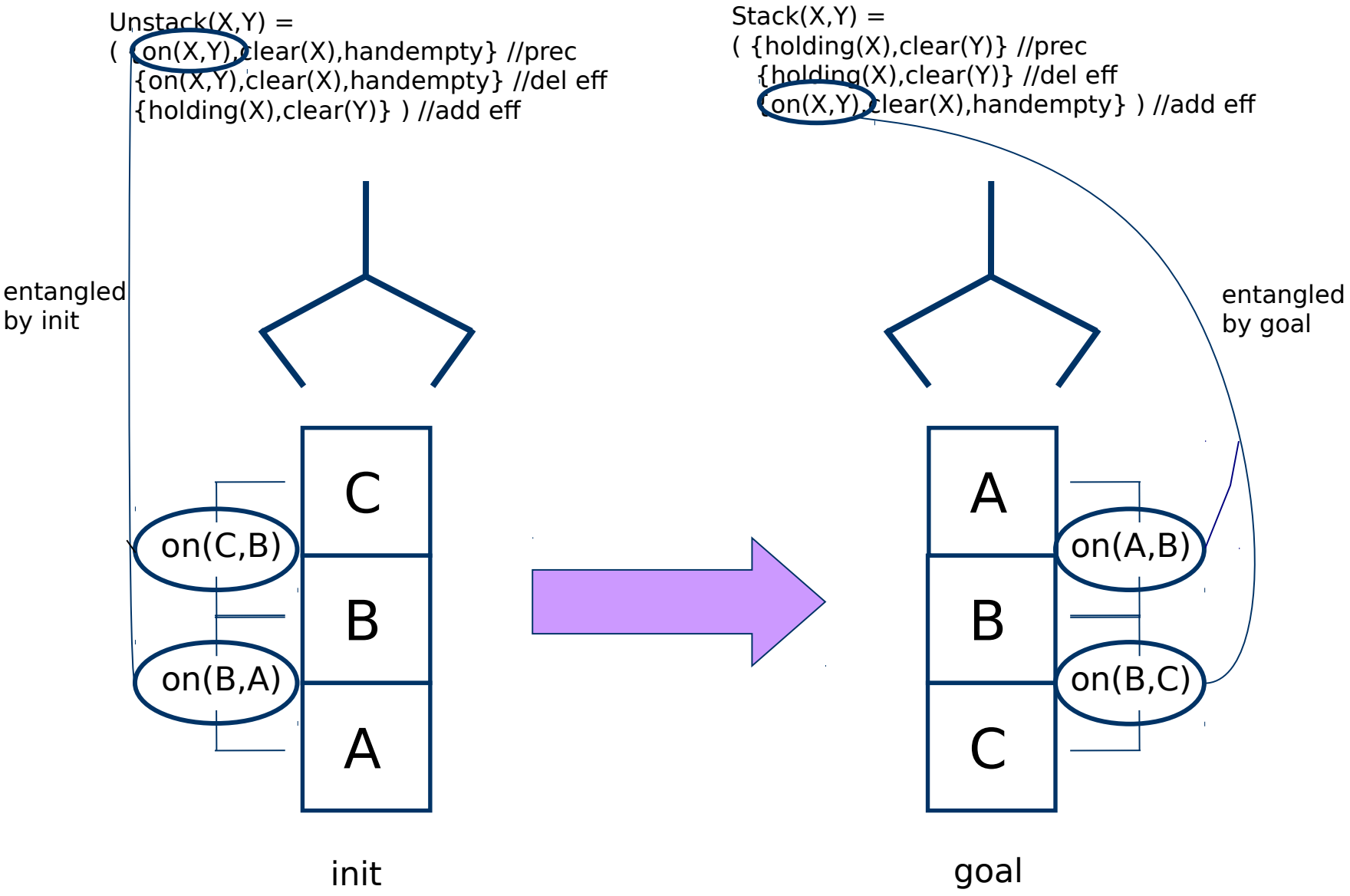
Planner-independent DCK



Obtaining DCK

- Automatically
  - training based
    - analyzing training plans, solution of simpler problems in a given domain
    - analyzing state-space of simpler problems
  - online
- Manually

Outer Entanglements



allowed: Unstack(C,B), Unstack(B,A)

allowed: Stack(A,B), Stack(B,C)

Outer Entanglements [Chrpa et al. 2018]

- Outer entanglements are relations between planning operators and initial or goal predicates
- **Entanglement by init –** allows only such instances of an operator requiring an initial predicate
  - e.g. unstacking blocks only from their initial positions, loading packages only in their initial locations
- **Entanglement by goal –** allows only such instances of an operator achieving goal predicates
  - e.g. stacking blocks only to their goal positions, unloading packages only in their goal locations

Encoding Outer Entanglements

1) Create a “twin” predicate *p’* of an “entangled” predicate *p*
2) Modify the “entangled” operator by adding *p’* into its precondition *(p’* has the same parameters as *p*)
3) Create instances of *p’* corresponding with instances of *p* in the initial state (resp. goal) and add them to the initial state

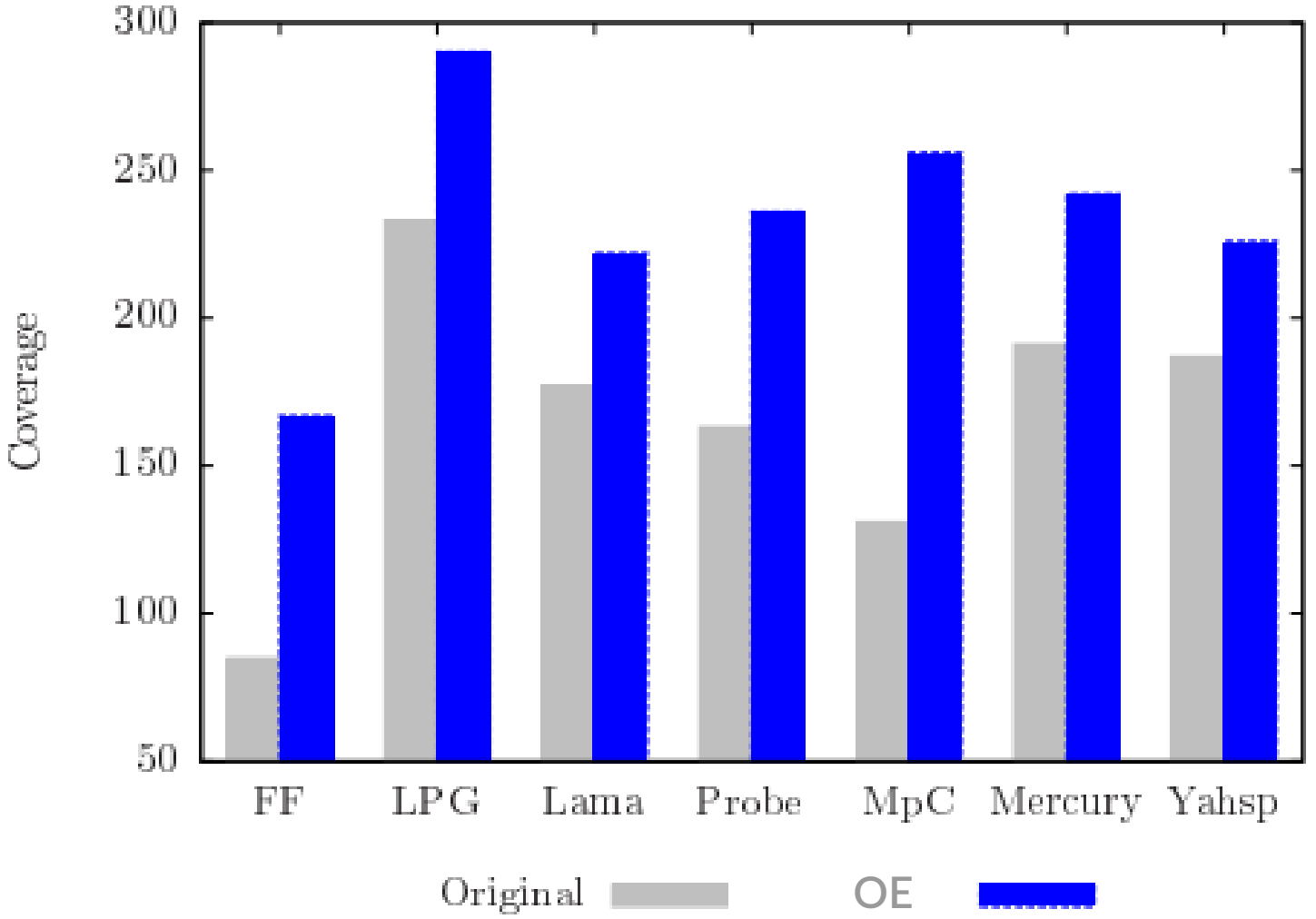
Example of “entangled” operators

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

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

Outer Entanglements – benefits and shortcomings

- Outer Entanglements **restrict** the number of instantiated operators and consequently might reduce the size of the state space
- Outer Entanglements (significantly) **reduces** memory requirements
- Remarkable performance in BlockWorld or Logistics types of domains
- Can be rather restrictive and might work efficiently in subclasses of domains/problems
- Outer entanglements might be **learnt** from a set of training plans – might **compromise completeness**

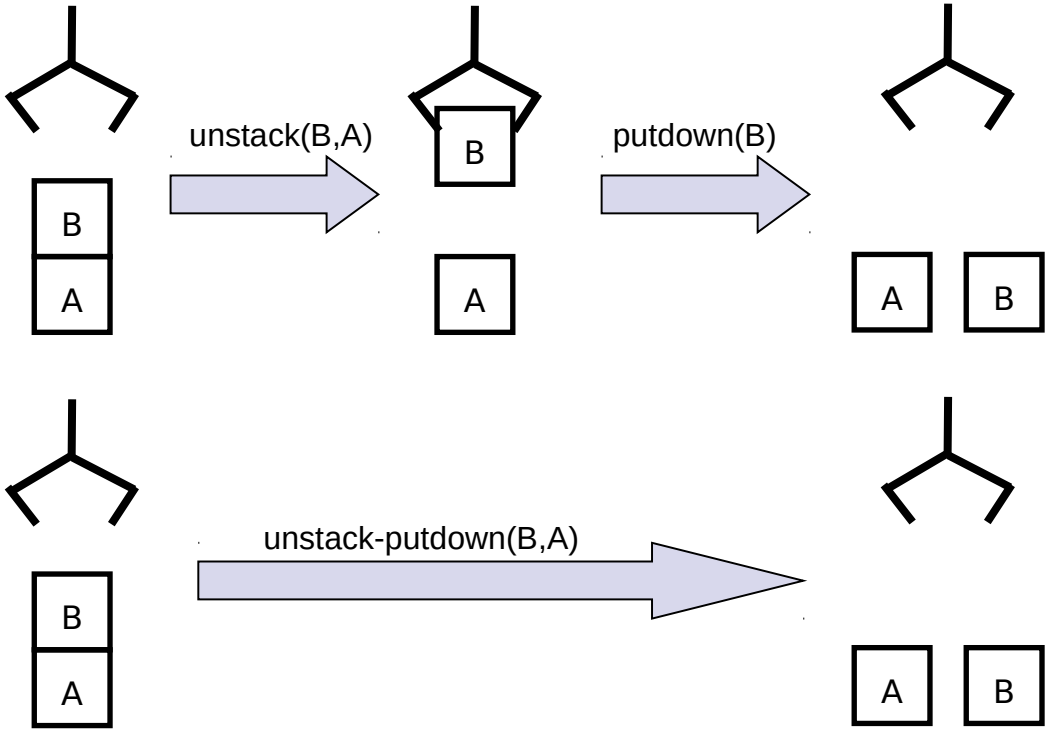
Outer Entanglements - results



Macro-operators (Macros)

- Primitive operators can be assembled into one single operator – macro-operator (macro)
- Assemblage of operators o_i and o_j into $o_{i,j}$:
  - $pre(o_{i,j}) = pre(o_i) \cup (pre(o_j) - add(o_i))$
  - $del(o_{i,j}) = (del(o_i) \cup del(o_j)) - add(o_j)$
  - $add(o_{i,j}) = (add(o_i) - del(o_j)) \cup add(o_j)$
- Widely studied (e.g. Macro-FF, Wizard, MUM, BLOMA)
- Can address a specific shortcoming of a planner (e.g. Marvin [Coles et al, 2007])

Macros - example



unstack(X,Y) =
{ {on(X,Y),clear(X),handempty} //prec
  {on(X,Y),clear(X),handempty} //del eff
  {holding(X),clear(Y)} } //add eff

pudown(X) =
{ {holding(X)} //prec
  {holding(X)} //del eff
  {ontable(X),on(X),clear(X),handempty} } //add eff

unstack-putdown(X,Y) =
{ {on(X,Y),clear(X),handempty} //prec
  {on(X,Y),holding(X),} //del eff
  {clear(X),clear(Y),ontable(X),handempty} } //add eff

Macros – Benefits and Shortcomings

- Macros can be understood as **'short-cuts'** in the search space
- Solution plans can be **much shorter**
- Introducing macros can **increase branching factor** considerably !
- There might be **high memory requirements** for planners

“A short-cut is the longest way between two points”

MUM [Chrpa et al., 2014]

- Outer entanglements can **reduce branching factor** the macros introduce
- Applying outer entanglements only on macros **does not compromise completeness**
- Outer entanglements **provide heuristics** in the macro learning process
- “Instance wise”
- Might find useful macros (e.g. pick-move-drop)

“Critical Section” Macros [Chrpa & Vallati, 2019,2022]

- Critical Sections use a shared resource and has to be completed at once (without any other process interfering)
- In planning, share resources involve *robotic hand, truck etc.*
- Critical Section Macros involves
  - **Locker** (locks the resource, e.g. *pick*)
  - *User* (uses the resource, e.g. *paint*)
  - “Gluing” op (“connects” other ops – e.g. *move*)
  - **Releaser** (releases the resource – e.g. *drop*)
- Can learn more complex macros (e.g. shaking the cocktail, pouring it to the glass and cleaning the shaker afterwards)
- Can be combined with other “chaining” approaches (e.g. MUM)
- Aggressive version removes replaced original ops

Macros - results

Planner	Coverage					PAR10					IPC quality				
	O	M	B	CS	ACS	O	M	B	CS	ACS	O	M	B	CS	ACS
Elevator	16.8	-	16.4	19.4	-	1494.8	-	1697.8	299.2	-	14.7	-	14.4	15.3	-
Floortile	8.4	8.8	-	8.5	20.0	5251.2	5081.4	-	5192.2	0.1	8.1	8.1	-	6.2	18.0
GED	13.8	13.0	15.0	15.6	-	2830.1	3172.8	2284.8	1984.1	-	11.6	11.1	9.7	13.7	-
Hiking	13.5	-	0.3	17.8	16.0	3000.8	-	8887.5	1034.1	1884.4	12.4	-	0.2	11.6	12.5
Termes	6.0	3.6	5.4	5.3	-	6320.3	7395.0	6610.6	6669.4	-	5.5	2.3	4.2	4.6	-
Transport	14.1	-	13.0	11.6	0.0	2697.6	-	3223.1	3858.1	9000.0	13.0	-	11.4	10.3	0.0
Barman	7.1	-	9.9	30.0	30.0	6918.8	-	6169.8	49.1	3.5	5.6	-	8.6	30.0	30.0
Bw	16.4	-	14.9	22.9	25.6	4137.1	-	4588.8	2189.8	1324.8	10.8	-	12.0	20.2	23.3
Depots	10.9	11.8	12.1	11.8	30.0	5833.0	5542.3	5424.8	5535.0	0.1	7.2	8.2	8.3	8.2	29.7
Gold	29.8	27.0	30.0	29.5	-	77.8	910.9	0.2	158.0	-	26.0	22.6	20.9	22.5	-
Gripper	0.9	11.0	11.0	11.1	30.0	8753.6	5746.9	5746.5	5706.3	44.1	0.8	11.0	11.0	11.1	29.5
Match-Bw	15.6	17.5	4.6	27.0	28.1	4330.9	3764.1	7613.5	914.7	562.5	11.8	13.9	3.6	24.0	26.7
Rovers	23.3	20.9	20.4	24.3	-	2193.9	2878.6	3027.9	1910.8	-	22.5	18.1	19.7	23.6	-
Sokoban	25.5	-	25.0	27.0	-	1373.7	-	1529.2	935.8	-	23.0	-	20.4	23.8	-
Storage	22.8	23.5	23.6	23.1	-	3272.5	2943.5	2878.1	3125.9	-	19.3	21.1	21.6	21.1	-
Thoughtful	19.0	19.3	18.8	18.6	-	3305.5	3232.6	3384.3	3420.8	-	18.1	18.1	17.9	17.7	-

Coverage, average PAR10 score (in seconds) and the IPC quality score of the (O)riginal, (M)UM, (B)LoMa, Critical Section Marcos (CS), Aggressive Critical Section Macros (ACS)

“Bagged” representation [Riddle et al, 2015]

- Representing individual objects might not be efficient if we care about their numbers
- In Gripper, K balls are moved from roomA to roomB
  - Standard representation: (at ball1 rooma), …, (at ballk rooma)
  - Bagged representation: (count ball rooma nk)
- Bagged representation alleviates some unwanted symmetries (e.g. which ball is picked first)

“Bagged” representation – sample operators

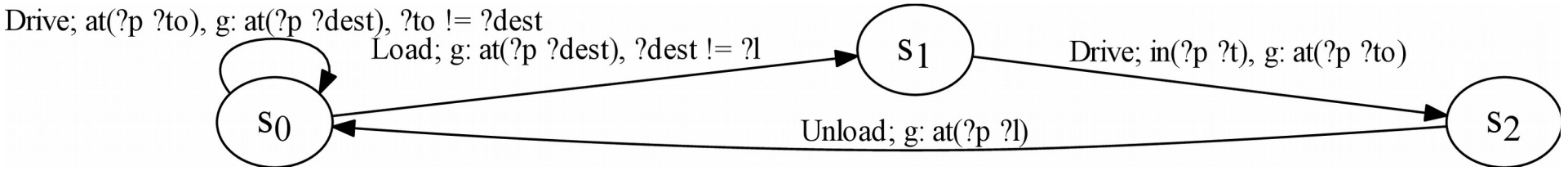
```
(:action pick
:parameters (?n1 ?n0 ?obj ?room ?gripper)
:precondition (and (ball ?obj)(room ?room)(gripper ?
gripper)(at-robby ?room)(free ?gripper)(more ?n1 ?n0)
(count ?obj ?room ?n1))
:effect (and (carry ?obj ?gripper)(not (count ?obj ?room
?n1))(count ?obj ?room ?n0)(not (free ?gripper))))

(:action drop
:parameters (?n1 ?n0 ?obj ?room ?gripper)
:precondition (and (ball ?obj)(room ?room)(gripper ?
gripper)(carry ?obj ?gripper)(more ?n1 ?n0)(at-robby ?
room)(count ?obj ?room ?n0))
:effect (and (not (count ?obj ?room ?n0))(count ?obj ?
room ?n1)(free ?gripper)(not (carry ?obj ?gripper)))))
```

Transition-based DCK [Chrpa & Bartak, 2016]

- Inspired by Finite State Automata
- Define “grammar” of solution plans
- “Schematical” representation is easier to understand by non-experts in planning
- Can be incorporated in planning domain models

Specifying Transition-based DCK – an example

- An empty truck (can carry at most one package) should move only to locations where some package is waiting to be delivered
- After a package that has to be delivered is loaded into the truck, the truck moves to package's goal location where the package is then unloaded



Transition-based DCK – formal specification

- A quadruple (S,O,T,s_0) where
  - S is a set of **DCK states**
  - $s_0 \in S$ is the **initial DCK state**
  - O is a set of **planning operators**
  - T is a set of **transitions**
- Each transition is in the form (s,o,C,s') where
  - $s,s' \in S$, $o \in O$
  - C is a set of **constraints** where each is in the form
    - $p, \neg p$ – p **must or must not be** in the current planning state
    - g: p – p must be **an open goal** in the current planning state

Use of Transition-based DCK

- Let s_{Π} be the current planning state and s_K be the current DCK state
- The intermediate step of the generic planning algorithm with embedded transition based DCK
  1) Non-deterministically select an action *a* such that
     - *a* is applicable in s_{Π}
     - There is a transition (s_K, o, C, s'_K) such that *a* is an instance of *o* and all constraints in *C* are satisfied
  2) Update the current planning state by applying *a* in s_{Π}
  3) Set s'_K as the current DCK state
- A constraint in *C* in the form $p, \neg p,$ g:p is satisfied iff $p \in s_{\Pi}, p \notin s_{\Pi}$, p is an open goal in s_{Π} respectively

Translating into PDDL - Example

(:action **Drive-empty**
:parameters (?t - truck ?from ?to ?dest - location ?p - package)
:precondition (and (at ?t ?from)(at ?p ?to) (DCK-state s0) (open-goal-at ?p ?dest) (not (= ?to ?dest)))
:effect (and (not (at ?t ?from))(at ?t ?to))
)

(:action **Drive-full**
:parameters (?t - truck ?from ?to - location ?p - package)
:precondition (and (at ?t ?from)(DCK-state s1)(in ?p ?t)(open-goal-at ?p ?to))
:effect (and (not (at ?t ?from))(at ?t ?to)
(not (DCK-state s1))(DCK-state s2))
)

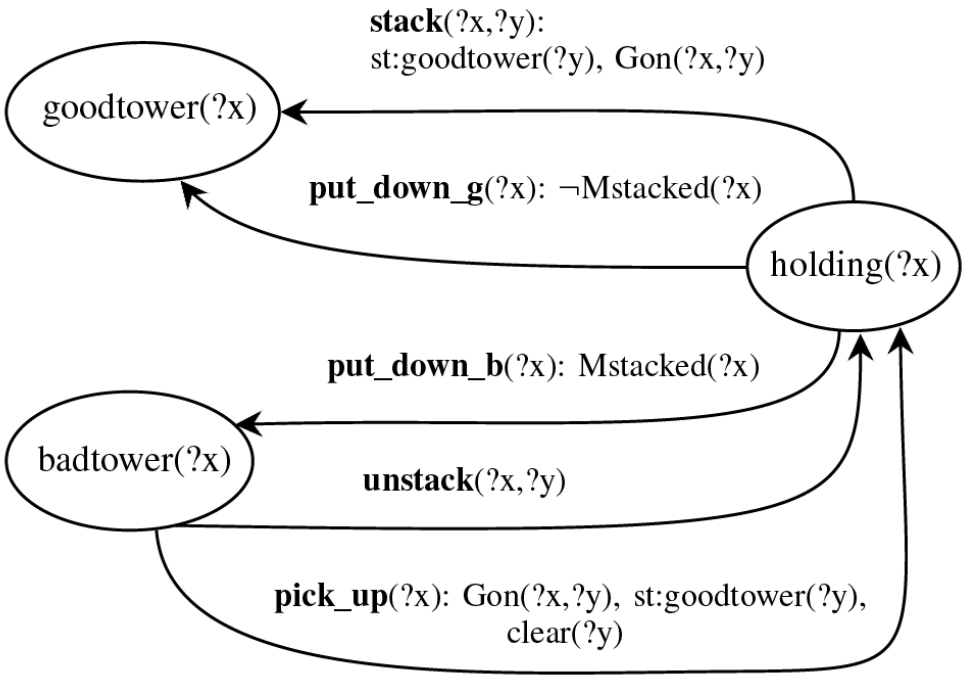
(:action **Load**
:parameters (?t - truck ?p - package ?l *?dest* - location)
:precondition (and (at ?t ?l)(at ?p ?l)(free ?t)(DCK-state s0)(open-goal-at ?p ?dest) (not (= ?to ?dest)))
:effect (and (not (at ?p ?l))(not (free ?t))(in ?p ?t)(not (DCK-state s0))(DCK-state s1))
)

(:action **Unload**
:parameters (?t - truck ?p - package ?l - location)
:precondition (and (at ?t ?l)(in ?p ?t)(DCK-state s2)(open-goal-at ?p ?l))
:effect (and (not (in ?p ?t))(free ?t)(at ?p ?l)
(not (open-goal-at ?p ?l))
(not (DCK-state s2))(DCK-state s0))
)

The PDDL encoding of a DCK enhanced Simple Logistic domain model (supplementary predicates in red)

Attributed Transition-based DCK [Chrpa et al., 2021]

- Extends DCK states by considering attributes and DCK memory predicates
- Again, it can be compiled into a planning task



```
(:action pick-up
   :parameters (?x - block ?y - block)
   :precondition (and (clear ?x) (ontable ?x) (handempty)
                      (g_on ?x ?y) (DCK_goodtower ?y)
                      (clear ?y)(DCK_badtower ?x)
   )
   :effect
   (and (not (ontable ?x)) (not (clear ?x))(not (handempty))
        (holding ?x)
        (DCK_holding ?x)(not (DCK_badtower ?x))
))

 (:action put-down_DCK-put-down-g
   :parameters (?x - block)
   :precondition (and (holding ?x)
                      (not (mg_Stacked ?x))(DCK_holding ?x)
                 )
   :effect
   (and (not (holding ?x))(clear ?x)(handempty)(ontable ?x)
            (not (DCK_holding ?x))(DCK_goodtower ?x)
        ))

(:action put-down_DCK-put-down-b
   :parameters (?x - block)
   :precondition (and (holding ?x)
                      (mg_Stacked ?x)(DCK_holding ?x)
                 )
   :effect
   (and (not (holding ?x))(clear ?x)(handempty)(ontable ?x)
        (not (DCK_holding ?x))(DCK_badtower ?x)
   ))
```

Enhanced operators referring to ATB-DCK transitions – DCK state change, constraints

	Original			ATB-DCK		
	C	P	Q	C	P	Q
barman (30)						
FF	0	9000	0.00	0	9000	0.00
lama	3	8142	2.72	**30**	42	**30.00**
mercury	27	1004	20.76	**30**	**40**	**30.00**
FDSS	26	1442	25.15	20	3268	19.84
BFWS	0	9000	0.00	0	9000	0.00
blocksworld (30)						
FF	0	9000	0.00	**30**	1	**30.00**
lama	29	364	15.06	**30**	**0**	**30.00**
mercury	19	3320	7.64	**30**	1	**30.00**
FDSS	26	1263	15.62	**30**	1	**30.00**
BFWS	4	7813	1.50	**30**	**0**	**30.00**
depots (30)						
FF	1	8703	0.67	**30**	**0**	**30.00**
lama	1	8722	0.41	**30**	1	**30.00**
mercury	0	9000	0.00	**30**	1	**30.00**
FDSS	20	3563	19.97	**30**	1	27.34
BFWS	12	5460	7.71	**30**	**0**	**30.00**
gripper (30)						
FF	0	9000	0.00	0	9000	0.00
lama	12	5575	11.18	**30**	**38**	**30.00**
mercury	0	9000	0.00	18	3832	18.00
FDSS	0	9000	0.00	10	6208	10.00
BFWS	0	9000	0.00	0	9000	0.00
spanner (30)						
FF	0	9000	0.00	**30**	8	**30.00**
lama	0	9000	0.00	**30**	13	**30.00**
mercury	0	9000	0.00	**30**	**4**	**30.00**
FDSS	0	9000	0.00	**30**	49	**30.00**
BFWS	0	9000	0.00	11	5701	11.00

(C)overage, (P)AR10 (in seconds) and IPC (Q)uality Score

Generalized Planning

- Planning programs [Segovia-Aguas et al., 2019]
- General planning policies [Frances et al., 2021]
- “Sketches” [Drexler et al., 2021]
- …

All the above can be learnt from small training problems (to some extent)

Impact of DCK

- Macros and Entanglements **have considerable impact on performance** in some cases
- Transition-based DCK in some cases **“determinize”** the planning process

- Changes in the domain model might require **considerable changes** in DCK

Impact of DCK on the KE process

- In practice, separating the “raw” domain model and DCK is easier to maintain
- Extend existing KE tools (e.g. itSimple, Planning.Domains) by supporting automatic/manual DCK acquisition
- Understanding in which cases planners fail and how DCK can alleviate such an issue
  - Even changing the order of operators and predicates in their preconditions/effects have a significant impact on planners' performance !

Conclusions

- KE process in planning is still “black art”
  - No guidelines/methodologies
  - Little support of KE tools
  - Effective DCK acquisition support
- A little to nothing has been done in non-classical planning
- **Addressing these issues will significantly strengthen the position of domain-independent planning in other AI areas**