Planning Modulo Theories: Extending the Planning Paradigm

Peter Gregory — Teesside University
Derek Long and Maria Fox — King's College London
J. Christopher Beck — University of Toronto

From SAT to SMT

SAT Instance:

Given a propositional formula, F, over propositional variables, V.

Question:

Is there a valuation on V that makes F true?

SMT Instance:

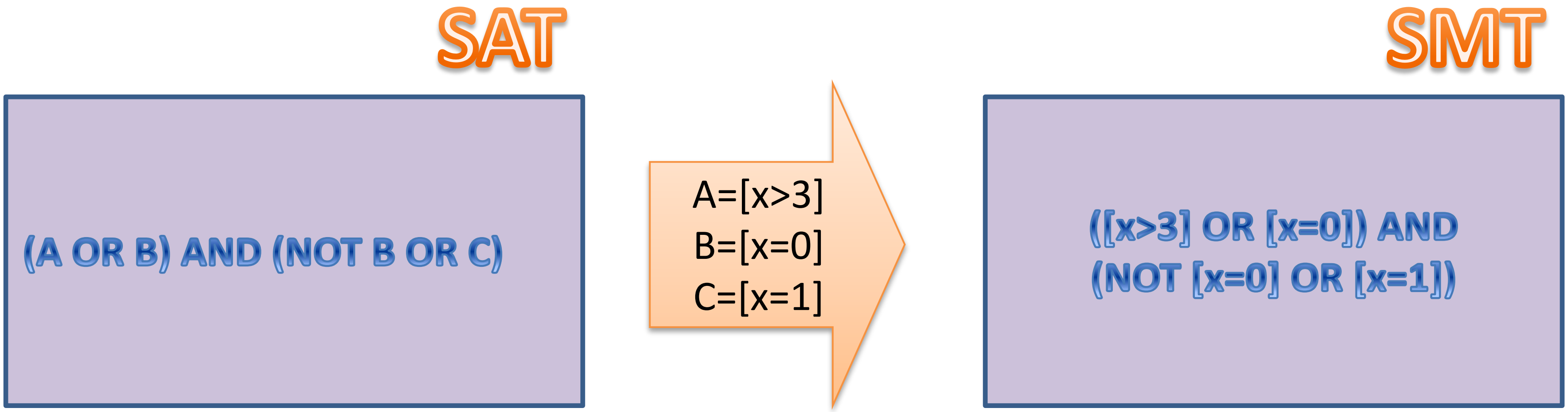
Given a *first order formula*, F, over constants, C, predicates, P, function symbols Fn, variables, V (of specified types) and a theory, T, defining the meanings of C, P, Fn (and types).

Question:

Is there a valuation on V that makes F true, subject to the constraints in T?

Solving SAT and SAT Module Theories, Nieuwenhuis, Oliveras and Tinelli, J. ACM, 2006

Simple Example



Solver links a **core** that performs search over propositional variables with **theory modules** to check satisfiability of conjunctions of literals within specific theories

From Planning to Planning Modulo Theories

- Classically, planning variables are propositions
  - Action *parameters*, drawn from finite enumerated sets, are not *variables* – the grounded literals of a problem are
- To find a plan, valuations for the variables must be found at each successive state
  - Action(s) selected at each transition to support these valuations
- Action preconditions are *propositional* sentences
- Action effects are assignments to *propositional* variables

- **PMT:**
  - Action preconditions are first order formulae over symbols with associated theories
  - Action effects assign values to variables of specified types (appropriate to the theories in use)

PMT Example

Classical

```
DRIVE (T, L1, L2,FL1, FL2)
PRE:   (and (at T L1)
            (fuel T FL1)
            (prev L1 L2))
EFF:  (and (at T L2) (fuel T FL2)
            (not (at T L1))
            (not (fuel T FL1)))
```

PMT

```
DRIVE (T, L2)
PRE:   (> (fuel T) 0)
EFF:  (and
         (assign (fuel T) (- (fuel T) 1))
         (assign (location T) L2))
```

(fuel ?t – truck) is a family of variables of type *Number*
(location ?t – truck) is a family of variables of type *Location*
(which is a finite enumerated type in this example)

PMT

```
TRANSFER (T1, T2)
PRE:   (= (at T1) (at T2))
EFF:   (and (assign   (in T1)    emptySet)
            (assign   (in T2)   (union (in T1) (in T2))))
```

Models for PMT

- Domains are described using expressions built over theories
- Theories are attached to the domain description using a module description language
  - Modules specify the names and type signatures of the relevant symbols
  - Types can be *infinite* and structured eg: Sets, Multisets, Lists, Arrays
- Each module is supported by an implementation of an *evaluation* function and a *satisfaction* tester for the expressions and literals defined by the signatures

Also relevant: **Geffner's** Functional Strips (2000); **Helmert's** multi-valued fluents in PDDL; **Dornhege et al** *Semantic attachments for domain independent planning systems* (ICAPS 09)

Planning with PMT

- Possibility 1: Translate PMT $\rightarrow$ SMT using Planning $\rightarrow$ SAT as starting point
  - Exploits advances in SMT
  - Fails to exploit the structure of planning problems
  - In practice, performance is poor

PMT→SMT

Problem	PMT-as-SMT (Z3) secs	MetricFF secs
Depots 03	57.59	0.03
Depots 04	85.43	0.38
Driverlog 12	10841.05	0.04
Driverlog 13	2102.93	0.34
Driverlog 14	3416.82	0.63
Rovers 07	21.65	0.05
Rovers 08	73.85	0.02
Reorder array with swaps (5)	1.2 [5 steps]	
Reorder array with swaps (10)	33.4 [6 steps]	
Reverse array segment (6)	11.1 [3 steps]	
Reverse+swaps (5)	0.8 [2 steps]	
Reverse+swaps (10)	32.9 [3 steps]	

Planning with PMT

- Possibility 1: Translate PMT $\rightarrow$ SMT using Planning $\rightarrow$ SAT as starting point
  - Exploits advances in SMT
  - Fails to exploit the structure of planning problems
  - In practice, performance is poor

- Possibility 2: Build a PMT planner...

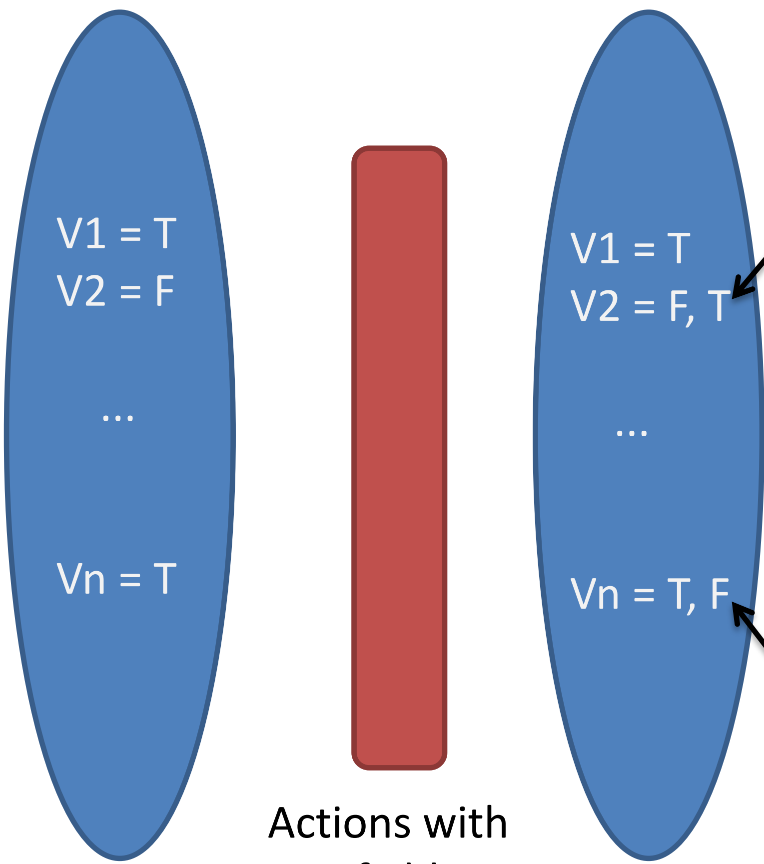
Heuristic Forward Search for PMT

- We extend the h_{max} heuristic from the propositional case to the general PMT case
- Recall: h_{max} heuristic is computed as the length of the shortest (parallel) plan in a relaxed state space
- One way to think about the relaxation is as "ignoring delete effects"
- We now consider a different view of the relaxed state space

Relaxed Reachability

Initial state

Accumulated state



Actions with satisfiable preconditions

Add effect assigns true

Assignments accumulate in variable domains

A (precondition) sentence is considered true if *some* assignment from the set satisfies it

Delete effect assigns false: when there are no negative preconditions, delete effects can be ignored in practice

Domain Abstraction

- Each variable, v, in the problem has a domain, D_v, and an associated *domain abstraction* $\mathcal{A}(D_v)$
- Each value in D_v is mapped to a value in $\mathcal{A}(D_v)$ by an abstract interpretation function
- The abstracted domain for a given type has corresponding abstract interpretations of the symbols for the theory including the type

- To build an abstracted reachability graph we start by abstracting the initial state and then:
  - For each action whose precondition is satisfied under the abstract interpretation
    - Apply the effects by abstract interpretation of the assignment within the current abstract state

An Example

- MetricFF can be interpreted in this way:
- For numeric variables the abstract interpretation uses the domain of *real-valued intervals*
- Predicates such as < (and functions such as +) are abstractly interpreted in the obvious ways (achieving relaxations)

  eg:

  > (V < W) if the lower bound on the interval for V is smaller than the upper bound on the interval for W
  >
  > (V + W) is the interval with lower bound equal to the sum of the lower bounds on V and W etc.

- Effects are implemented by making each assignment to a numeric variable extend its interval to include the new value

Example Abstractions

- One of the simplest abstract interpretations is an enumerated set of reachable values:
  - $\mathcal{A}(D_v)$ = power set of D_v
  - Abstracted predicates are satisfied if *some* value in the abstract interpretation satisfies the base predicate
  - Assignment V := W is handled by adding to the set for V all the set of possible values of W

- Another abstraction is the *finite abstraction* where we use $\mathcal{A}(D_v)$ = power set of *a finite subset of* D_v coupled with a special value “top”
  - Use enumerated abstraction over the subset and “top” when additional values are required

Finite Abstractions: Choosing a Basis

- Automatic strategy for basis selection - we generate a set of constants using a probing strategy:
  - We perform an initial exploration of the space by selecting actions that maximise the introduction of new constants, until we reach the goals (in the relaxed space)
  - This set is then used as the basis throughout all subsequent reachability analyses

- By removing “top” from the finite abstraction we create an inadmissible heuristic (some actions can be incorrectly considered inapplicable), but it is more informed
  - In this case we assign high values to apparent dead ends and leave them in the search space to ensure completeness

Implementation

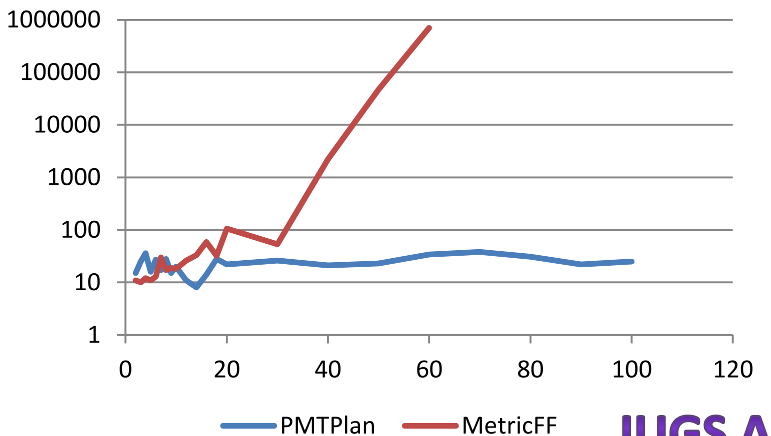
- We have implemented PMT Plan, which uses these abstractions, and applied it to benchmark problems using infinite types:
  - Integers (Jugs and Water)
  - Sets (Dump-trucks, Storytellers)
  - Multisets (Airport – new encoding)

PMTPlan using $\mathbf{h_{max}}$ with finite basis, enumerated set abstraction heuristic

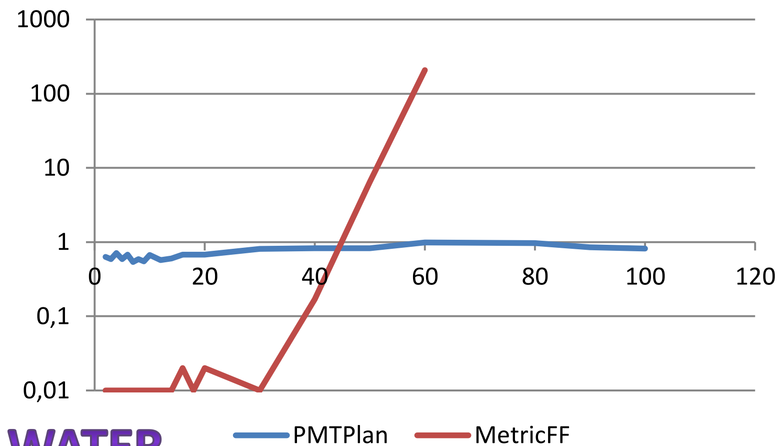
ome Resu

MetricFF standard implementation; Storytellers solved by using best mechanism encoding for the set behaviours

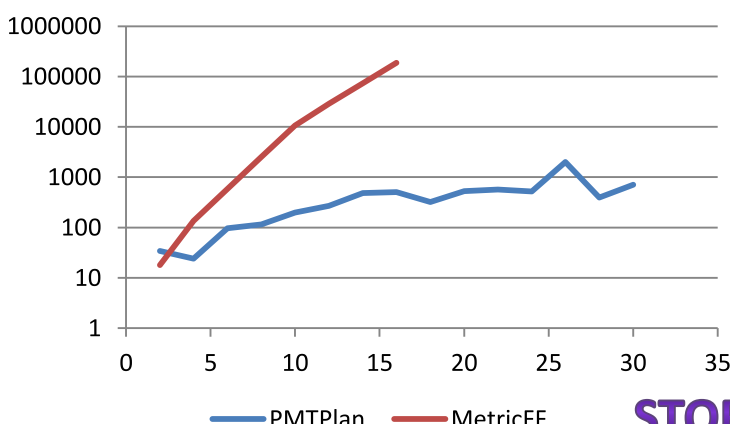
Nodes Generated

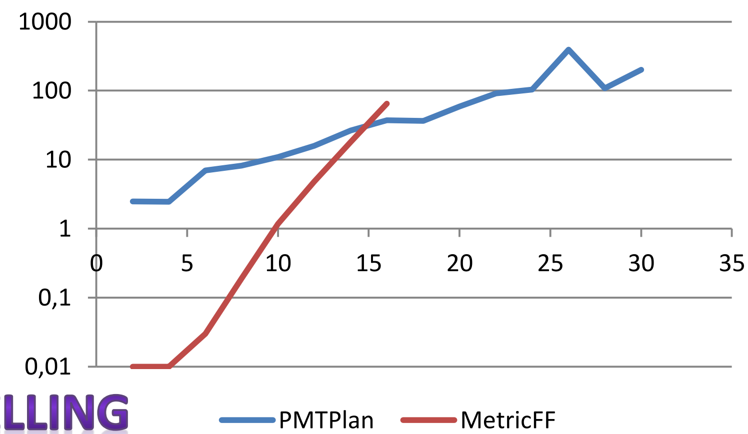


Time Taken



JUGS AND WATER





STORYTELLING

Lots More Types and Other Possibilities

- We are already considering lots of other interesting types, including “robot configurations”, voltages and power-flows (in circuits) and angles (trig functions)

- We are exploring ways to improve the heuristic from h_{max} to more informed variants
  - A challenge in implementing h_{FF} say, is in assigning responsibilities for the changes in values in an abstract domain to actions in the preceding layer
- A key aspect of SMT is the communication of no-goods from theory-solvers back to the core solver: we are considering similar techniques

- We want to integrate temporal planning with PMT planning, exploiting prior work in Crikey and POPF

	PMTPlan		MetricFF	
Jugs	Nodes	Time	Nodes	Time
02	34	2.47	18	0.01
04	24	2.44	136	0.00
06	97	6.94	582	0.03
08	116	8.20	2516	0.19
10	198	10.89	10564	1.17
12	270	15.79	28740	4.79
14	484	26.58	73558	18.00
16	507	37.07	186206	64.51
18	323	36.26		
20	529	58.70		
22	568	91.41		
24	524	104.10		
26	1995	392.32		
28	395	108.66		
30	707	201.68		

Table 2: Jugs and Water with increasing numbers of jugs.

	PMTPlan		MetricFF	
Packages	Nodes	Time	Nodes	Time
10	1247	11.44	54658	1.07
12	1855	16.78	84759	2.83
15	10933	52.60	271705	47.90
17	20247	156.83	551430	215.14
20	49414	1095.77		

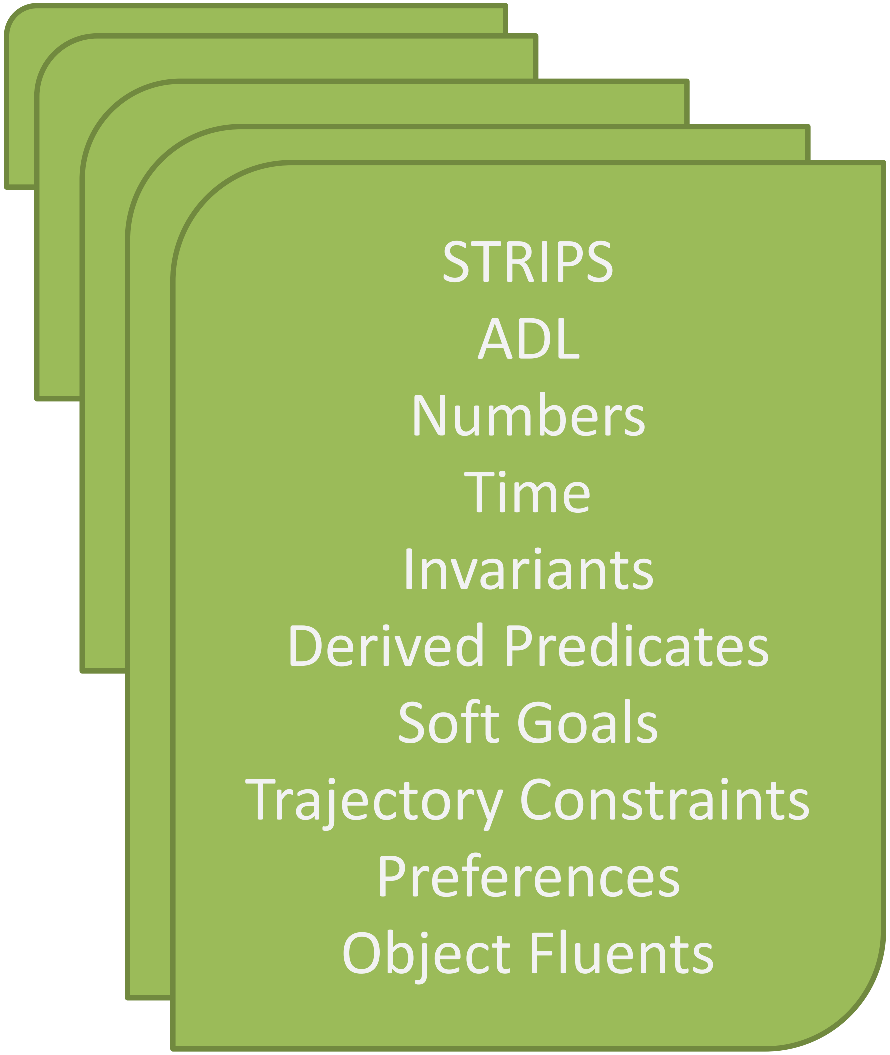
Table 3: Dump-trucks problems with increasing numbers of packages.

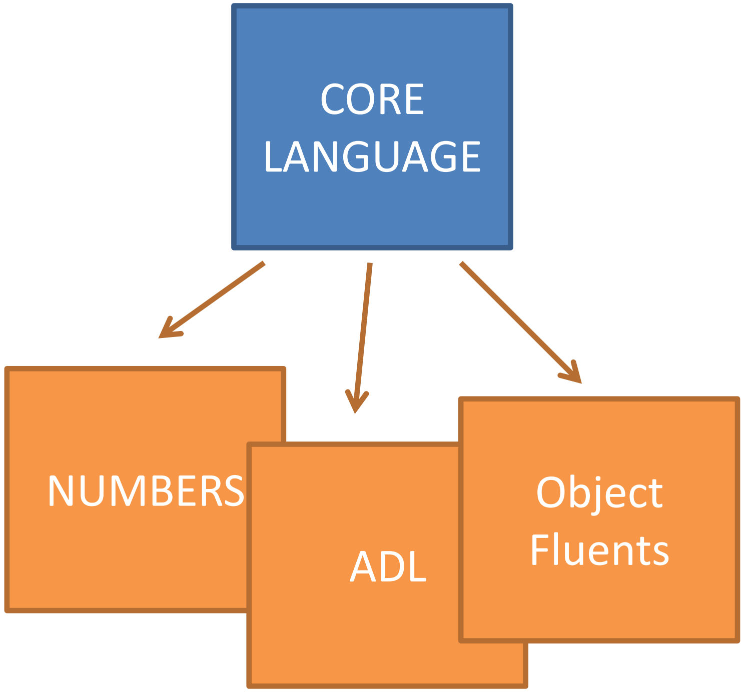
Extensions of PDDL

- Over successive extensions, PDDL supports:
  - ADL
  - Numbers
  - Time
  - Derived predicates
  - Soft constraints
  - Action Costs
  - Object fluents
  - Various specialised additions

- Each extension has required a new revision of the PDDL syntax and interactions between extensions are not always fully resolved

Motivation for PMT





The Integer Module

```
(define (module integer)
  (:type integer)
  (:functions
    (<        ?x - integer ?y - integer) - boolean
    (>        ?x - integer ?y - integer) - boolean
    (<=       ?x - integer ?y - integer) - boolean
    (>=       ?x - integer ?y - integer) - boolean
    (+        ?x - integer ?y - integer) - integer
    (-        ?x - integer ?y - integer) - integer
    (/        ?x - integer ?y - integer) - integer
    (*        ?x - integer ?y - integer) - integer
    (increase ?x - integer ?y - integer) - unit
    (decrease ?x - integer ?y - integer) - unit
  )
)
```

The Set Module

```
(define (module set)
  (:type set of a')
  (:functions
    (cardinality ?s – set of a')                   - integer
    (member     ?s - set of a' ?x – a')            - boolean
    (subset     ?s1 ?s2 – set of a')               - boolean
    (union      ?x – set of a' ?y – set of a')     – set of a'
    (intersect ?x - set of a' ?y – set of a')      – set of a'
    (difference ?x – set of a' ?y – set of a')     – set of a'
    (add-element ?s - set of a' ?x – a') – set of a'
    (rem-element ?s - set of a' ?x – a') – set of a'
    (empty-set)                          - set of a'
    (construct-set ?x+ - a')             - set of a'
  )
)
```