Planning Modulo Theories: Extending the Planning Paradigm

Peter Gregory    Derek Long and Maria Fox    J. Christopher Beck
Teesside University    King’s College London    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_
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

(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)
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 → SMT using Planning → SAT as starting point
  – Exploits advances in SMT
  – Fails to exploit the structure of planning problems
  – In practice, performance is poor
<table>
<thead>
<tr>
<th>Problem</th>
<th>PMT-as-SMT (Z3) secs</th>
<th>MetricFF secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depots 03</td>
<td>57.59</td>
<td>0.03</td>
</tr>
<tr>
<td>Depots 04</td>
<td>85.43</td>
<td>0.38</td>
</tr>
<tr>
<td>Driverlog 12</td>
<td>10841.05</td>
<td>0.04</td>
</tr>
<tr>
<td>Driverlog 13</td>
<td>2102.93</td>
<td>0.34</td>
</tr>
<tr>
<td>Driverlog 14</td>
<td>3416.82</td>
<td>0.63</td>
</tr>
<tr>
<td>Rovers 07</td>
<td>21.65</td>
<td>0.05</td>
</tr>
<tr>
<td>Rovers 08</td>
<td>73.85</td>
<td>0.02</td>
</tr>
<tr>
<td>Reorder array with swaps (5)</td>
<td>1.2 [5 steps]</td>
<td></td>
</tr>
<tr>
<td>Reorder array with swaps (10)</td>
<td>33.4 [6 steps]</td>
<td></td>
</tr>
<tr>
<td>Reverse array segment (6)</td>
<td>11.1 [3 steps]</td>
<td></td>
</tr>
<tr>
<td>Reverse+swaps (5)</td>
<td>0.8 [2 steps]</td>
<td></td>
</tr>
<tr>
<td>Reverse+swaps (10)</td>
<td>32.9 [3 steps]</td>
<td></td>
</tr>
</tbody>
</table>
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 $h_{\text{max}}$ heuristic from the propositional case to the general PMT case

• Recall: $h_{\text{max}}$ 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

$V_1 = T$
$V_2 = F$
...
$V_n = T$

Actions with satisfiable preconditions

Accumulated state

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:
  – $A(D_v) = \text{power set of } D_v$
  – 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 $A(D_v) = \text{power set of a finite subset of } D_v$ 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)
Some Results

- PMTPlan using $h_{\text{max}}$ with finite basis, enumerated set abstraction heuristic
- MetricFF standard implementation; Storytellers solved by using best mechanism encoding for the set behaviours

**Nodes Generated**

**Time Taken**
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 $h_{\text{max}}$ to more informed variants
  – A challenge in implementing $h_{\text{FF}}$ 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
<table>
<thead>
<tr>
<th>Jugs</th>
<th>PMTPlan</th>
<th>MetricFF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Time</td>
</tr>
<tr>
<td>02</td>
<td>34</td>
<td>2.47</td>
</tr>
<tr>
<td>04</td>
<td>24</td>
<td>2.44</td>
</tr>
<tr>
<td>06</td>
<td>97</td>
<td>6.94</td>
</tr>
<tr>
<td>08</td>
<td>116</td>
<td>8.20</td>
</tr>
<tr>
<td>10</td>
<td>198</td>
<td>10.89</td>
</tr>
<tr>
<td>12</td>
<td>270</td>
<td>15.79</td>
</tr>
<tr>
<td>14</td>
<td>484</td>
<td>26.58</td>
</tr>
<tr>
<td>16</td>
<td>507</td>
<td>37.07</td>
</tr>
<tr>
<td>18</td>
<td>323</td>
<td>36.26</td>
</tr>
<tr>
<td>20</td>
<td>529</td>
<td>58.70</td>
</tr>
<tr>
<td>22</td>
<td>568</td>
<td>91.41</td>
</tr>
<tr>
<td>24</td>
<td>524</td>
<td>104.10</td>
</tr>
<tr>
<td>26</td>
<td>1995</td>
<td>392.32</td>
</tr>
<tr>
<td>28</td>
<td>395</td>
<td>108.66</td>
</tr>
<tr>
<td>30</td>
<td>707</td>
<td>201.68</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Packages</th>
<th>PMTPlan</th>
<th>MetricFF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Time</td>
</tr>
<tr>
<td>10</td>
<td>1247</td>
<td>11.44</td>
</tr>
<tr>
<td>12</td>
<td>1855</td>
<td>16.78</td>
</tr>
<tr>
<td>15</td>
<td>10933</td>
<td>52.60</td>
</tr>
<tr>
<td>17</td>
<td>20247</td>
<td>156.83</td>
</tr>
<tr>
<td>20</td>
<td>49414</td>
<td>1095.77</td>
</tr>
</tbody>
</table>

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

- STRIPS
- ADL
- Numbers
- Time
- Invariants
- Derived Predicates
- Soft Goals
- Trajectory Constraints
- Preferences
- Object Fluents

 CORE LANGUAGE

 NUMBERS

 ADL

 Object Fluents
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'
  )
)