Impact of Modeling Languages on the Theory and Practice in Planning Research

Jussi Rintanen

Department of Computer Science
Aalto University, Finland
(Also affiliated with Griffith University, Brisbane, Australia, and the Helsinki Institute of Information Technology, Finland.)

January 2015

This Talk

Goal

Modeling languages that

- are suitable for modeling realistic large-scale planning problems, and
- support efficient implementations

Current issues

- Languages lacking features necessary for effective modeling
- Low-level modeling which limits efficient implementation possibilities

Modeling Languages Used by the Planning Community (IPC&ICAPS)

- Most visible language is PDDL (1.0, 2.1). Other languages exist, but their use is limited.
- Many types of works on classical and temporal planning exclusively use the Planning Competition (IPC) benchmark sets $\sim$ PDDL
  - mostly toy problems (mix cocktails, park cars, naive logistics, ...)
  - small number of action schemas (often 3 to 6)
  - large instances obtained by increasing number of objects
- Most aspects of PDDL 2.1 were heavily criticized upon its introduction (Bacchus 2003; Boddy 2003; Geffner 2003; Smith 2003) and its limitations are widely recognized, but in last years have got little attention. (Many people seem perfectly happy with PDDL!)

PDL unnecessarily low level, misses important concepts

- PDDL 1.0 (classical planning)
  - no enumerated types
  - no (bounded) integers
  - standard benchmark formulations ignore existing features
- PDDL 2.1 (temporal planning)
  - resources $\subseteq$ state variables, no separate concept of resources
  - handling of action dependencies (implicit resources) incompatible with effective formulation with constraints (MILP, SAT, SMT, ...)
- No formal (= precise) semantics exists for either, which especially for PDDL 2.1 has been problematic. Correctness of implementations?

Why does it matter?

1. Low-level of models have negative implications w.r.t. performance
   - PDDL models force poor implementation details
   - Cleverly recovering the underlying high-level models tricky, uninteresting
2. Developing complex models tedious

Lack of Many-valued (Enum) State Variables

- Standard benchmark sets dominated by (implicit) many-valued variables, which have to be represented as Booleans
- Variable x with values $v_1, \ldots, v_n$ represented as Booleans $x_1, \ldots, x_n$
- Many heuristics defined in terms of many-valued variables:
  - Domain-Transition Graphs a standard starting point (Helmert 2004)
  - DTGs probably main reason 50 per cent of IPC 2014 planners using FD
- All constraint-based PDDL planners extract invariants to be able to recognize dependencies between $x_1, \ldots, x_n$.

Extraction of many-valued variables expensive: FD can spend several minutes doing it even in simple cases.
Extraction of invariants (mostly induced by many-valued variables) can similarly be expensive.

Lack of Bounded Integers

- integer variable n with range $0..N$ represented as Booleans $n_1, \dots, n_N$ (in several standard benchmarks)
- Could use PDDL reals, but many/most planners do not support them.
- increment of $0..N$ modeled as N actions each turning some i to $i+1$
- Does not seem to be an issue for explicit state-space search.
- Rintanen (AIJ 2012) observed correlation between poor performance with SAT-based planners and presence of implicit integers for standard benchmarks
- (*Could*, in principle, automatically "recognize" implicit integers, and derive sensible and efficient representations. (Uninteresting, makes little sense as scientific research))

Lack of Explicit Resources

- In PDDL 2.1 (implicit) resources are allocated by a two-step process:
  1. Confirm that given resource is available (precondition $x = 0$)
  2. Allocate the resource (assign $x := 1$ *at start*)

  This takes place inside a 0-duration critical section.
- Bad for constraint-based frameworks SMT, MILP, CP: conceptually two separate time points, one with $x = 0$ and another with $x = 1$, leading to ϵ gaps in plans, with an exponential performance penalty.

PDDL schedule			desired schedule		
$\mathit{move}_{a,b}$	$\mathit{move}_{b,c}$	$\mathit{move}_{c,d}$	$\mathit{move}_{a,b}$	$\mathit{move}_{b,c}$	$\mathit{move}_{c,d}$

- Explicit resources with allocations and deallocation preferable

What Should Be Done: A Proposal

Adopt a more modern modeling language to replace PDDL.

- Clean modern syntax, targeting human use (can also define XML/Lisp syntax if needed)
- Richer datatypes: enums, Booleans, (bounded) integers, rationals
- For temporal planning: explicit resources
- Support for modular construction of large-scale models
- Formal semantics

Role of classical planning needs to be re-assessed: most of "classical" planning actually temporal planning with time stripped off: standard benchmarks mostly multi-vehicle, multi-machine, multi-agent problems

What Should Be Done: A Proposal

Inspiration:

- automata-based formalisms (as used by Verification, Diagnosis communities), with synchronization of automata as a modular mechanism for building large models
- Functional STRIPS (Hector Geffner, 2000)

Requirements:

- Compatibility with leading search methods
  - constraint-programming
  - SAT, SMT
  - MILP

Benefits from A New Planning Modeling Language

- Easier to formulate more realistic and complex planning problems
- Bring Planning+Scheduling back to the agenda
- New research will emerge from more challenging and differently structured benchmark problems.

How to Transition to a New Language?

- Many planners already using higher-level models e.g. with many-valued variables, internally
  Half of classical planners in IPC-2014 used FD front-end: modify FD
  $\Rightarrow$ problem half solved for existing classical planners
- Temporal planning not very active area: only 6 planners in IPC-2014
- Opposition from inside IPC/ICAPS? Expected.
- Open question: Who will be modeling more complex planning & scheduling problems?

Disclaimers

- Real-world applications have varying modeling requirements
  - Every language has something essential missing
  - Futile and fruitless to try to include everything in one language
- Domain-independent planning may be doomed anyway
  - domain-independent $\not\rightarrow$ less modeling work
  - domain-specific constraints, representations, heuristics too often vital
  - relevance to both theory and practice questionable
  - Constraint Programming, MILP, SAT, SMT etc often better as "domain-independent" languages than any *planning* language

Conclusion

- PDDL abstraction level too low $\rightarrow$ issues with modeling and implementation
- Connection between Planning and Scheduling forgotten
- Both issues could be fixed with more modern modeling languages
- Potential for similar fast progress in research as right after PDDL's initial adoption in 1998