Domain Model Acquisition from Binary Traces

Arash Haratian[1] **Arnaud Lequen**[1] Daniel Gnad[1,2] Jendrik Seipp[1]

[1]Linköping University [2]Heidelberg University

Outline

Outline

Planners need symbols — the world gives us signals

- Domain-independent planners require a **symbolic model**: predicates, typed objects, action preconditions & effects.
- Real-world data is **subsymbolic**: sensor readings, neural-network encodings, raw logs, etc.

Can we learn a symbolic planning model *directly* from binary observations?

Domain Model Acquisition (DMA)

Learn a symbolic domain $\mathcal{D}$ (predicates + action schemas) from observed *traces* of states and actions.

- Existing methods assume states are given as **sets of fluents** and that action arguments are known.
- We drop that assumption: states are **opaque bit vectors**.

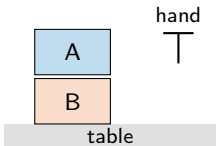
Contribution

- An approach to DMA from purely **binary-encoded** traces.
- BitL1 *simultaneously* learns
  1. the mapping from bits to symbolic fluents, and
  2. lifted action schemas (preconditions + effects).
- Output is standard PDDL

Outline

Classical planning in one slide

A **domain** $\mathcal{D}$ has typed **predicates** and **action schemas**; a **state** is a set of true **fluents**.



State:
`{on(A,B), on(B,table), handempty}`

Action schema `pickup(x, y)`

pre	`on(x,y), clear(x), handempty`
del	`on(x,y), handempty`
add	`holding(x), clear(y)`

Applying `pickup(A,B)`:

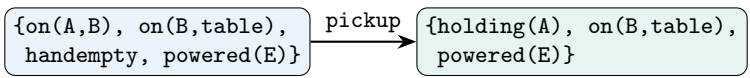
$$s' = (s \setminus del(\cdot)) \cup add(\cdot)$$

A *trace* is an alternating sequence of states and action labels $t = \langle s_0, \ell_1, s_1, \ell_2, \dots \rangle$.

Outline

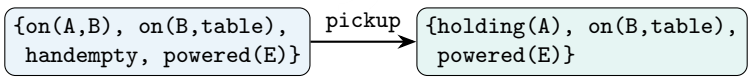
Same transition, two worlds

Symbolic setting: states are sets of fluents, action label known



Same transition, two worlds

Symbolic setting: states are sets of fluents, action label known



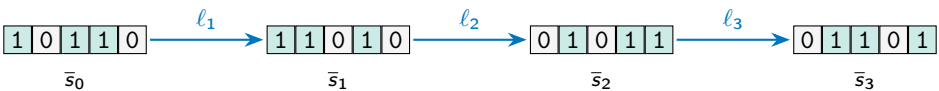
Subsymbolic setting: same states, but encoded as **bit vectors** (mapping unknown)

ordering `[on(A,B), on(B,table), handempty, holding(A), powered(E)]`

$$\bar{s}_b = [\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{1}] \xrightarrow{\text{pickup}} \bar{s}_a = [\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{1}]$$

The Subsymbolic DMA problem

Input: traces of binary state vectors linked by action labels.

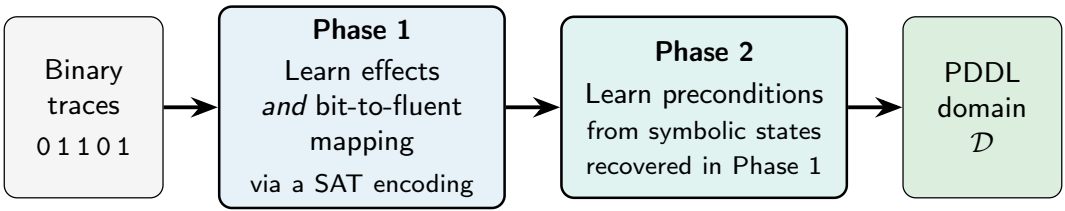


Goal: find...

- Action schemas $\mathcal{A}$ (preconditions + effects)
- A bit-to-fluent mapping σ^i for each trace.

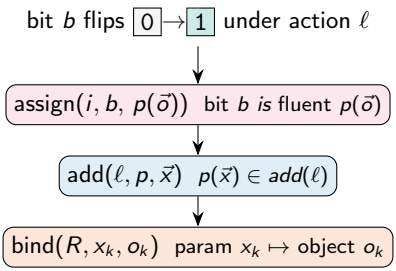
Outline

Generalises the **L1** algorithm [Balyo et al., KR 2024] from the symbolic to the subsymbolic case.



- **Phase 1** jointly fixes *what each bit means* and the *action effects*
- **Phase 2** reuses the recovered symbolic states to read off preconditions

For every transition and every **changed bit** b, encode all consistent assignments as a SAT formula.



$$\textsc{Eff}(t^i, b, R) := \bigvee_{p \in \mathcal{P}^D} \bigvee_{\vec{o} \preceq p} \Big(\text{assign}(i, b, p(\vec{o})) \wedge \bigvee_{\vec{x}} \text{add}(\ell, p, \vec{x}) \wedge \textsc{Bind}(R, \vec{x}, \vec{o}) \Big)$$

- One global formula over the whole trace $i \Rightarrow$ a *consistent* mapping.
- A satisfying assignment yields the effects and the bit-to-fluent map σ^i.

Phase 1 gives, for each transition, the objects bound to each action parameter $\Rightarrow$ we can recover **symbolic states**.

Lift, then intersect

For each transition: keep the pre-state fluents over bound objects, and *lift* them (objects $\rightarrow$ parameters). The precondition is what holds **across all** transitions of the action:

$$pre^{+}(\ell) = \bigcap_{R=(s_b,\ell,s_a)} \{\, p(u_R^{-1}(\vec{o})) \mid p(\vec{o}) \in s_b \,\}$$

- Same subroutine as $\mathrm{L1}$, but it depends on the mapping from Phase 1.
- Also consider negative preconditions, with a symmetric construction over fluents *absent* from s_b ($pre^{-}(\ell)$).

Outline

Setup

Benchmarks

- 18 IPC domains (same set as L1).
- 16 STRIPS + 2 with negative preconditions (Termes, Tidybot).
- Symbolic traces re-encoded as bit vectors.

Baseline: L1

- Solves the *symbolic* DMA problem.
- Sees **more** information than BitL1.
- Re-implemented in Python + PySAT.

- Most domains solved in < 5 min; learned models written to PDDL.
- Goal: how close are BitL1's models to the reference, given only bits?

Results

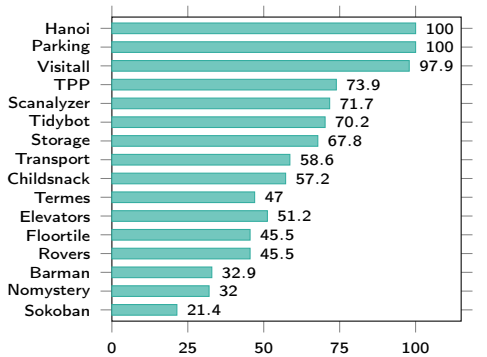
Effects

BitL1 **matches or beats** L1 on **10 / 16** solved domains, despite seeing only bits.

Preconditions

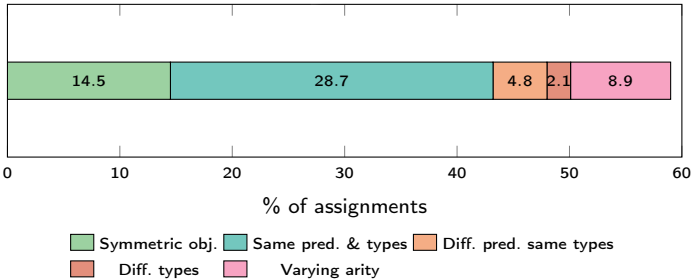
- **Fewer redundant** preconditions than L1.
- Misses static ones (invisible in bits).
- Negative preconditions learned well (Tidybot: 21 learned, only 3 missed).

Bit-to-fluent mapping accuracy (% w/ symmetry)



Where the mapping bends, and why it still works

Categorising imperfect bit-to-fluent assignments (avg. % across domains):



Take-aways

BitL1: domain models from binary traces

- First DMA method that works on **purely subsymbolic** states.
- SAT encoding **jointly** learns the bit-to-fluent mapping and the action effects; preconditions follow (incl. negative ones).
- Matches L1 on most domains while seeing far less information; outputs ready-to-use PDDL.

Future work

- Continuous / neural encodings $\rightarrow$ learn straight from sensor data.
- Numeric PDDL; object-level cues to shrink the search space.

Take-aways

BitL1: domain models from binary traces

- First DMA method that works on **purely subsymbolic** states.
- SAT encoding **jointly** learns the bit-to-fluent mapping and the action effects; preconditions follow (incl. negative ones).
- Matches L1 on most domains while seeing far less information; outputs ready-to-use PDDL.

Future work

- Continuous / neural encodings $\rightarrow$ learn straight from sensor data.
- Numeric PDDL; object-level cues to shrink the search space.

Thank you! Questions?