Domain-Independent Reward Machines for Modular Integration of Planning and Learning

Giuseppe De Giacomo, **Marco Favorito**, Luca Iocchi and Fabio Patrizi

Sapienza University of Rome

Context: Integrating Planning and Learning

The integration of Planning and Learning has many advantages:

- High-level Reasoning + Low-level Adaptivity
- Model based -> Sample efficiency
- Decomposable tasks and reuse of sub-policies

In this work*:*

- high-level symbolic action models and plans (Symbolic Planning)
- low-level reward-based control (Reinforcement Learning)

Related Work and Limitations

- HRL and Options framework (Sutton, Precup, and Singh 1999)
  - Non-trivial modelling effort
  - Mapping between the different representation layers

- Plans that produce rewards to drive RL agents (Grzes and Kudenko, 2008)
  - Cannot easily transfer learned policy to new tasks

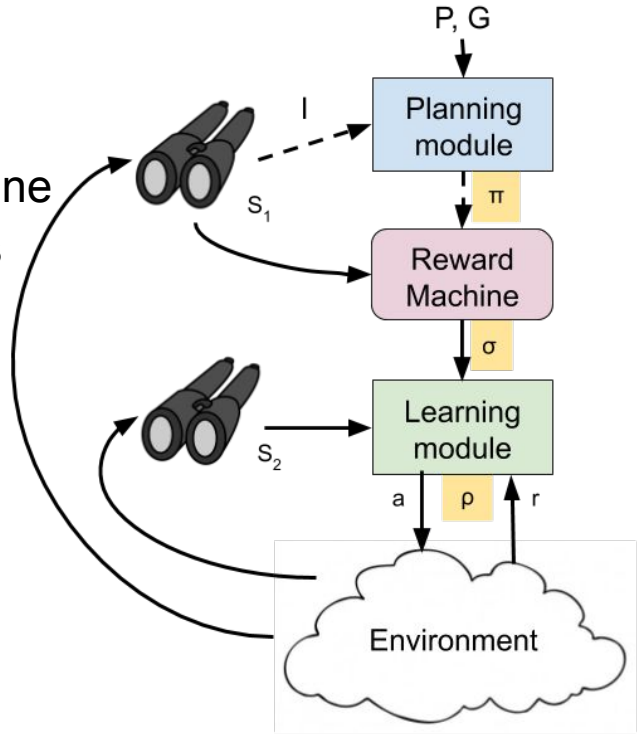
- Reward Machines (Icarte et al. 2018)
  - Automaton is manually specified
  - Explicit mapping is required

Domain-independent Reward Machines

Our approach:

- High-level decision making generates a reward machine
- RL learns optimal policy maximizing received rewards
- Automatic generation of options
- **No explicit mapping between state spaces**

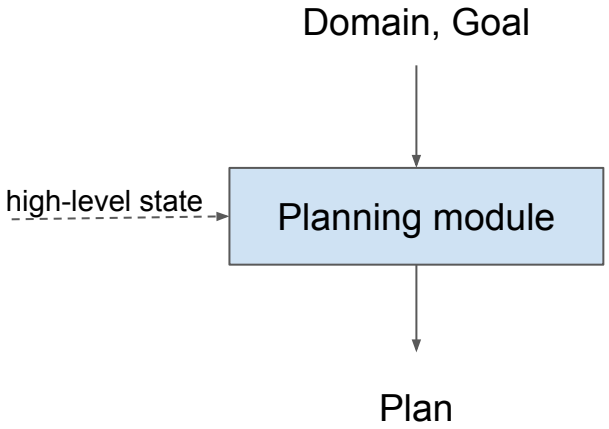
Similar to Taskable RL (Illanes et al. 2020), except that it allows greater **decoupling** between planning and learning modules



Planning Module

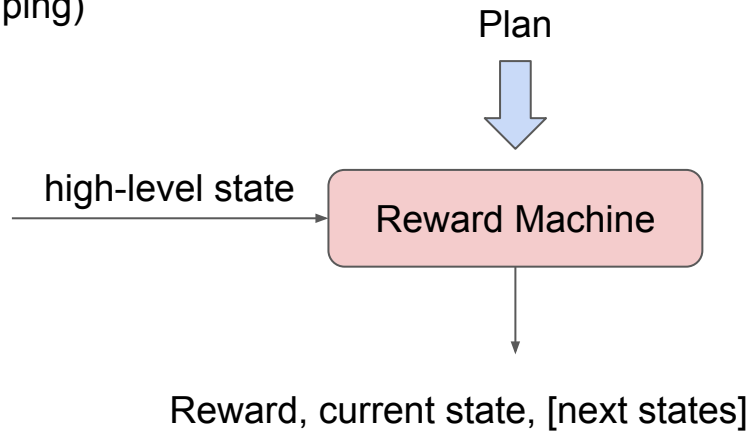
- Inputs (offline):
  - a (deterministic) **planning domain** and
  - a **goal**
- Output (offline): a **plan**
  - Computed using any off-the-shelf planner

Could also monitor the environment and recompute the plan.



Reward Machine

- A reward machine is generated from a **plan**
  - From the plan, get the transition graph
  - Add rewards on transitions (e.g. using reward shaping)
- **Input**: new high-level env state
- **Outputs**:
  - Task rewards
  - Current goal state
  - Available next (high-level) goal states



Two roles:

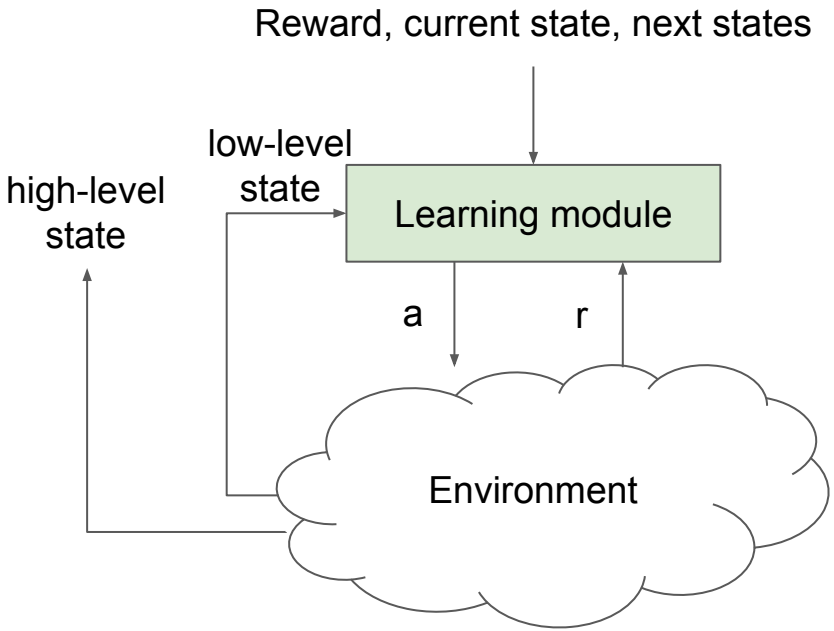
- Monitoring of the learning agent via rewards
- Machine transitions can be seen as **options**

Learning Module

- Input: **reward**, **current goal state**, low-level **env state**
- Output: a **policy**

Remarks:

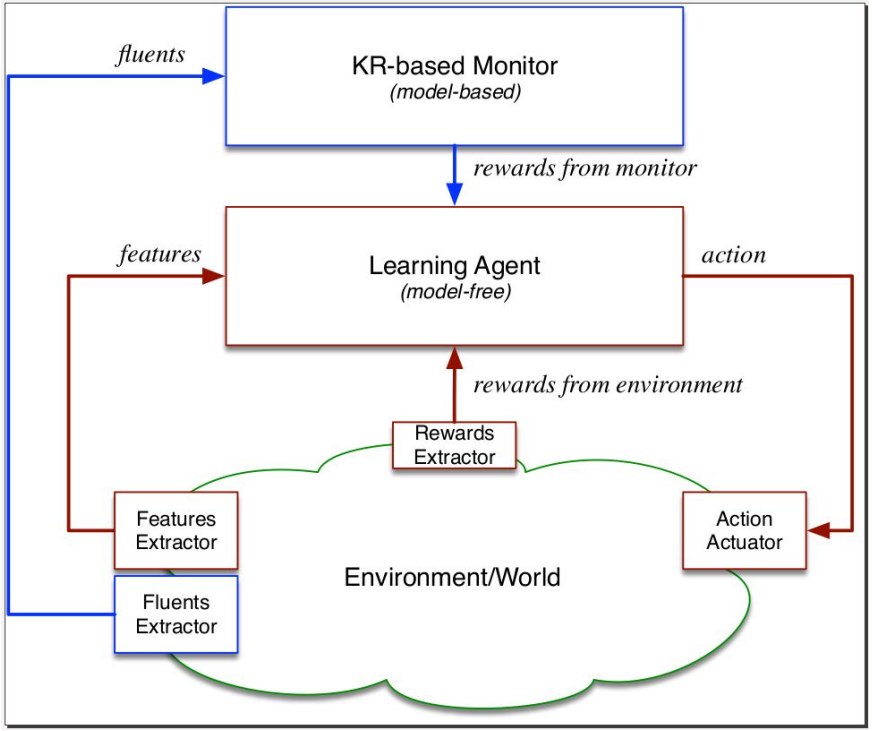
- Agnostic w.r.t. learning algorithm
- *Can* use options
- **Decoupled** from high-level representation



Domain-independence

From the learning module's point-of-view, setting is the same of **Restraining Bolts (De Giacomo et al. 2019)**

Agent only needs to keep track of RB states and agent's features (not the fluents)



G. De Giacomo, M. Favorito, L. Iocchi, and F. Patrizi. Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications. ICAPS, 2019.

Automaton transitions as sub-tasks

Next goal states can be exploited to improve sample efficiency:

- Sub-task exploration policies:
  - Epsilon-greedy with sub-task epsilon decay
  - Probabilistic choice base on success rate of transition (q_t, q_t+1)
- Meta-controller of options
  - Choose the current best available macro-action
  - Observe total return from an automaton transition
  - Update Q-function of meta-controller
- Shared sub-tasks allow reuse of acquired knowledge

Example: Taxi Environment (Dietterich, 2000)

A Taxi has to move on a gridworld, pick-up a passenger, and drop-off him to a specific location (Red, Gree, Yellow, Blue).

Planning domain: goto X, pick-up and drop-off

Learning domain: east, west, north, south, pick-up, drop-off

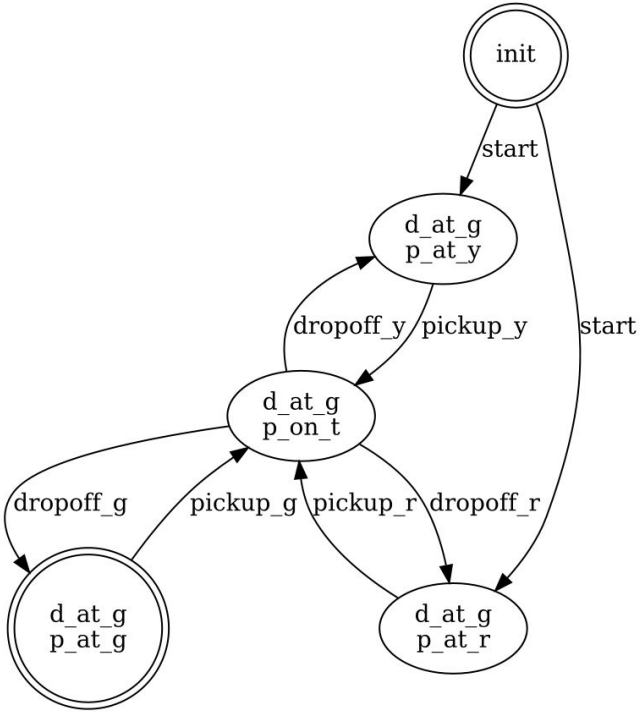
Example: Taxi Environment (Dietterich, 2000)

Simplified Taxi planning domain. Fluents:

- **p_at_X**: passenger is at location **X**
- **p_on_t**: passenger is on the taxi
- **d_at_X**: final destination is **X**

At the beginning:

- the passenger can be either at Red or at Yellow.
- The destination is always Green.

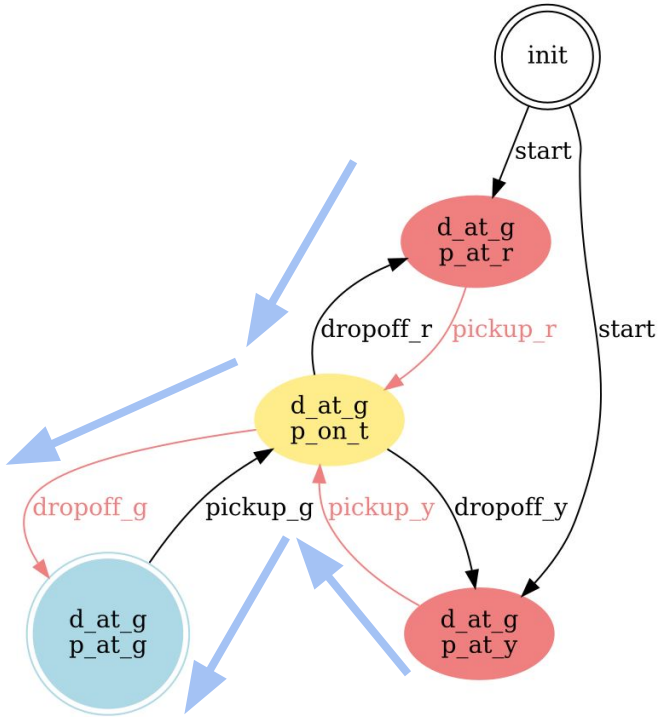


Example: Taxi Environment (Dietterich, 2000)

- In Red, the initial states
- In Blue, the goal state

The two sequences of arrows are the plans for two different tasks.

The Yellow state is shared between the two plans, no need to learn again from it

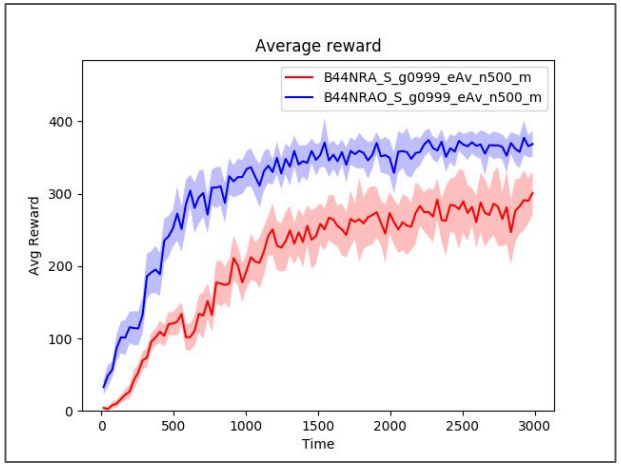


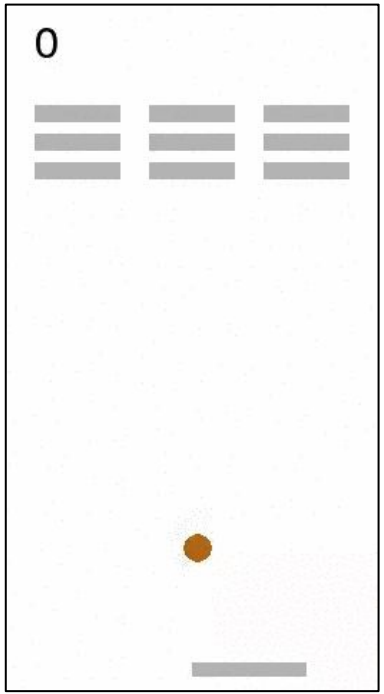
Example: Breakout (De Giacomo et al. 2019)

Task: remove columns in a specific order (e.g. left to right)

Planning domain: only models column removal

Learning domain: low-level paddle control and sensing



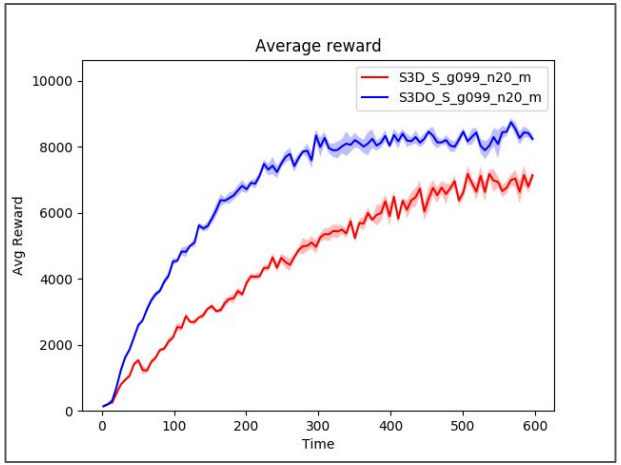


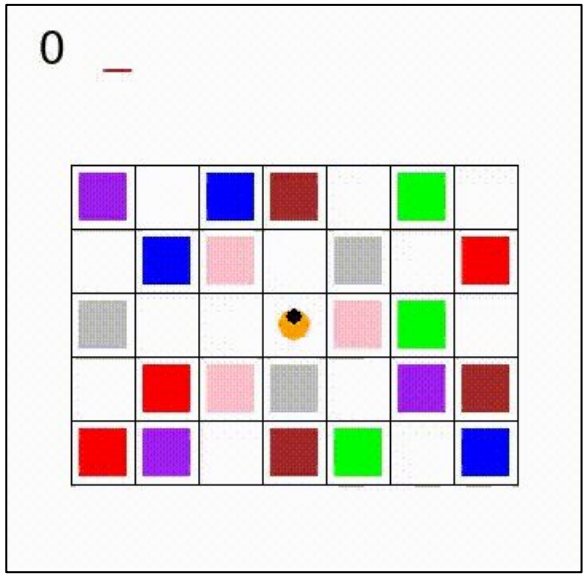
Example: Sapientino (De Giacomo et al. 2019)

Task: visit colors in a certain order

Planning domain: only models high-level moves

Learning domain: low-level grid movement controls



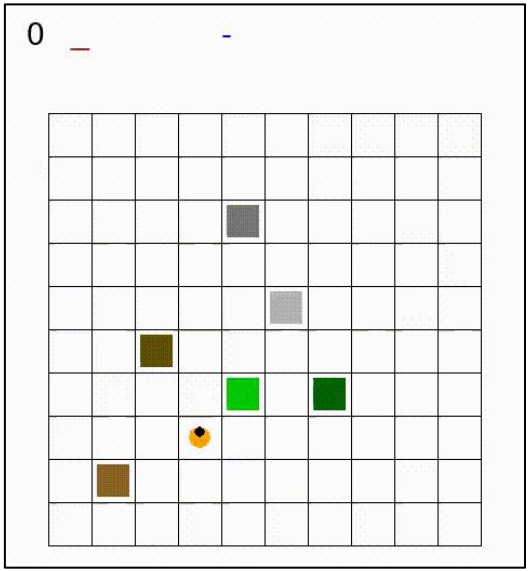


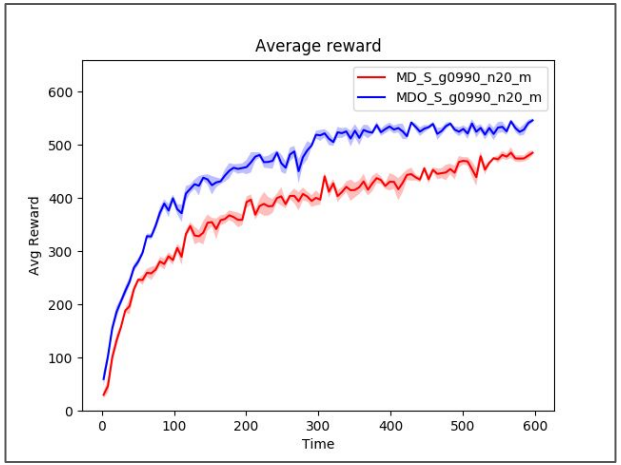
Example: Minecraft (De Giacomo et al. 2019)

Task: use tools and collect resources

Planning domain: goto location, collect res., use tool

Learning domain: low-level grid movement controls





Conclusions

- Modular Integration of symbolic planning and RL
- Planning and Learning modules are decoupled
- Yet, the agent can potentially optimize for the plan
  - if the environment allows it
- Sample efficient and decomposable

Future work:

- Non-deterministic planning domain
  - plans as graphs rather than sequences
- Online planning: react to unforeseen events
- Model refinement from learning module feedback