1 SAT-based Automata Learning

SAT-based automata learning refers to a method that uses satisfiability (SAT) solvers to construct automata, such as Reward Machines from observed data (in our case, trajectories). In Algorithm 1, the `LearnRM` function (Line 19) constructs this RM from counterexamples X using SAT-based automata learning, inspired by [1, 2, 3] (established method for learning automata). Specifically, `LearnRM` encodes trajectories' constraints into an SAT formula, defining states, transitions (δ), and rewards (σ) of the RM. It iteratively increases the state count until a minimal RM consistent with all counterexamples is found, solved via an SAT solver (satisfiability solvers such as PySAT). Performance-wise, this ensures equivalence to the ground truth RM almost surely, as proven in Lemma 1, with complexity tied to the number of states and trajectories (bounded by Γ).

Algorithm 1: DPLL-recursive(F, ρ)

Input: A CNF formula F and an initially empty partial assignment ρ
Output: UNSAT, or an assignment satisfying F

1 **Procedure** `DPLL-recursive(`F, ρ`)`
2     $(F, \rho) \leftarrow$ `UnitPropagate`(F, ρ);
3   **if** F *contains the empty clause* Λ **then**
4   **return** UNSAT;
5   **end**
6   **if** F *has no clauses left* **then**
7   Output ρ;
8   **return** SAT;
9   **end**
10     $\ell \leftarrow$ a literal not assigned by ρ ; `// the branching step`
11   **if** `DPLL-recursive` $(F \mid \ell, \rho \cup \{\ell\}) = SAT$ **then**
12   **return** SAT;
13   **end**
14   **return** `DPLL-recursive` $(F \mid \neg\ell, \rho \cup \{\neg\ell\})$;
15 **end**
16 **Procedure** `UnitPropagate(`F, ρ`)`
17   **while** F *contains no empty clause* Λ *but has a unit clause* x **do**
18         $F \leftarrow F \mid x$;
19         $\rho \leftarrow \rho \cup \{x\}$;
20   **end**
21   **return** (F, ρ);
22 **end**

There are many algorithms for SAT solving, but the DPLL algorithm is the base algorithm for most modern SAT solvers [4]. The DPLL (Davis-Putnam-Logemann-Loveland) algorithm [5] is a systematic search procedure for determining the satisfiability of a Boolean formula in Conjunctive Normal Form (CNF), denoted as F. It uses a recursive approach with partial assignment

ρ, initially empty, to assign truth values to variables. The algorithm simplifies F through unit propagation, handled by the sub-procedure `UnitPropagate`, which sets unit clauses (clauses with one literal, denoted x) to true. If F contains the empty clause Λ, it returns UNSAT, indicating unsatisfiability. If no clauses remain, it returns SAT with ρ as a satisfying assignment. Otherwise, it branches on an unassigned literal ℓ, recursively exploring both ℓ and its negation $\neg\ell$, ensuring a complete search for a solution.

References

[1] Daniel Neider. *Applications of automata learning in verification and synthesis.* PhD thesis, Aachen, Techn. Hochsch., Diss., 2014, 2014.

[2] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 590–598, 2020.

[3] Daniel Neider, Jean-Raphael Gaglione, Ivan Gavran, Ufuk Topcu, Bo Wu, and Zhe Xu. Advice-guided reinforcement learning in a non-markovian environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9073–9080, 2021.

[4] Carla P Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. *Foundations of Artificial Intelligence*, 3:89–134, 2008.

[5] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.