Department of Computer Science
Toronto Metropolitan University (TMU)

Planning as Reasoning in the Real World

October 17, 2024

Mikhail Soutchanski

Why theorem proving approach to planning?

We consider a deductive approach to planning. This approach is formulated in the Situation Calculus (SC) where the initial theory $\mathcal{D}_{S_0}$ is in first order logic (FOL) and a goal can be a FOL formula.

This allows for representation of a broad class of planning problems.

- Can plan without the Domain Closure Assumption (DCA) that restricts to finitely many $C_1, \ldots, C_n$ s.t. $\forall x(x = C_1 \vee \cdots \vee x = C_n)$. But objects can be unknown, created or destroyed at run-time.
- Can plan when actions and fluents have parameters that vary over infinite domains, since search for a plan is done over the situation tree, and situations serve as concise symbolic proxies for infinite models. State is used only to compute heuristic.
- Can plan using action schemas and instantiates actions at run-time (lifted planning), without building explicit state space in advance. (Augusto Correa works on lifted classical planning)

Towards Planning in the Real World

Problem: How can a computer with its discrete actions achieve goals in the real continuous world where not all objects are known?

Mixed discrete-continuous systems: instantaneous transitions between states due to discrete actions and events, and within each state there is a continuous change. States have *relational* structure: go beyond hybrid automata.

In general, the planning problem is undecidable already for simple hybrid automata. Develop sound algorithms that can compute sometimes plans approximating optimization objectives.

Example of Hybrid Relational Systems: How to integrate Task and Motion Planning (TAMP) in Robotics.

(1) *Theorem proving* (deductive) approach to *lifted* planning, where *search is done over situation tree*, but not over the state space.

(2) Developed an efficient general deductive planner NEAT that uses a new non-trivial domain independent heuristic.

(3) Our NEAT planner demonstrates competitive performance on popular benchmarks from King's College, London, UK (developed in Maria Fox and Derek Long's research group).

Mixed Discrete-Continuous Systems: SC + Timeline

- Planner does search over sequences of actions (situations).
- Situations are convenient concise proxies for states. The state space remains implicit, states are not saved in memory.
- Each action is instantaneous - situations have unique start time.
- Reiter's book: $start(S_0) = 0$ and $start(do(A, S)) = time(A, T)$.

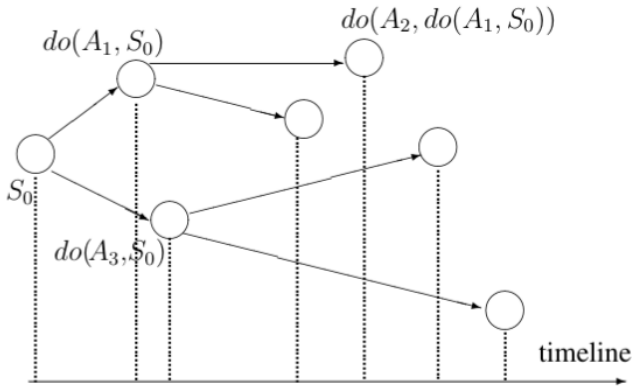

Figure: Situations are aligned with moments on a time line: next situation $do(A, S)$ starts at the moment T when an action A is executed in S.

Hybrid Temporal Situation Calculus (HTSC)

- Our Non-linEAr Temporal (NEAT) planner is based on the recently developed Hybrid Temporal Situation Calculus (HTSC).
- HTSC describes dynamics of a hybrid system in first-order logic in terms of precondition and successor state axioms, + new State Evolution Axioms (joint work with Giuseppe De Giacomo).
- We shall illustrate HTSC using the bouncing ball example.

Precondition Axioms for Actions (**agent** vs **natural**):

$$\forall s \forall t \forall b.\ poss(drop(b,t),s) \leftrightarrow ball(b) \wedge \neg falling(b,s) \wedge \neg flying(b,s) \wedge \underline{t \geq start(s)}.$$

$$\forall s \forall t \forall b.\ poss(bounce(b,t),s) \leftrightarrow ball(b) \wedge falling(b,s) \wedge \underline{distance(b,t,s) = 0} \wedge \underline{velocity(b,t,s) \geq \epsilon} \wedge \underline{t \geq start(s)}.$$

Agent action $catch(b,t)$ and natural action $atPeak(b,t)$ are similar.

Use an *external solver* to deal with numerical constraints. At run-time, add constraints to a store to be (repeatedly) evaluated whenever the planner checks whether the goal logical conditions are satisfied.

Follows Constraint Logic Programming approach (CLP): developed by Alain Colmerauer, Joxan Jaffar, Michael Maher, Peter Stuckey.

Temporal Fluents Change within Situation

To model continuously varying physical quantities, we introduce new functional *temporal fluents* with time as an argument. Values of these fluents change with time within situation.

$$\begin{aligned}(\forall s \forall t \forall b).\ distance(b,t,s) = y \leftrightarrow\ & \exists y_0.\, y_0 = init_{dist}(b,s) \wedge \\ & (\neg falling(b,s) \wedge \neg flying(b,s) \wedge y = y_0) \vee \\ & \big(falling(b,s) \wedge \underline{y = y_0 - \textstyle\int_{start(s)}^{t} (9.81 \cdot x)\, dx}\big) \vee \\ & \big(flying(b,s) \wedge \underline{y = y_0 + \textstyle\int_{start(s)}^{t} (9.81 \cdot x)\, dx}\big).\end{aligned}$$

$$\begin{aligned}(\forall s \forall t \forall b).\ velocity(b,t,s) = y \leftrightarrow\ & \exists y_0.\, y_0 = init_{vel}(b,s) \wedge \\ & (\neg falling(b,s) \wedge \neg flying(b,s) \wedge y = y_0) \vee \\ & \big(falling(b,s) \wedge \underline{y = y_0 + \textstyle\int_{start(s)}^{t} 9.81 dx}\big) \vee \\ & \big(flying(b,s) \wedge \underline{y = y_0 - \textstyle\int_{start(s)}^{t} 9.81 dx}\big).\end{aligned}$$

Collect all (underlined) numerical constraints in a separate data structure. Postpone evaluation until the planner checks if s is a goal.

Assume that the logical goal conditions and the numerical conditions are properly aligned in each application domain, e.g., when the car stops (reached a destination) its velocity must be 0.

Atemporal (usual) Fluents Define Context

There are atemporal fluents that define a context for continuous change.

(1) $falling(B,S)$ means the ball B is falling down and accelerating under the Earth gravity ($9.81 m/s^2$).
(2) The atemporal fluent $flying(B,S)$ means the ball B bounced, it is flying up in situation S and decelerating.
(3) The last context: ball is at rest, i.e., it is neither falling nor flying.

Successor State Axioms for atemporal (usual) fluents:

$$\begin{aligned}(\forall a \forall s \forall b).\ falling(b, do(a,s)) \leftrightarrow\ & \exists t(a = drop(b,t)) \vee \exists t(a = atPeak(b,t)) \vee \\ & falling(b,s)) \wedge (\neg \exists t(a = catch(b,t)) \wedge \neg \exists t(a = bounce(b,t))\end{aligned}$$

$$\begin{aligned}(\forall a \forall s \forall b).\ flying(b, do(a,s)) \leftrightarrow\ & \exists t(a = bounce(b,t)) \vee \\ & flying(b,s)) \wedge \neg \exists t(a = catch(b,t)) \wedge \neg \exists t(a = atPeak(b,t))\end{aligned}$$

In a general case, a context expression is a boolean combination of atemporal logical fluents. Numerical fluents in a context: future work.

There are finitely many (parameterized) contexts which are pairwise mutually exclusive.

Inside each context, temporal fluents change in real physical time.

Methodology: Planning in Logic + External NLP Solver

Objective (metric): reach a goal in *minimal total time* wrt constraints.

Numerical constraints are **not** handled by our planner directly. Collect all the encountered numerical constraints in a data structure.

- A deductive lifted regression-based planner NEAT that does heuristic planning over the situation tree (state space is implicit)
- Delegate Constraints + Objective to external Non-Linear Programming solver (NLP). It computes moments of time when the actions have to be executed. No ad-hoc distretization, the optimal values of physical quantities are determined from NLP.
- Our greedy best first search (GBFS) planner is guided by a new domain-independent heuristic: finds the most promising action by relaxing the flow of underlying continuous processes.
- We did experimental evaluation wrt the state-of-the-art PDDL+ planners. NEAT's performance is comparable or better on most benchmarks except of Linear Generator (lots of symmetry).
- Solving a NLP on each step of planning was not a bottleneck.

Conclusion and Future Work

Our NEAT planner demonstrates performance that is comparable with DiNo, SMTPlan+, ENHSP (the state-of-the-art of PDDL+ planning).

Future Work.

- Consider a broader class of hybrid systems where actions can change logical fluents only, but have no effect on processes.
- Deal correctly with exogenous natural events: this requires reasoning about hypothetical futures.
- Implement more efficient solving of closely related incremental non-linear programming problems by using warm starts.
- Optimize constructed NLP, use GPUs
- Implement A* search (take costs into account) instead of GBFS.
- Prove that under certain conditions our planner is sound.

Acknowledgement: This is an ongoing joint work with Nick Kadovic, Shaun Mathew and Ryan Young.

Invitation: Looking for collaborations, research partners.