PLTL_f and PLDL_f to DFA
+ PDDL Encoding of DFAs

Francesco Fuggitti

Seminar Reasoning Agents
April 23, 2021

SAPIENZA
UNIVERSITÀ DI ROMA

YORK
UNIVERSITÉ
UNIVERSITY

Agenda

Part I

- Pure-past Linear Temporal and Dynamic Logic on finite traces ($PLTL_f$/$PLDL_f$)
- Translation to Deterministic Finite-state Automaton (DFA)
- Transformation to LTL_f/LDL_f and vice versa
- Impact of adopting $PLTL_f$/$PLDL_f$

Part II

- Fully Observable Non-deterministic Planning (FOND) for Temporally Extended Goals
- Planning Domain Definition Language (PDDL)
- Compile a DFA within PDDL

Part I

- Pure-past Linear Temporal and Dynamic Logic on finite traces ($PLTL_f$/$PLDL_f$)
- Translation to Deterministic Finite-state Automaton (DFA)
- Transformation to LTL_f/LDL_f and vice versa
- Impact of adopting $PLTL_f$/$PLDL_f$

Motivation

- Linear-time Temporal and Dynamic Logic studied as compelling formal languages to express temporal specifications
  - *Ease of use* and *intuitiveness*
- The nature of most AI applications is *finite*
  - agents change tasks along the way (vs. computer system that can do the same thing forever)
- Most of work focused on the pure-future LTL_f/LDL_f
- Sometimes specifications are *easier* and *more natural* to express referring to the past ($PLTL_f$ /$PLDL_f$) [Lichtenstein et al. 1985]
  - non-Markovian models [Gabaldon2011]
  - non-Markovian rewards in MDPs [Bacchus et al. 1996]
  - normative properties in multi-agent systems [FisherWooldridge2005;Knobbout et al. 2016;Alechina et al. 2018]
- Computational advantage wrt LTL_f /LDL_f [Chandra et al., 1981]

LTL_f and LDL_f (pure-future)

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\mathcal{U}\varphi$$

- Same syntax of LTL

$\bigcirc\varphi$: φ holds at the *next* step

$\varphi_1\mathcal{U}\varphi_2$: φ_1 holds *until* φ_2 does

$\Diamond\varphi \doteq \top\mathcal{U}\varphi$

$\Box\varphi \doteq \neg\Diamond\neg\varphi$

Example: "for every request you send, you will eventually receive a reply"

$$\Box(request \supset \Diamond reply)$$

$$\begin{array}{lcl} \varphi & ::= & tt \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\varrho\rangle\varphi \\ \varrho & ::= & \phi \mid \varphi? \mid \varrho + \varrho \mid \varrho;\varrho \mid \varrho^* \end{array}$$

- Merging LTL_f and regular expressions

$\langle\varrho\rangle\varphi$: there exists an "execution" of ϱ that ends with φ holding

$[\varrho]\varphi \doteq \neg\langle\varrho\rangle\neg\varphi$: all "executions" of ϱ end with φ holding

Example: "alternating occurrence until the end of the trace"

$$\langle(\psi;\varphi)^*;(\psi;\varphi)\rangle end$$

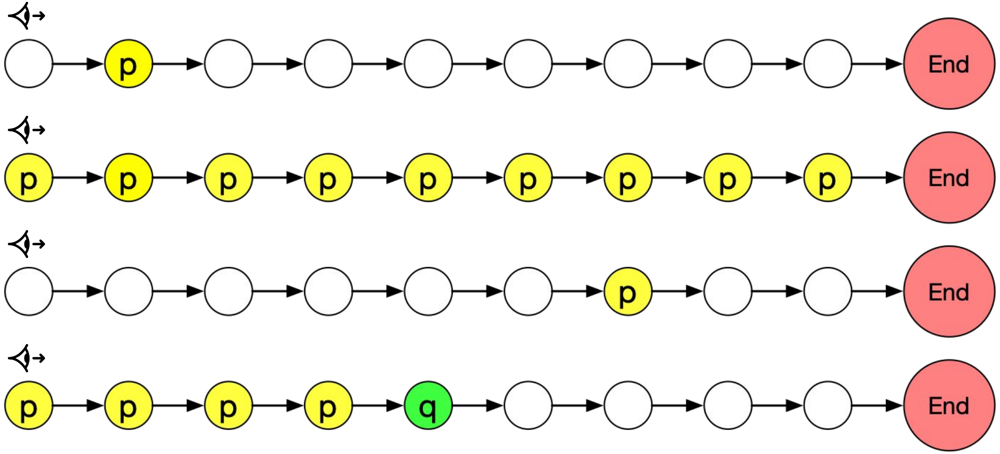
LTL_f examples

$\mathcal{X} p$ NEXT TIME

$\Box p$ ALWAYS

$\Diamond p$ EVENTUALLY

$p \mathcal{U} q$ UNTIL



PLTL$_f$ syntax

PLTL$_f$ is the *pure-past* version of LTL$_f$

- finite set of atomic propositions
- Boolean connectives $\neg, \wedge, \vee,$ and $\rightarrow$
- past temporal operators

$$\varphi \quad ::= \quad a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \ominus\varphi \mid \varphi\mathcal{S}\varphi$$

$\ominus\varphi$	previous, yesterday, before
$\varphi_1\mathcal{S}\varphi_2$	since
$\diamondminus\varphi \doteq \top\mathcal{S}\varphi$	once
$\boxminus\varphi \doteq \neg \diamondminus \neg\varphi$	historically

PLTL$_f$ semantics

PLTL$_f$ formulas are satisfied if they hold in the last instant of the trace

$\ominus\varphi$: φ held in the *previous* time-step

$\varphi_1 \mathcal{S} \varphi_2$: φ_2 held once, and *since* that time φ_1 holds

$\diamondminus\varphi \doteq \top \mathcal{S} \varphi$: φ held *once* in the past

$\boxminus\varphi \doteq \neg \diamondminus \neg\varphi$: φ held always in the past

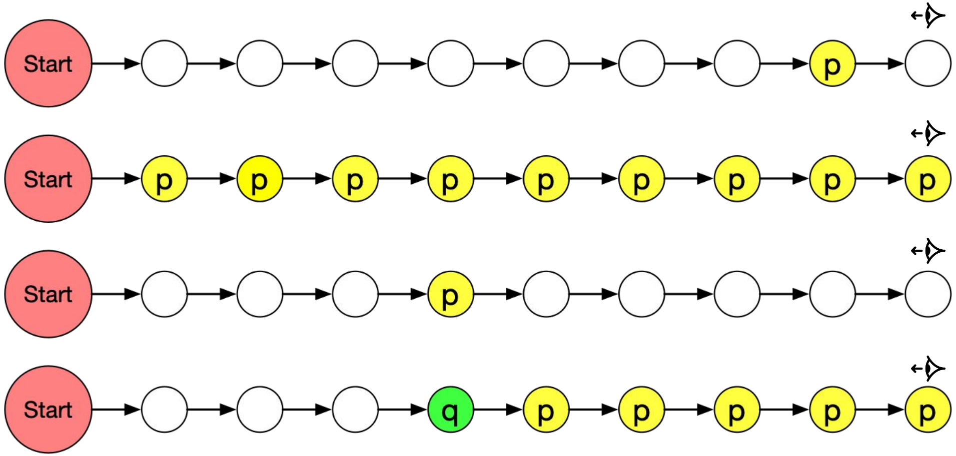
PLTL$_f$ examples

$\ominus p$ PREVIOUS TIME

$\boxminus p$ HISTORICALLY

$\diamondminus p$ ONCE

$p \mathcal{S} q$ SINCE



More examples

- $a\,\mathcal{S}\,b \equiv \Diamond(b \wedge (\neg last \supset \circ(\Box a))$, where $last \doteq \neg\circ true$
- $\ominus a \equiv \Diamond(a \wedge \circ last)$
- $a \wedge \diamondminus b \equiv \Diamond(b \wedge \Diamond(a \wedge last))$
- $\boxminus(a \supset \diamondminus b) \equiv \neg((\neg b)\,\mathcal{U}(a \wedge \neg b))$
- $\boxminus(a \supset \ominus(\diamondminus(b \wedge \neg a))) \equiv \boxminus(a \supset \diamondminus(b \wedge \neg a)) \equiv \neg((\neg b)\,\mathcal{U}\,a)$
- $task \wedge (\neg inArea\,\mathcal{S}\,clean) \equiv \Diamond(clean \wedge (\neg(task \wedge last) \supset \circ(\neg inArea\,\mathcal{U}(\neg inArea \wedge task \wedge last))))$
- $start \supset (batt\,\mathcal{S}\,charge) \equiv \Diamond(charge \wedge (\neg last \supset \circ(batt\,\mathcal{U}(batt \wedge last)))) \vee \Diamond(\neg start \wedge last)$
- $start \wedge \neg(batt\,\mathcal{S}\,charge) \equiv \Box(\neg charge \vee (\neg last \wedge \neg\circ(\Box batt))) \wedge \Diamond(start \wedge last)$
- $\boxminus(a \supset \ominus(\neg a\,\mathcal{S}\,b)) \equiv (b\,\mathcal{R}\,a) \wedge \Box(a \supset (b \vee \circ(b\,\mathcal{R}\,\neg a)))$

PLDL_f syntax

PLDL_f is the *pure-past* version of LDL_f

- Boolean connectives
- regular expressions on atomic propositions (including *true* and *false*)
- past dynamic operators

$$
\begin{array}{lcl}
\varphi & ::= & tt \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle \varrho \rangle\!\rangle \varphi \\
\varrho & ::= & \phi \mid \varphi? \mid \varrho + \varrho \mid \varrho ; \varrho \mid \varrho^*
\end{array}
$$

$\langle\!\langle \varrho \rangle\!\rangle \varphi$ backward diamond

$[\![\varrho]\!] \varphi \doteq \neg \langle\!\langle \varrho \rangle\!\rangle \neg \varphi$ backward box

PLDL_f semantics

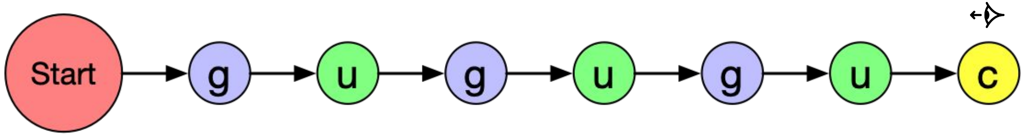
PLDL_f formulas are satisfied if they hold in the last instant of the trace

$\langle\!\langle \varrho \rangle\!\rangle \varphi$: there exists an "execution" of ϱ (going backwards) that ends with φ holding

$[\![\varrho]\!] \varphi \doteq \neg \langle\!\langle \varrho \rangle\!\rangle \neg \varphi$: all "executions" of ϱ (going backwards) end with φ holding

Example: "every time, if the cargo-ship departed (*cs*), then there was an alternation of *grab* and unload (*unl*) of containers before"

$$[\![true^*]\!] \big(\langle\!\langle cs \rangle\!\rangle tt \supset \langle\!\langle (unl; grab)^*; (unl; grab) \rangle\!\rangle start \big)$$



Key property

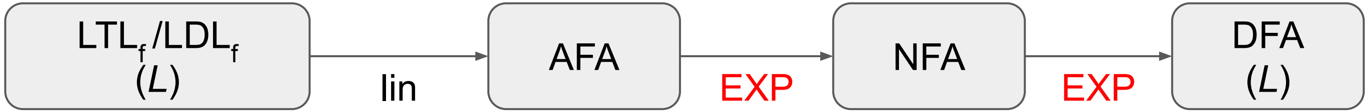
LTL_f /LDL_f /PLTL_f /PLDL_f formulas can be translated into *deterministic finite-state automata* (DFA)

$$\tau \models \varphi \text{ iff } \tau \in \mathcal{L}(A_\varphi)$$

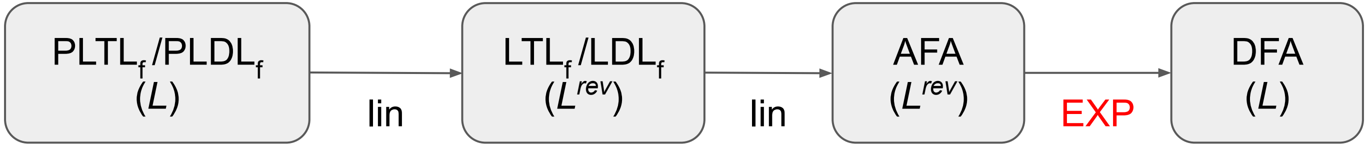
where A_φ is the DFA associated to φ

From φ to Automata

- LTL_f/LDL_f:



- $PLTL_f$/$PLDL_f$:



- Given an AFA of k states for language L, there exists a DFA of at most 2^k states for language $L^{reverse}$ [Chandra et al. 1981]

Swap formulas

$$\text{PLTL}_f/\text{PLDL}_f\ (L) \xrightarrow{\text{lin}} \text{LTL}_f/\text{LDL}_f\ (L^{rev}) \longrightarrow$$

$$\mathcal{L}^{rev} = \{\tau^{rev} : \tau \in \mathcal{L}\}$$

Swapping: syntactic replacement of each past (future) temporal/dynamic operator with the corresponding future (past) one.

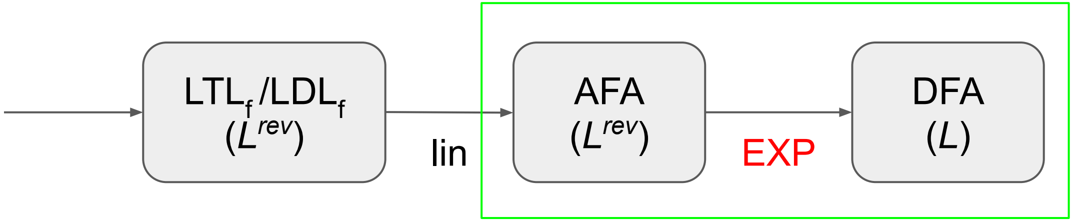
Example:

$$inRoom \land roomDecontaminated \land \diamondminus(getPermit)$$

$$\downarrow$$

$$inRoom \land roomDecontaminated \land \Diamond(getPermit)$$

From AFA to DFA



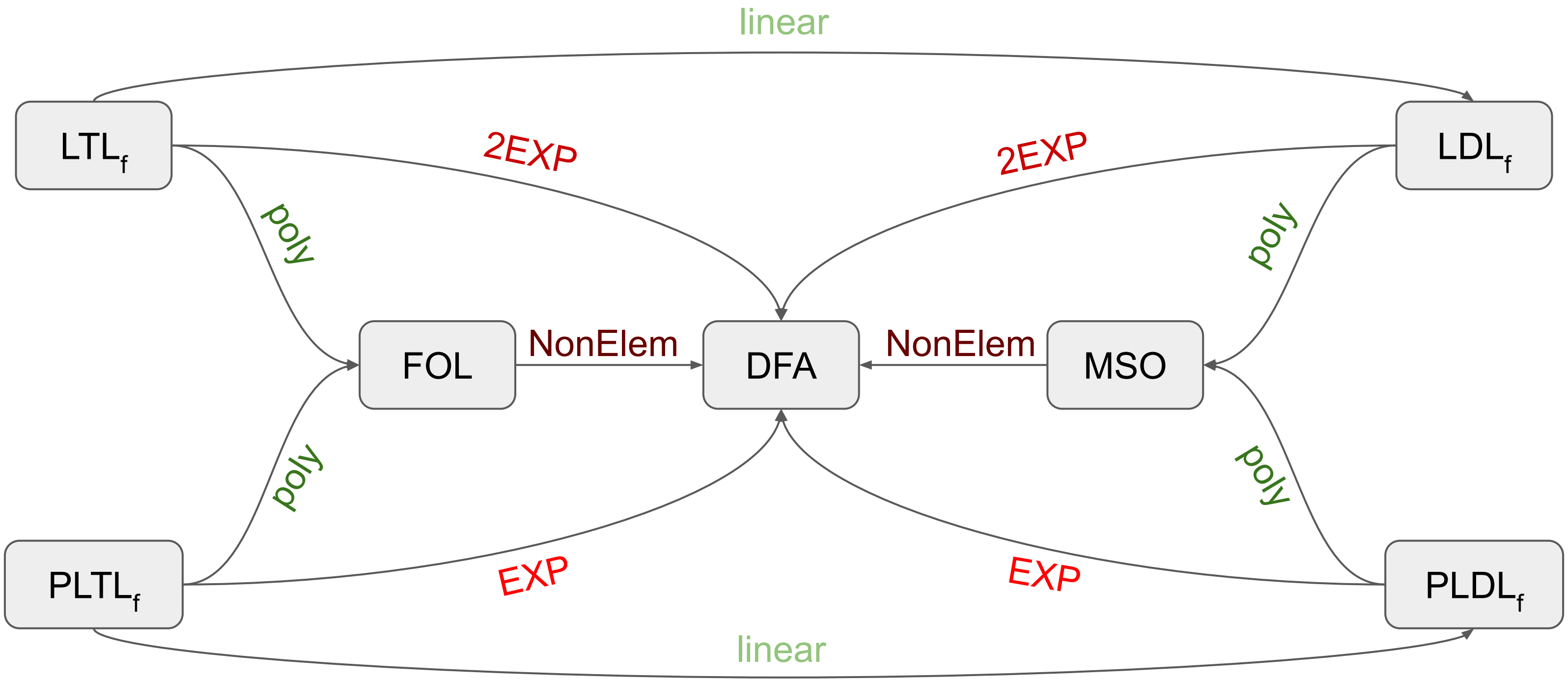
AFA $A^{rev} = (\Sigma, Q, q_0, \delta, F)$

DFA $A = (\Sigma, S, s_{init}, T, F')$

- $S = 2^Q, s_{init} = F$
- $T(V, a)$ is the set of all q such that $V \models \delta(q, a)$, for $v \in S, a \in \Sigma$
- $V \in F'$ iff $q_0 \in V$

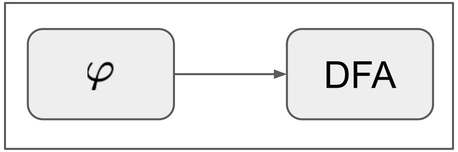
Intuitively, we move from a past view of the trace to a future one.

Transformations

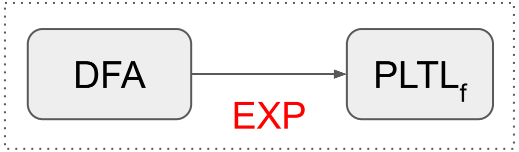


Useful Results

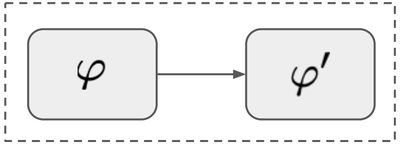
1. Transformations seen before



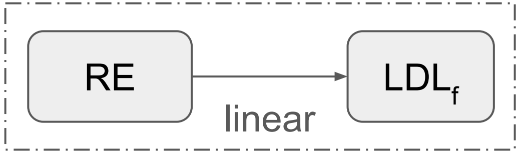
2. From DFA (star-free) to $PLTL_f$ [Maler&Pnueli 1990]



3. Syntactic swap of temporal operators

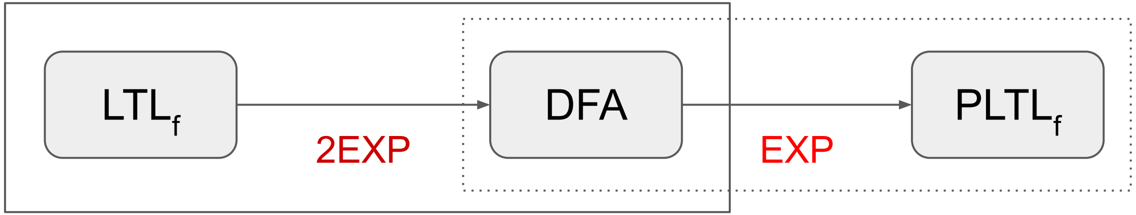


4. From RE to LDL_f [DeGiacomo&Vardi 2013]

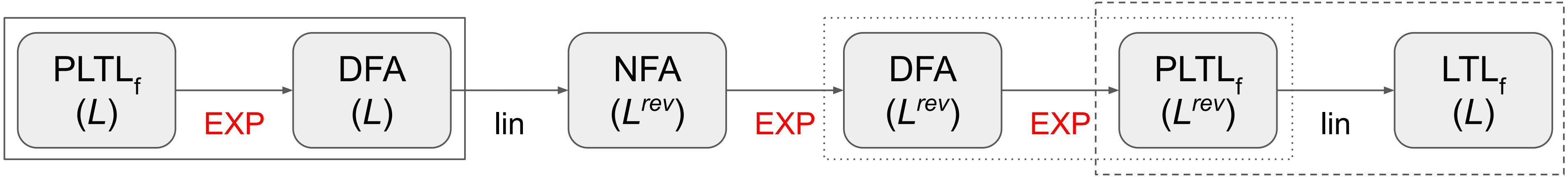


LTL_f to $PLTL_f$ (and vice versa)

- From LTL_f to $PLTL_f$

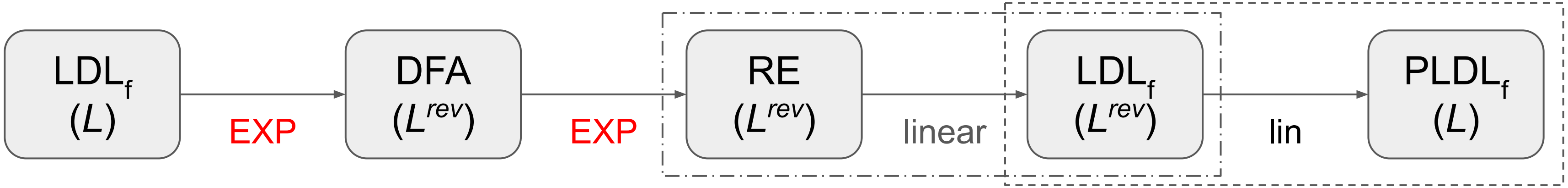


- From $PLTL_f$ to LTL_f

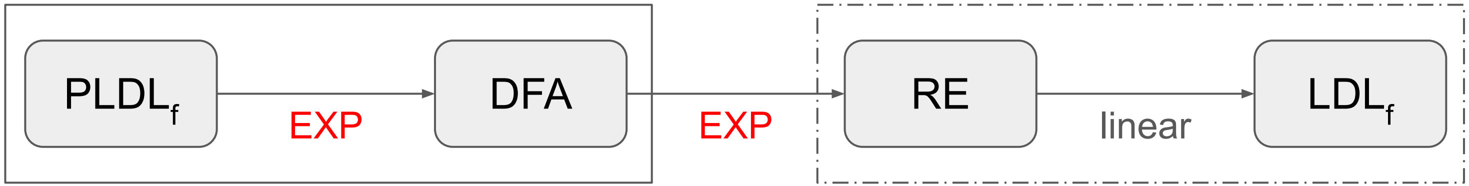


LDL_f to $PLDL_f$ (and vice versa)

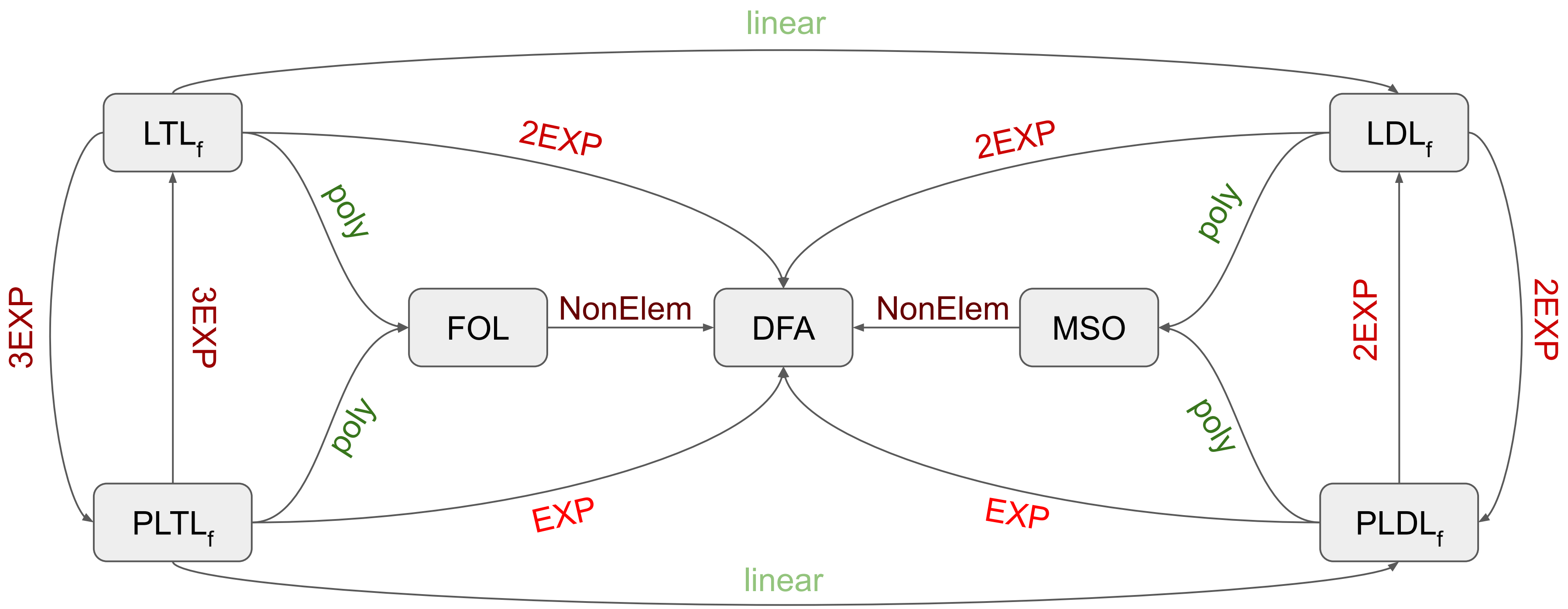
- From LDL_f to $PLDL_f$



- From $PLDL_f$ to LDL_f



Transformations (ii)



Impact of adopting $PLTL_f$ /$PLDL_f$

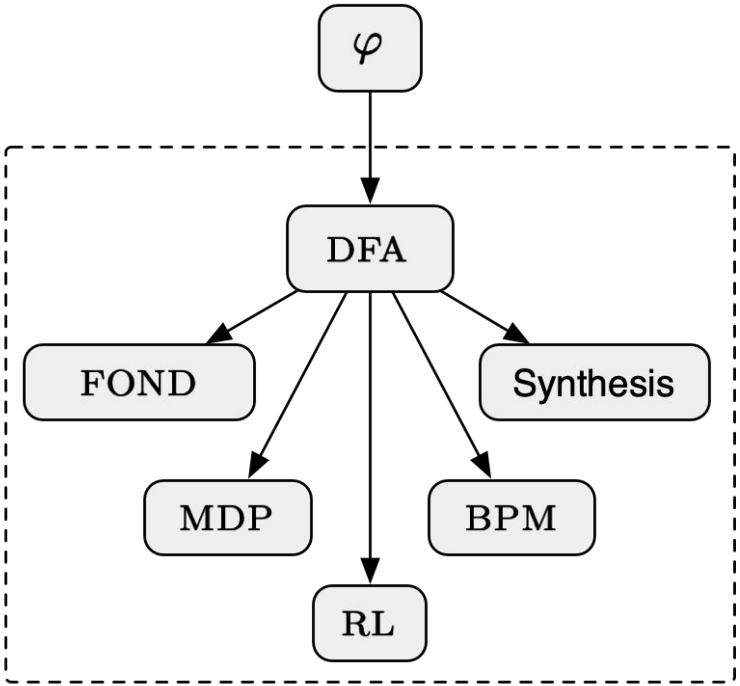
- Exponential gain reflected in an exponential gain in solving different forms of sequential decision making involving temporal specifications:
  - FOND Planning for temporally extended goals
  - MDPs with non-Markovian rewards
  - Reinforcement Learning with temporally extended rewards
  - Planning in non-Markovian domains
  - non-Markovian decision processes

- Problems are EXPTIME-complete in the goal/reward natively expressed in $PLTL_f$ /$PLDL_f$ (vs. 2EXPTIME-complete in the LTL_f /LDL_f goal/reward)

Takeaways

- If you can *naturally* express the specification in $PLTL_f$ /$PLDL_f$, then do it to get the computational advantage.
- Converting LTL_f /LDL_f into $PLTL_f$ /$PLDL_f$ to get the exponential advantage is not computationally sensible
- Complexities are just worst-case, in many applications the size of the resulting DFA is actually manageable

How can we use DFAs?



Part II

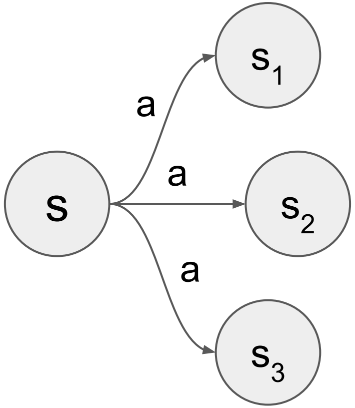
- Fully Observable Non-deterministic (FOND) Planning for Temporally Extended Goals
- Planning Domain Definition Language (PDDL)
- Compile a DFA within PDDL

FOND planning

Nondeterministic domain (including initial state)

$\mathcal{D} = (2^{\mathcal{F}}, \mathcal{A}, s_0, \delta, \alpha)$ where:

- $\mathcal{F}$ **fluents** (atomic propositions)
- $\mathcal{A}$ **actions** (atomic symbols)
- $2^{\mathcal{F}}$ set of states
- s_0 initial state (initial assignment to fluents)
- $\alpha(s) \subseteq \mathcal{A}$ represents **action preconditions**
- $\delta(s, a, s')$ with $a \in \alpha(s)$ represents **action effects (including frame)**.



Who controls what?

Fluents controlled by **environment**

Actions controlled by **agent**

Observe: $\delta(s, a, s')$

Goals, planning, and plans

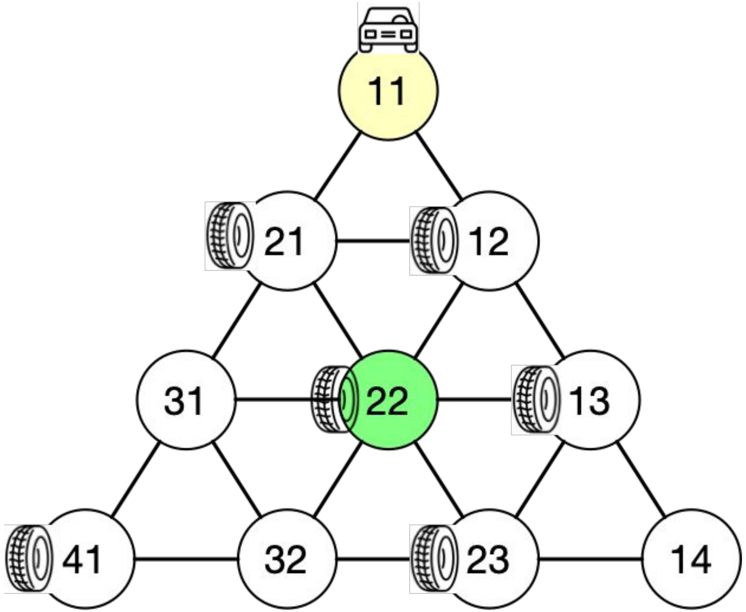
Goal = propositional formula G on fluents
Planning = **game** between two players:

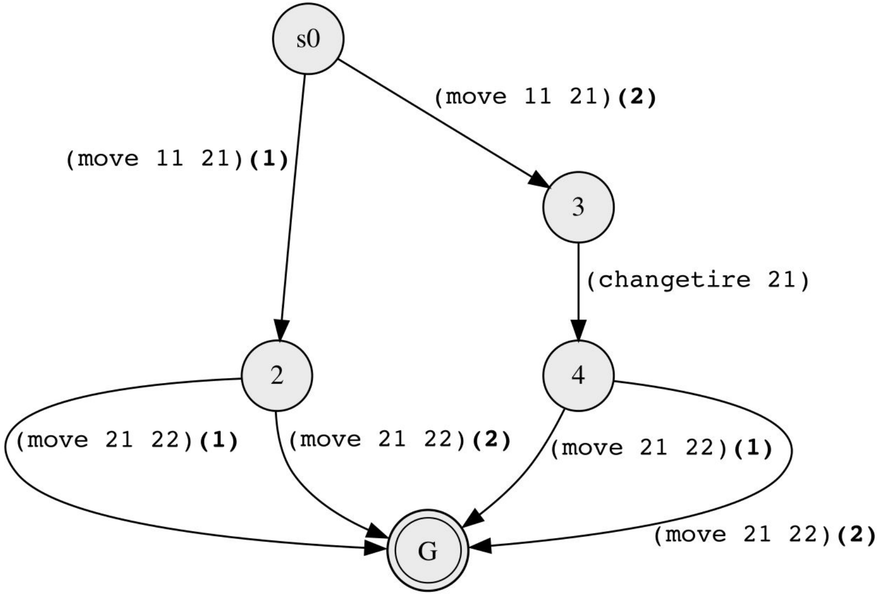
> **agent** *tries to force eventually reaching* G *no matter how other* **environment** *behave.*

Plan = **strategy** to **win** the game.

(FOND_{sp} is EXPTIME-complete)

Running example - TriangleTireworld



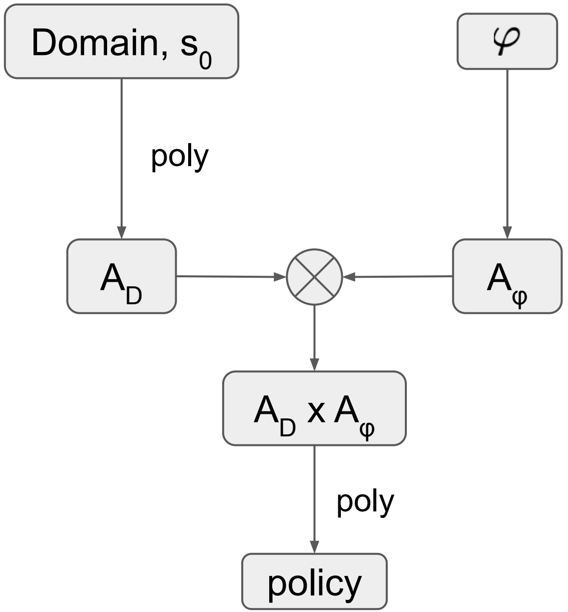


Why use temporally extended goals?

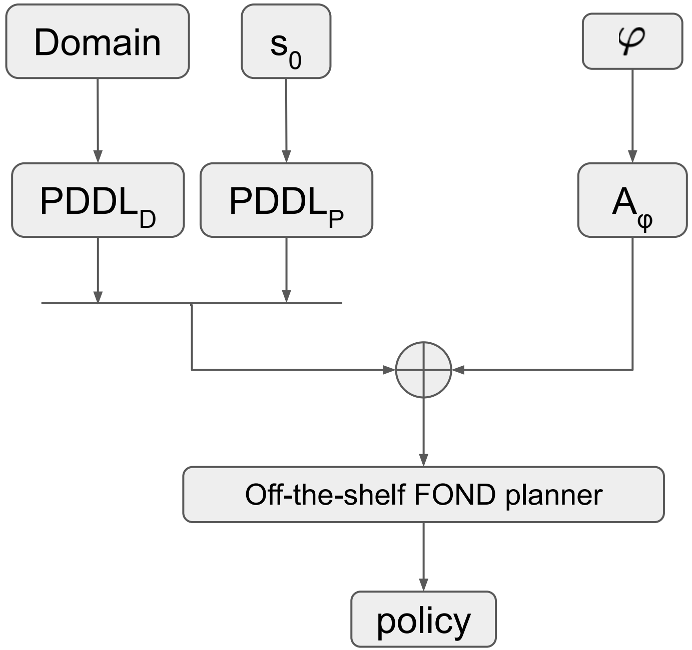
- Idea:
  - Capture a richer class of plans (more general specifications)
  - Restrict the way the planner achieves the goal
- This problem has a long history in the AI Planning community
  - Deterministic planning [BacchusKabanza98;DeGiacomoVardi99;DohertyKvarnstram01;BaierMcIlraith06;...]
  - Non-deterministic planning [Patrizietal13;Camachoetal17,18;DeGiacomoRubin18;..]
- We can use any LTL_f /LDL_f /$PLTL_f$ /$PLDL_f$ goals

Solutions to FOND for TEGs

Automata-theoretic Techniques



Automata Encoding in PDDL



Recap on PDDL (i)

PDDL = Planning Domain Definition Language

PDDL is the de-facto standard for the specification of planning tasks

Main components of a PDDL planning tasks:

- **Objects:** Things in the world that interest us
- **Predicates:** Properties of objects that we are interested in; can be *true* or *false*.
- **Initial state:** The state of the world that we start in.
- **Goal specification:** Things that we want to be true. (classical setting)
- **Actions/Operators:** Ways of changing the state of the world.

Recap on PDDL (ii)

Planning tasks specified in PDDL are separated into two files:

1. A **domain file** for predicates and actions.
2. A **problem file** for objects, initial state and goal specification.

Note:

- Generally, PDDL domains are *independent* from PDDL problems. We can have several problems for a specific domain.
- PDDL domains are parametric. They are instantiated/grounded (becoming propositional) at planning time.

Running example PDDL domain

```
(define (domain triangle-tire)
  (:requirements :typing :strips :non-deterministic)
  (:types location)
  (:predicates
    (vehicleat ?loc - location)
    (spare-in ?loc - location)
    (road ?from - location ?to - location)
    (not-flattire)
  )

  (:action move-car
    :parameters (?from - location ?to - location)
    :precondition (and (vehicleat ?from) (road ?from ?to) (not-flattire))
    :effect (and
            (oneof
                (and (vehicleat ?to) (not (vehicleat ?from)))
                (and (vehicleat ?to) (not (vehicleat ?from))
                     (not (not-flattire)))
            )
        ))

  (:action changetire
    :parameters (?loc - location)
    :precondition (and (spare-in ?loc) (vehicleat ?loc))
    :effect (and (not (spare-in ?loc)) (not-flattire)))
```

Running example PDDL problem

```
(define (problem triangletire)
  (:domain triangletire)
  (:objects 11 12 13 14
            21 22 23
            31 32
            41 - location)
  (:init    (vehicleat 11)
            (road 11 12)(road 12 13)(road 13 14)
            (road 11 21)(road 12 22)
            (road 13 23)(road 21 12) ...

            (spare-in 12)(spare-in 13)
            (spare-in 21)(spare-in 22)(spare-in 23)
            (spare-in 41)

            (not-flattire))

  (:goal (vehicleat 14)))
```

PDDL: a note on action effects

Action effects can be more complicated than what seen so far

They can be **conditional:**

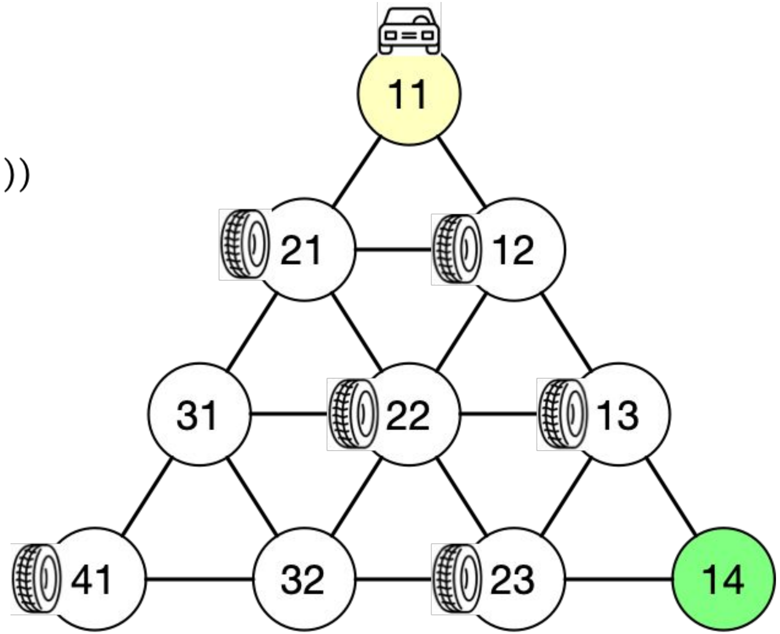
```
(when <condition>
      <effect>)
```

They can be **universally quantified**:

```
(forall (?v1 ... ?vn)
        <effect>)
```

Possible TEGs for our example

- $\Diamond(\mathsf{vehicleat}(14))$
- $\Diamond(\mathsf{vehicleat}(12) \land \mathcal{X}(\Diamond(\mathsf{vehicleat}(13) \land \mathcal{X}(\Diamond(\mathsf{vehicleat}(14))))))$
- $\mathsf{vehicleat}(14) \land \diamondminus(\mathsf{vehicleat}(22))$
- $\mathsf{vehicleat}(14) \land \ominus(\mathsf{vehicleat}(23))$
- $\mathsf{vehicleat}(14) \land (\neg\mathsf{vehicleat}(13)\ \mathcal{S}\ \mathsf{vehicleat}(12))$
- ...



A simple solution

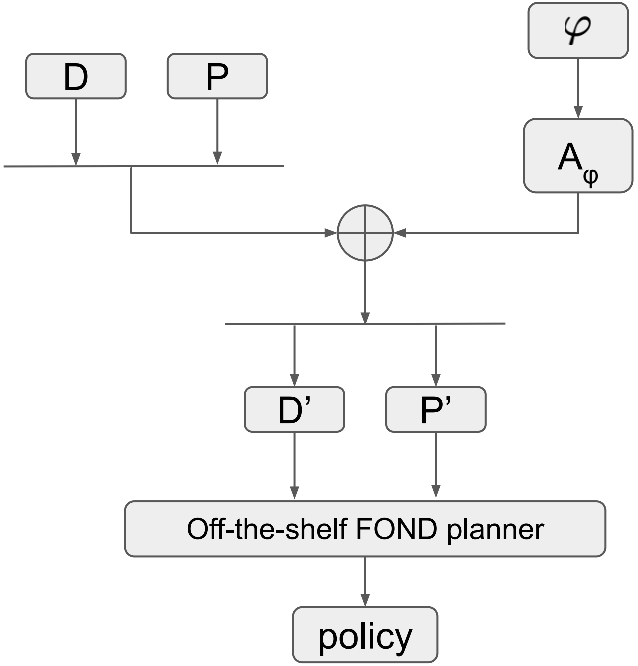
Steps:

1. Build the *parametric* DFA of A_φ (PDFA)
2. Encode dynamics of the PDFA in PDDL
3. Generate <D', P'>

This solution has been implemented:

- whitemech/fond4ltlfpltlf
- http://fond4ltlfpltl.diag.uniroma1.it/

Automata Encoding in PDDL



1. Build the *parametric* DFA

Why? In our DFA, propositions are represented by domain fluents grounded on specific objects of interest, but in the PDDL domain this is not the case! So, we replace propositions with objects variables

How will the policy "talk" about our specific objects? We use a mapping function m^{obj} that maps objects variables into the problem instance (i.e., in P')

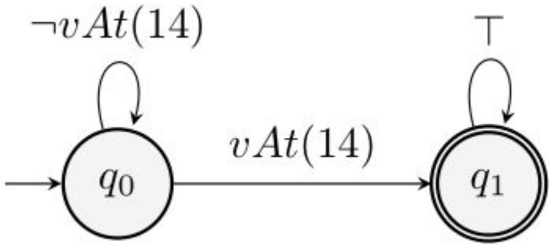
DEFINITION 6. *A parametric DFA (PDFA) is a tuple* $\mathcal{A}^p_\varphi = \langle \Sigma^p, Q^p, q^p_0, \delta^p, F^p \rangle$, *where:*

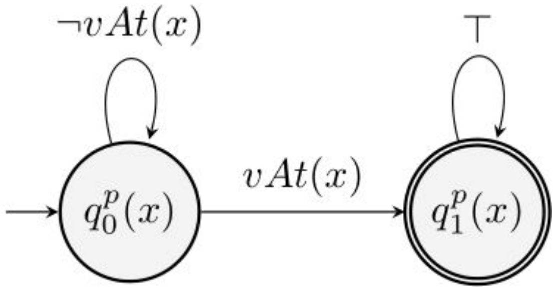
- $\Sigma^p = \{\sigma^p_0, \ldots, \sigma^p_n\} = 2^{\mathcal{F}}$ *is the alphabet of planning domain fluents;*
- Q^p *is a nonempty set of parametric states;*
- q^p_0 *is the parametric initial state;*
- $\delta^p : Q^p \times \Sigma^p \to Q^p$ *is the parametric transition function;*
- $F^p \subseteq Q^p$ *is the set of parametric final states.*

$\Sigma^p, Q^p, q^p_0, \delta^p$ *and* F^p *can be obtained by applying* m^{obj} *to all the components of the corresponding DFA.*

1. Build the *parametric* DFA (example)

Our LTL_f goal formula: $\Diamond\, \mathsf{vehicleat}(14)$





2. Dynamics of PDFA in PDDL

New Fluents of D': $F \cup \{q \mid q \in Q^p\} \cup \{\texttt{turnDomain}\}$

Modified Agent Actions in D': for every $a \in A$

$Pre'_a = Pre_a \cup \{\texttt{turnDomain}\}$

$Eff'_a = Eff_a \cup \{\texttt{(not (turnDomain))}\}$

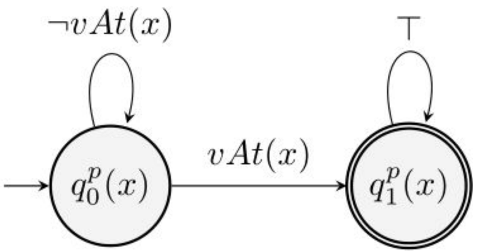
New PDFA transition function in D':

$Pre_{\texttt{transition}} = \{\texttt{(not (turnDomain))}\}$

$Eff_{\texttt{transition}} = \{\texttt{turnDomain}\} \cup \{\texttt{when } (q^p, \sigma^p), \texttt{then } \delta^p(q^p, \sigma^p) \cup \{\neg q \mid q \neq q^p, q \in Q^p\}\}$, for all $(q^p, \sigma^p) \in \delta^p$.

2. Dynamics of PDFA in PDDL (example)

```
(:action transition
  :parameters (?x - location)
  :precondition (not (turnDomain))
  :effect (and (when (and (q0 ?x) (not (vAt ?x)))
                     (and (q0 ?x) (not (q1 ?x)) (turnDomain))
                 (when (or (and (q0 ?x) (vAt ?x)) (q1 ?x))
                     (and (q1 ?x) (not (q0 ?x)) (turnDomain))
          )
)
```



3. Generate <D', P'>

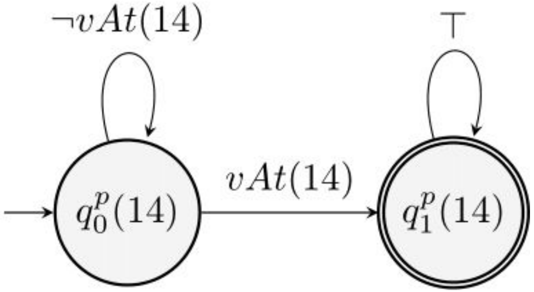
New Initial Condition in P': $s_0' = s_0 \cup \{q_0^p\} \cup \{\texttt{turnDomain}\}$

New Goal Condition in P': $G' = \{\bigvee q_i \mid q_i \in F^p\} \cup \{\texttt{turnDomain}\}$

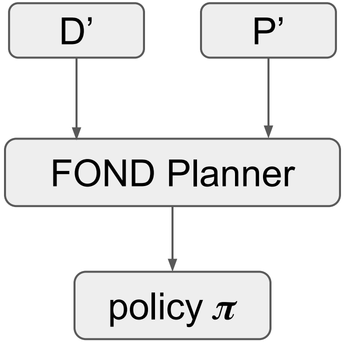
In our running example:

```
(:init (and (road 11 21) (road 11 21) ...
            (spare-in 21) (spare-in 12) ...
            (q0 14)
            (turnDomain)
       )
)
```

```
(:goal
    (and (q1 14) (turnDomain))
)
```



Planners and policy



- FOND-SAT
- MyND
- PRP

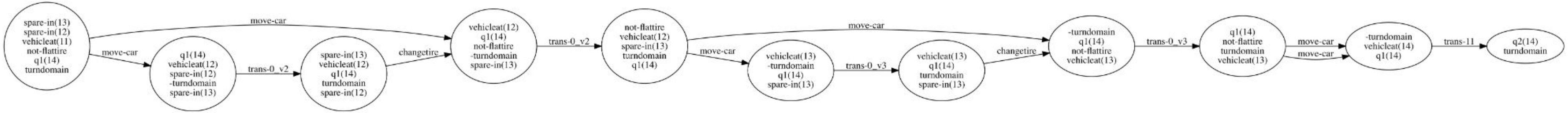
Given the policy, its executions will be of the form

$$e_i^\pi : [a_1, t_1, a_2, t_2, \ldots, a_n, t_n]$$

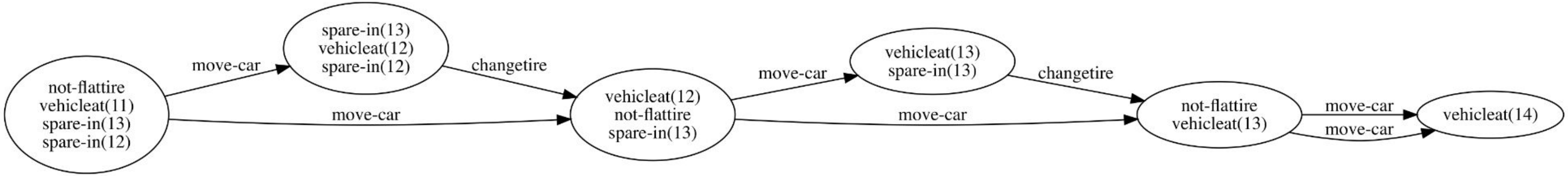
where $a_1 \ldots a_n$ *are agent actions* and $t_1 \ldots t_n$ are synchronization “transition” actions, which, at the end, can be easily removed to extract the desired execution (plan).

Results for our running example

Policy with “transitions” of the DFA



Final policy



Main Limitations

- This encoding is not very efficient
  - FOND Planning for TEGs is EXPTIME-complete in the size of the domain (*number of fluents*), 2EXPTIME-complete in the size of the LTL_f /$PLTL_f$ goal
  - Each DFA state is a new fluent in the domain, and DFAs can be very large!!!

Planner side:

- State-of-the-art FOND planners do not support conditional effects (and other PDDL features)
- Performances, especially for SAT-based planners

Project Ideas

- Proof-of-concept implementation of $PLTL_f$/$PLDL_f$ to DFA
- Provide and test other DFA encodings in PDDL
- Benchmark planners against different PDDL encodings and/or temporally extended formulas