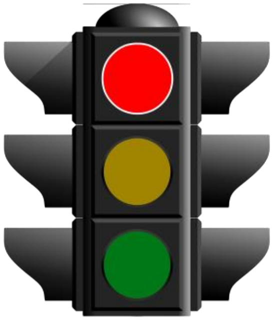
Planning with Temporal Logic

April 25, 2016

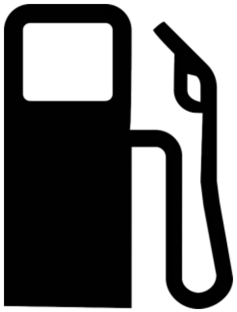
Motivation

- Consider a self-driving car…
- Regardless of our destination, we also want to make sure we always follow the rules of the road.

Motivation



Motivation



Key Takeaways

- Modeling temporally-extended goals with linear temporal logic (LTL)
- Modeling preferences between alternative plans

Outline

- **Introduction to Linear Temporal Logic**
  - Why use Linear Temporal Logic?
  - Linear Temporal Logic Operators
  - Example LTL Problems
- **Applications to Planning**
- **Planning with Preferences**
  - Expressing Preferences
  - Planning in LPP

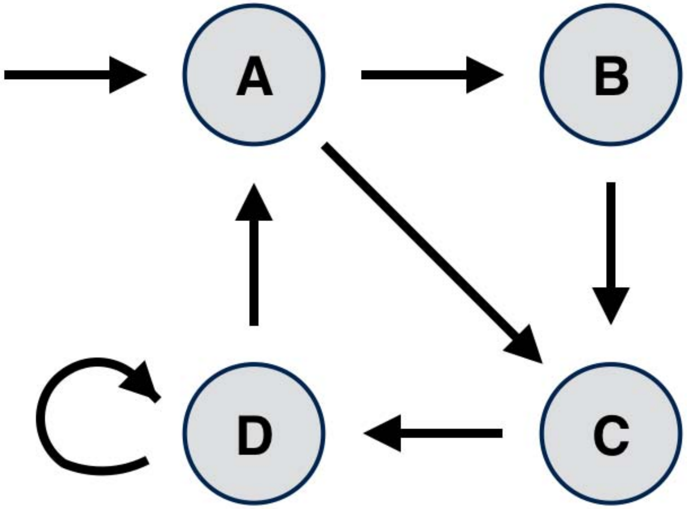
Linear Temporal Logic

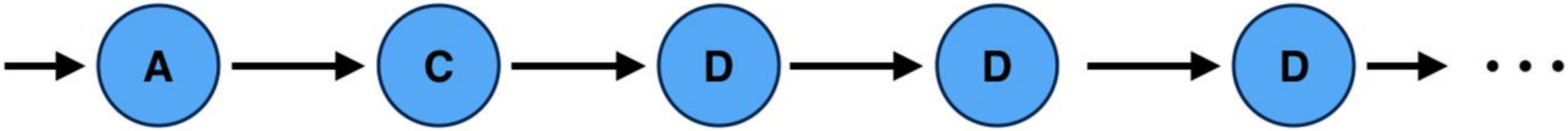
Temporal Logic

- Formalism for specifying properties of systems that vary with time

Temporal Logic

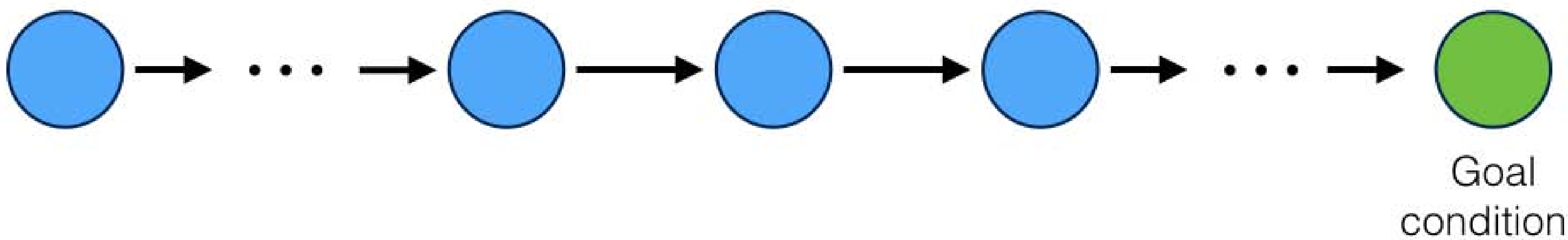
- Systems proceed through a sequence of discrete states



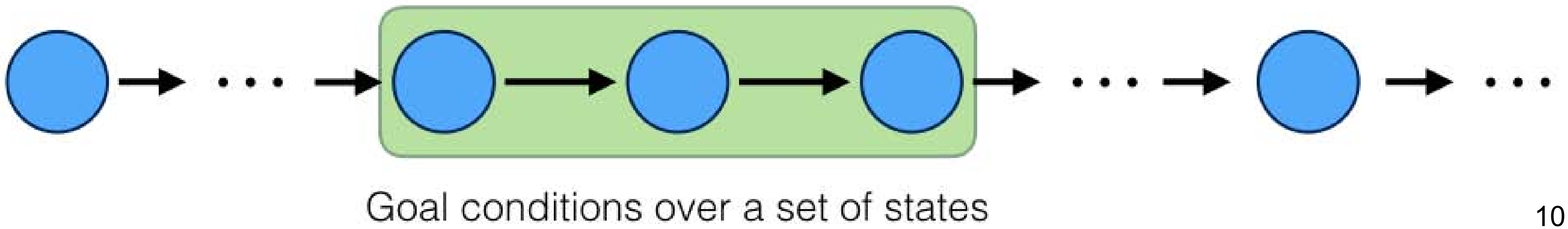


Why Temporal Logic?

- Previously our planning algorithms have used propositional logic to specify goals dealing with a **single state** at a single point in time

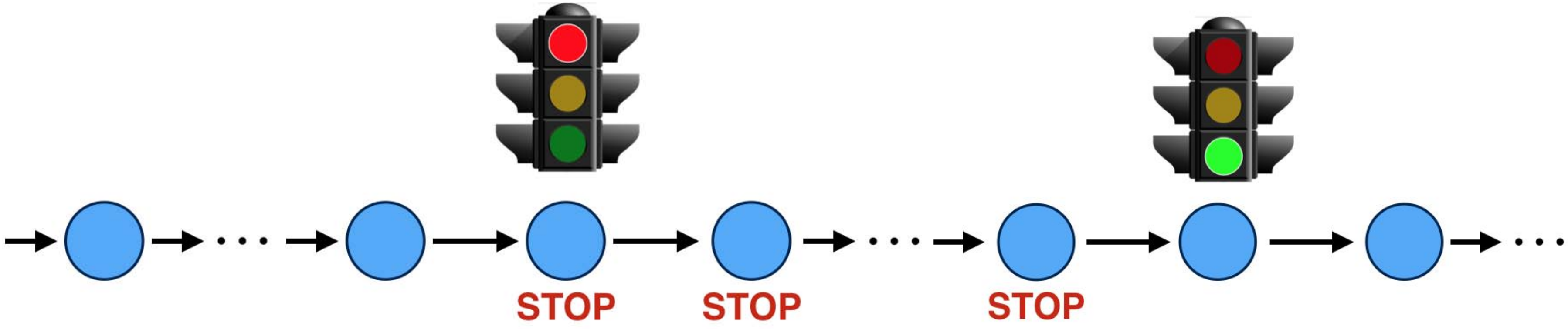


- Temporal logic allows these goals to be specified over a sequence of states



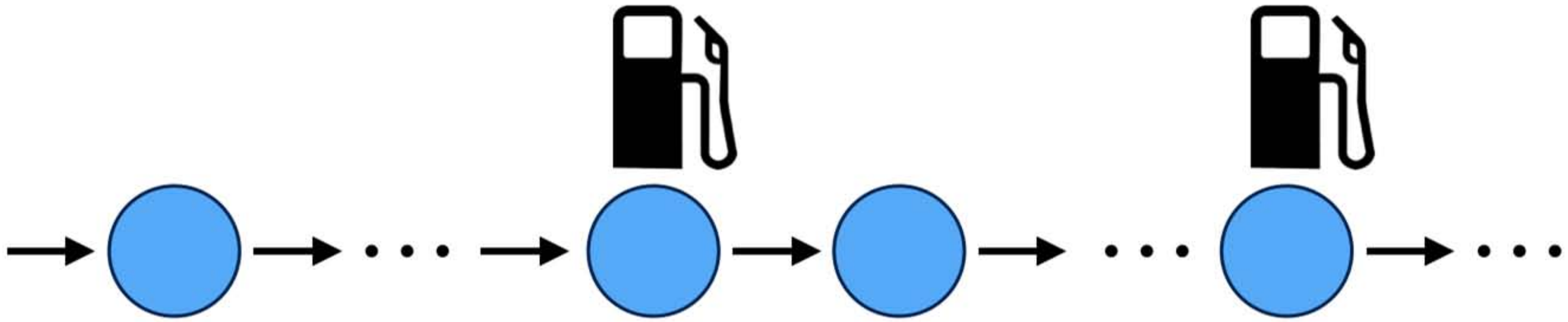
Why Temporal Logic?

- What if the problem requires a condition to:
  - –Be met until another condition is met...
    - •For example: **red** implies (**stop** until **green**)



Why Temporal Logic?

- What if the problem requires a condition to:
  - Always eventually be met
    - For example, always have some point in the future when you visit a gas station



Temporal Logic

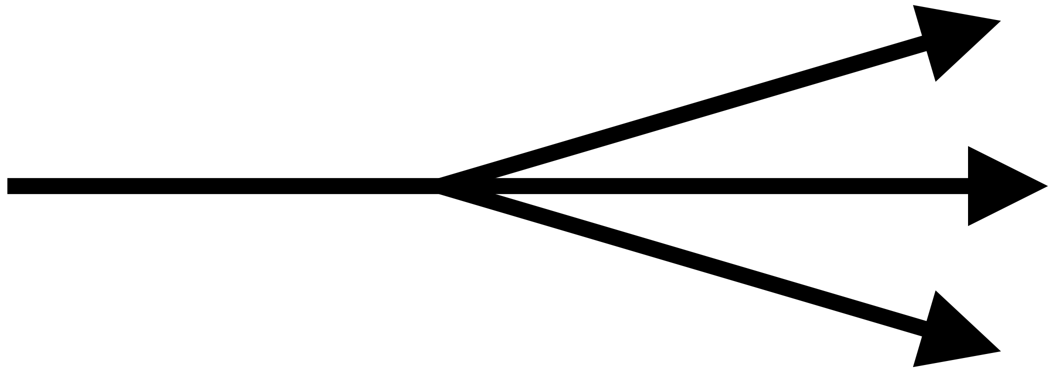
Branching vs linear time

- Linear time
  - Models physical time
  - At each time instant, <u>only one</u> of the future behaviors is considered
  - We can reason about **always**

Temporal Logic

Branching vs linear time

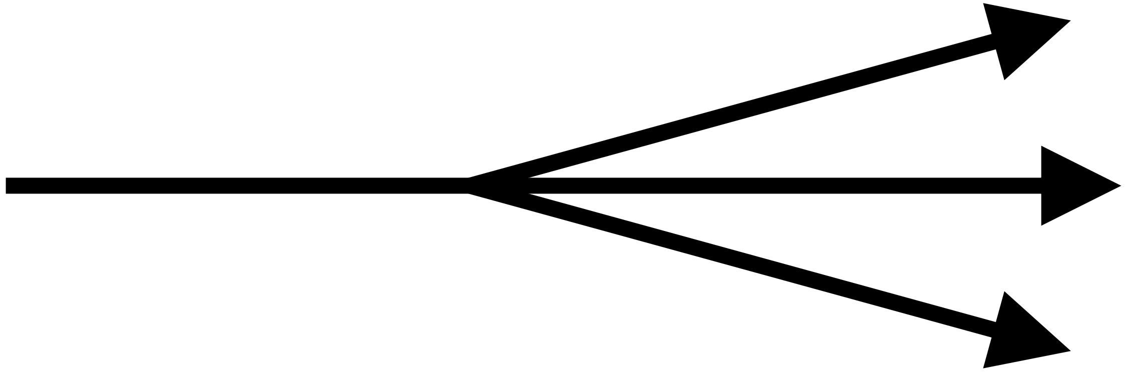
- Branching time
  - At each time instant, all possible future behaviors are considered
  - Time may split into alternate courses
  - We can reason about **possibilities**



Branching vs linear time

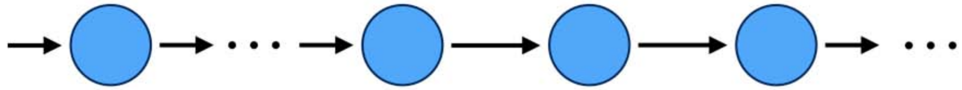
- Linear time

- Branching time



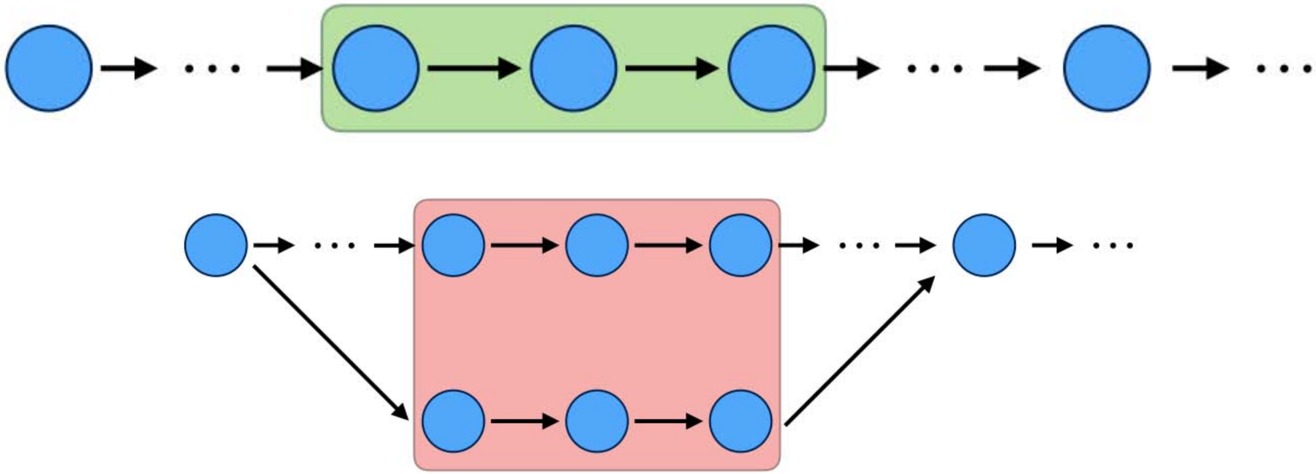
Linear Temporal Logic

- Linear Temporal Logic (LTL) involves:
  - Linear time model
  - Infinite sequences of states



  - Forward-looking conditions

- Cannot express properties over a set of different paths



Applications of Temporal Logic

- Temporal logic is used in:
  - Verification and Model Checking
    - Safety and Maintenance
  - **Planning**

LTL Syntax

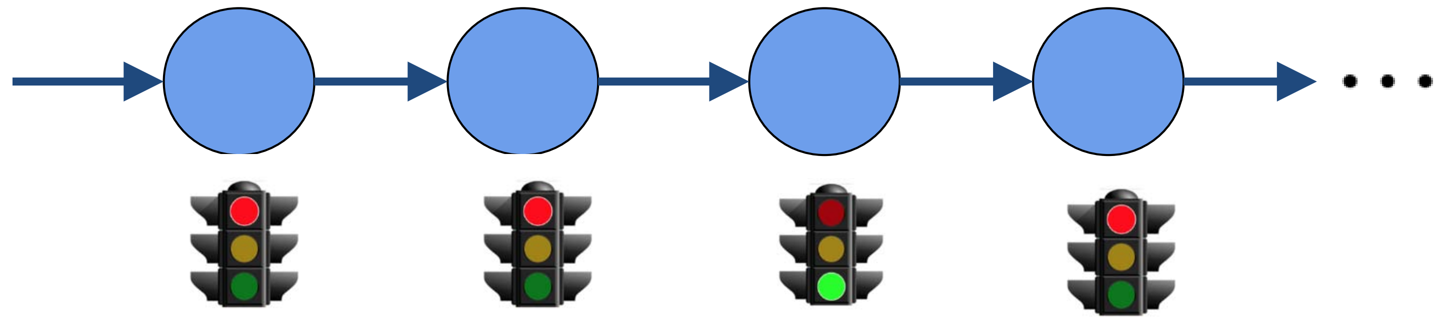
LTL formula f := true | p_i | $f_i \wedge f_j$ | $\neg f_i$ | X f_i | f_i U f_j

An LTL formula is built from:

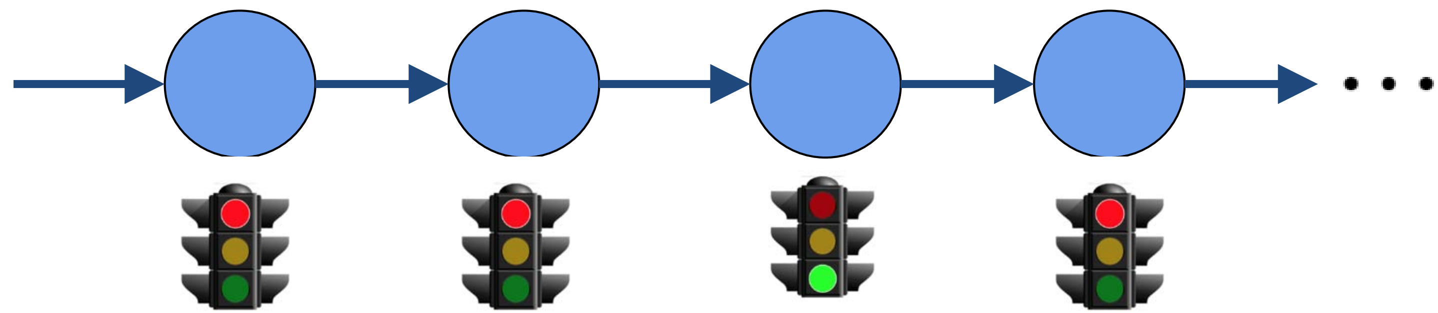
1. Propositional variables: p, ρ, ϕ, ω etc.
   Can be True or False
2. Logical Operators: ¬, ∨, ∧, →, ↔, True, False
   - ¬ = not
   - ∨ = or
   - ∧ = and
   - → = implies
   - ↔ = if and only if
   - True, False

Logical Operator Examples

Logical Operators	Example
true	**true**

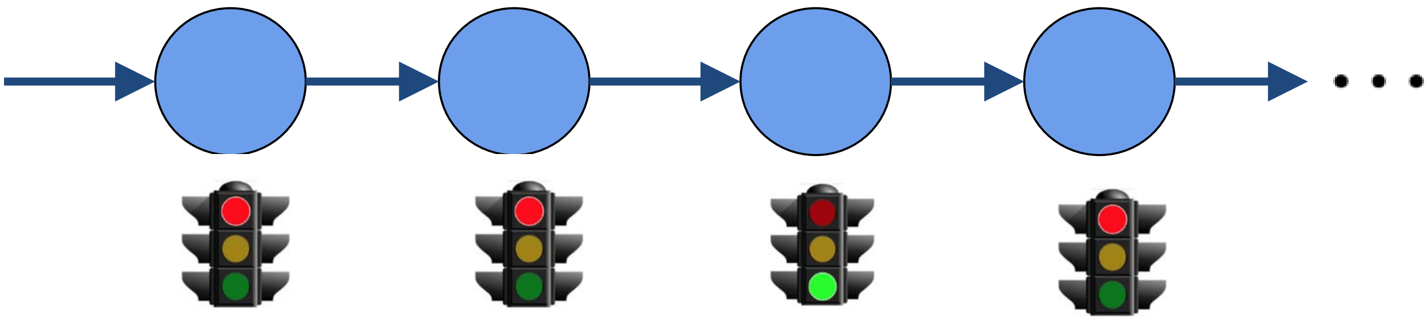


Logical Operators	Example
p = true	**R** = red light

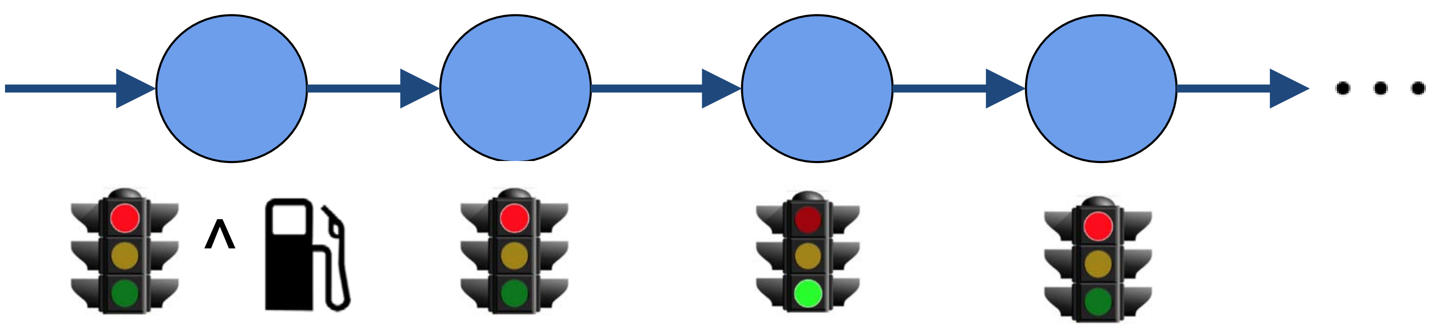


Logical Operator Examples

Logical Operators	Example
not, ¬	¬ **G** = green light

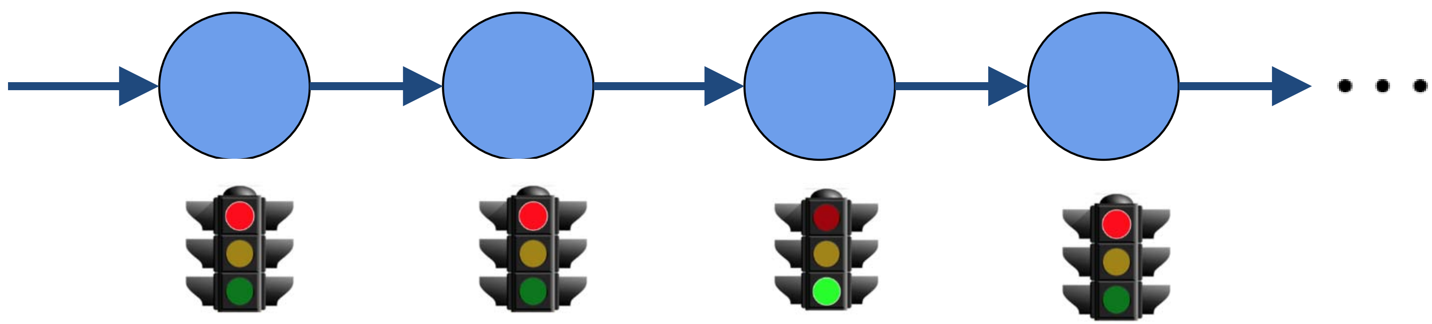


Logical Operators	Example
and, ∧	**R** ∧ **B** = gas station



Logical Operator Examples

Logical Operators	Example
or, ∨	**R** ∨ **G**



Or (∨) can be rewritten with and (∧) and not (¬)

R ∨ G = ¬(¬R ∧ ¬G)

Similar process can be done for implies and iff, but we won’t be explaining them due to time constraints

LTL Syntax

LTL formula f := true | p_i | $f_i \wedge f_j$ | $\neg f_i$ | X f_i | f_i U f_j

An LTL formula is built from:

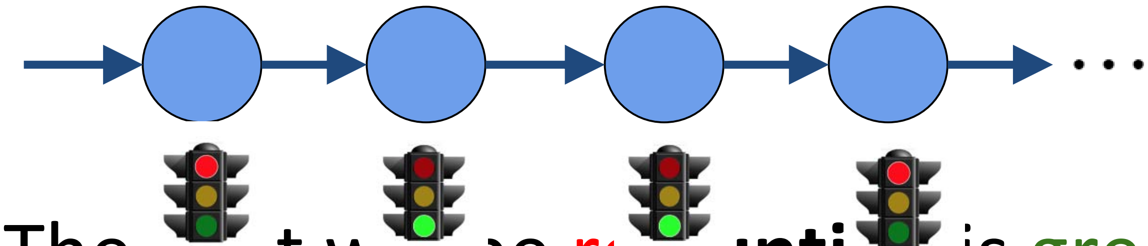
1. Propositional variables: p, ρ, ϕ, ω etc.
   Can be True or False
2. Logical Operators: ¬, ∨, ∧, →, ↔, True, False
   - ¬ = not
   - ∨ = or
   - ∧ = and
   - → = implies
   - ↔ = if and only if
   - True, False
3. Temporal Operators

Temporal Operators

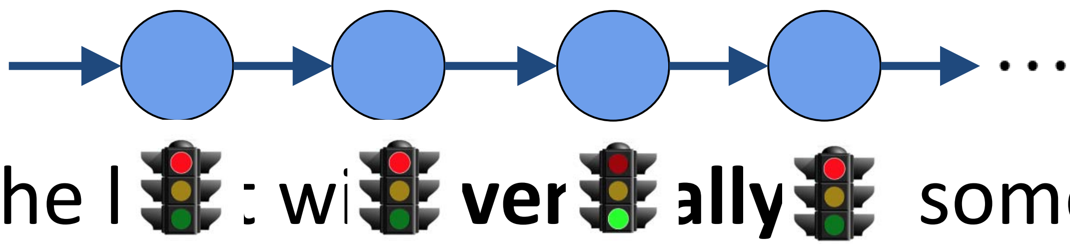
What are some useful operators we may want to describe our car?

Temporal Operators

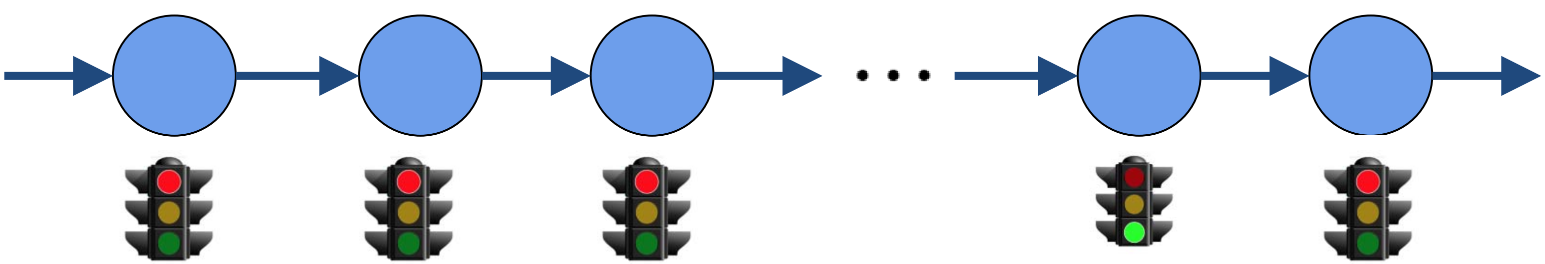
- The **next** light to be green



- The light will be red **until** it is green

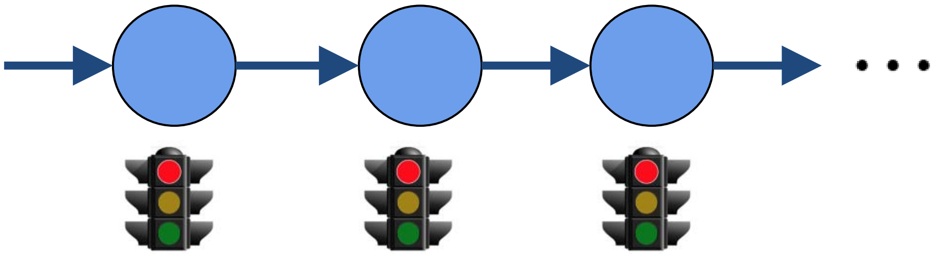


- The light will **eventually**, at some point in the **future**, turn green

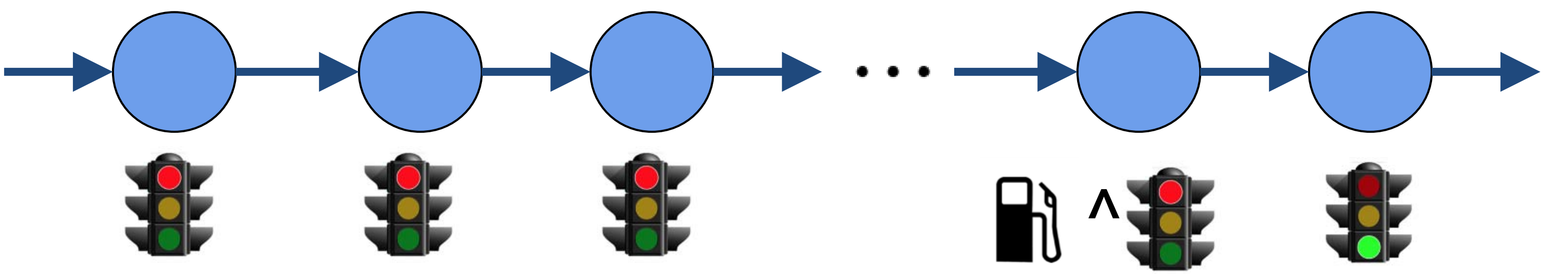


Temporal Operators

- The light will **always** be red



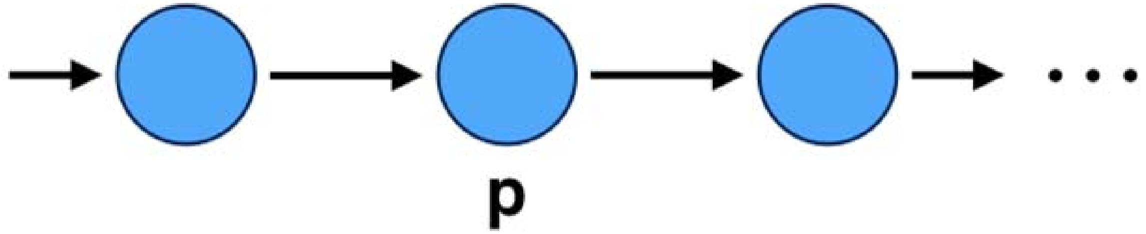
- The light will be red until the car gets gas and the state after it's **released**, the light can be whatever



Next

Operator	Textual Operator
ne**X**t	**X**ρ

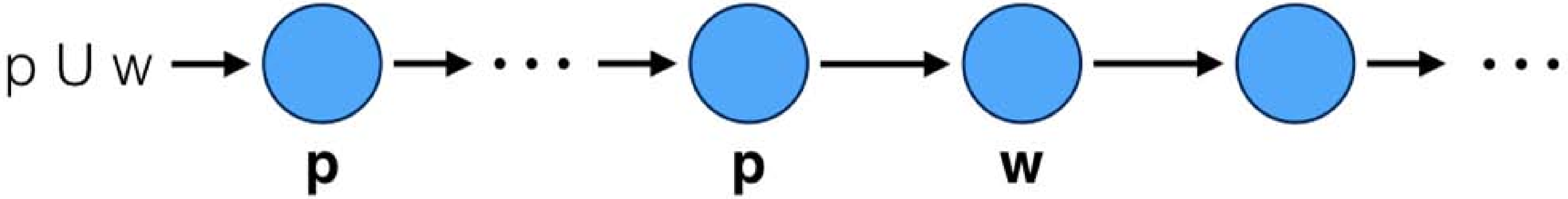
Definition: Variable ρ must be true in the next state



Until

Operator	Textual Operator
Until	ρ**U**ω

Definition: Variable ρ must remain true up until the state where variable ω becomes true, at which point ρ becomes unconstrained

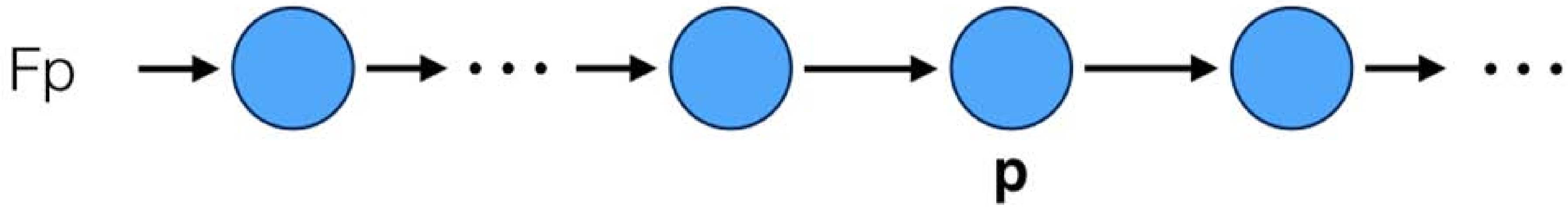


Note that ω is required to become true in some future state

Future

Operator	Textual Operator
Future/Eventually	**F**ρ

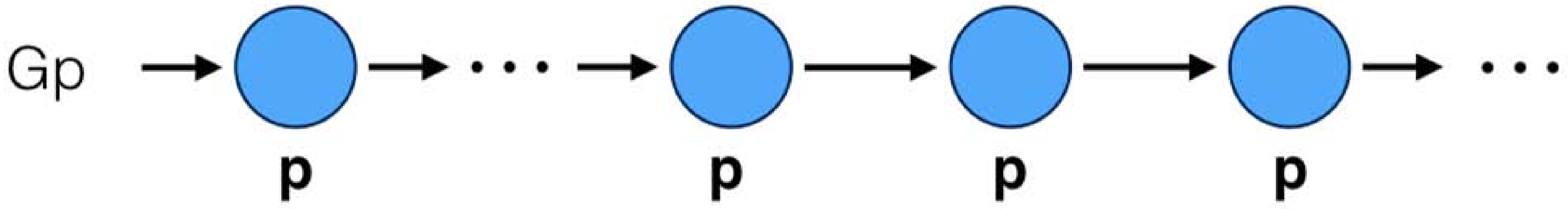
Definition: Variable ρ must become true in some future state



Global

Operator	Textual Operator
Globally	**G**ρ

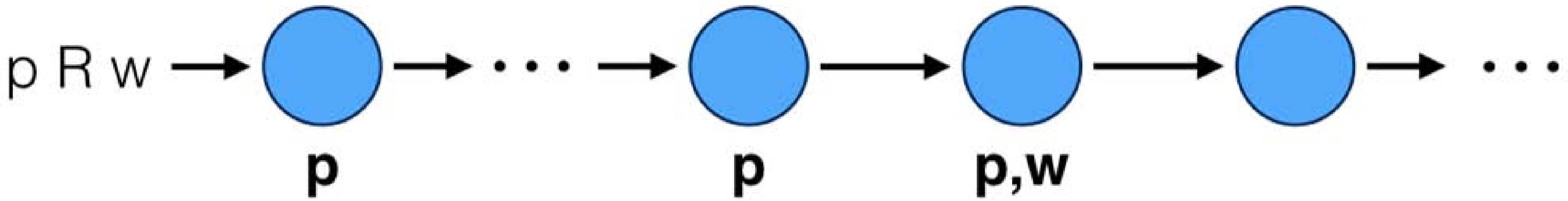
Definition: Variable ρ must be true in all future states



Release

Operator	Textual Operator
Release	ρ**R**ω

Definition: Variable ρ must be true up until and including the state where ω becomes true, after which ω is unconstrained. If ρ is not true in any future state, then ω is true in all future states



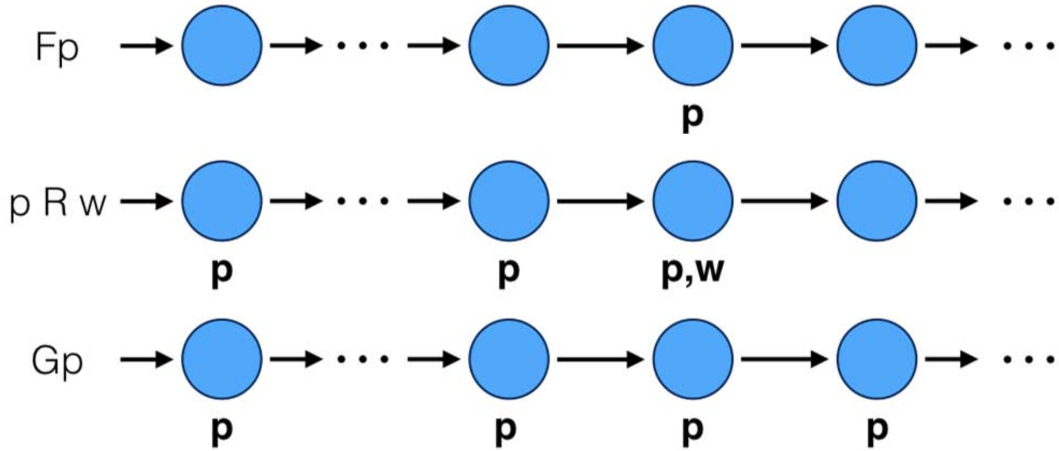
Different from **U** in that both ρ and ω are true in one state

Which describe the other?

Future/Eventually

Release

Globally



?

?

? ≡ True U ρ

≡ ¬F¬ρ

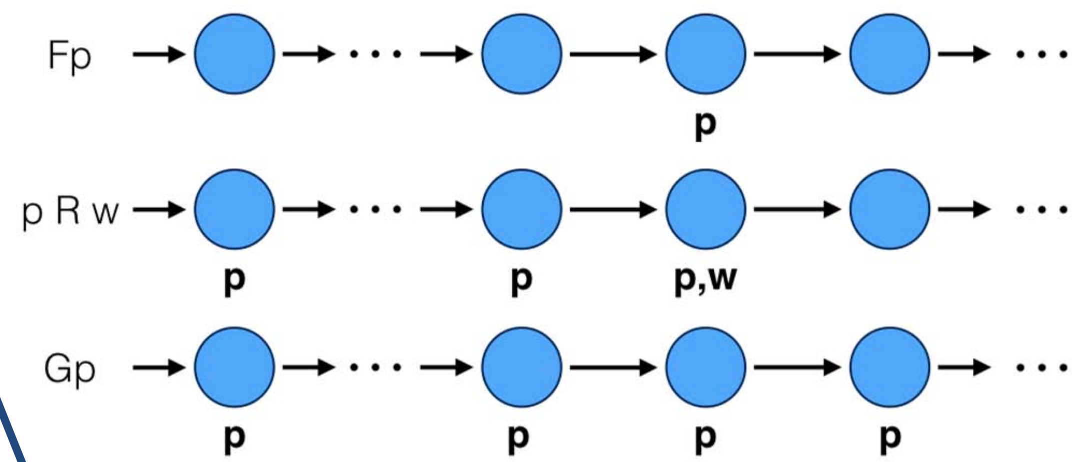
≡ ¬(¬ρ U ¬ω)

Which describe the other?

Future/Eventually

Release

Globally



≡ True U ρ

≡ ¬F¬ρ

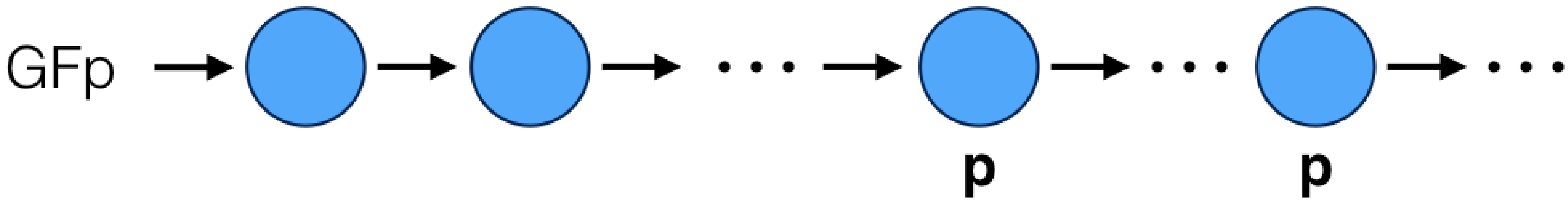
≡ ¬(¬ρ U ¬ω)

Temporal Operators (Recap)

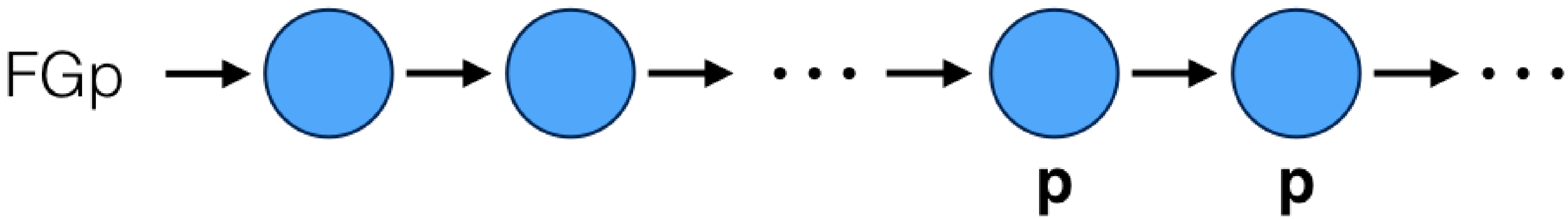
Operator	Textual Operator	
ne**X**t	**X**ρ	
Until	ρ**U**ω	
Future/Eventually	**F**ρ	≡ True U ρ
Globally	**G**ρ	≡ ¬F¬ρ
Release	ρ**R**ω	≡ ¬(¬ρ U ¬ω)

Combination of Operators

Infinitely Often

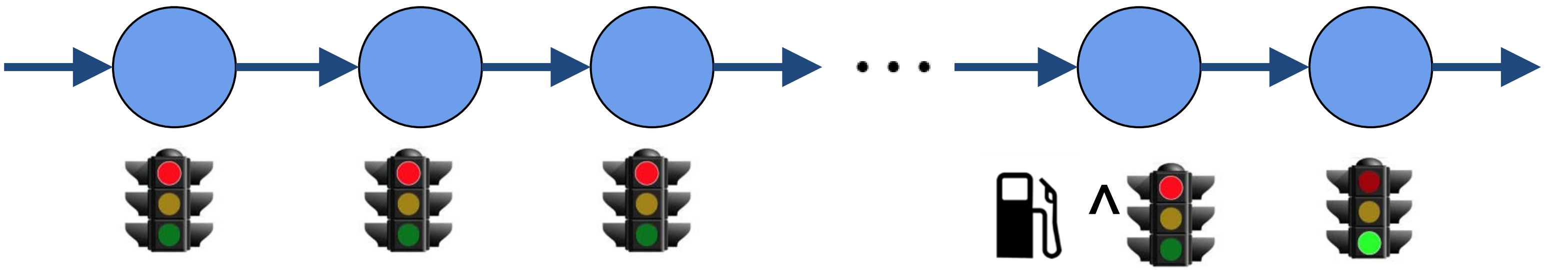


Eventually Forever



Example Problem

What are some true statements about this LTL formation?



- **X**R
- **F**G
- R**U**G
- (R**U**G)∧(**F**G)∧(**X**R)

PDDL3 Goal Description

```
<GD> ::= (at end <GD>)
   | (always <GD>)
   | (sometime <GD>)
   | (within <num> <GD>)
   | (at-most-once <GD>)
   | (sometime-after <GD> <GD>)
   | (sometime-before <GD> <GD>)
   | (always-within <num> <GD> <GD>)
   | (hold-during <num> <num> <GD> | …
```

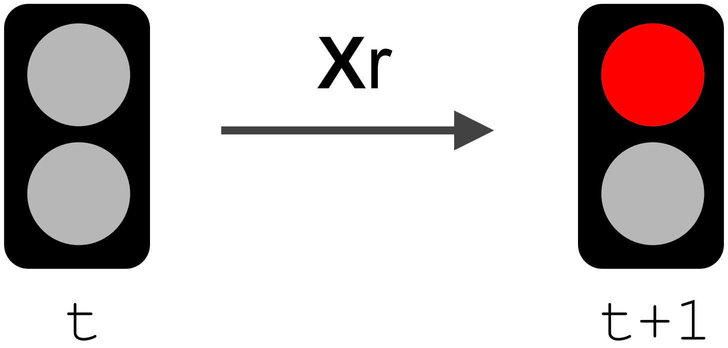
Temporal Operators

Operator		PDDL3
ne**X**t	**X**ρ	`(within 1 ρ)`
Until	ρ**U**ω	`(always-until ρ ω)`
Future	ρ**F**ω	`(sometime-after ρ ω)`
Globally	**G**ρ	`(always ρ)`
Release	ρ**R**ω	`(or (always ω) (always-until ω ρ))`

```
(or
   (always ω)
   (always-until ω ρ))
```

Expressing Temporal Logic in PDDL

- The traffic light will turn red in the next state



(:goal (within 1 (turn red)))

Command Syntax

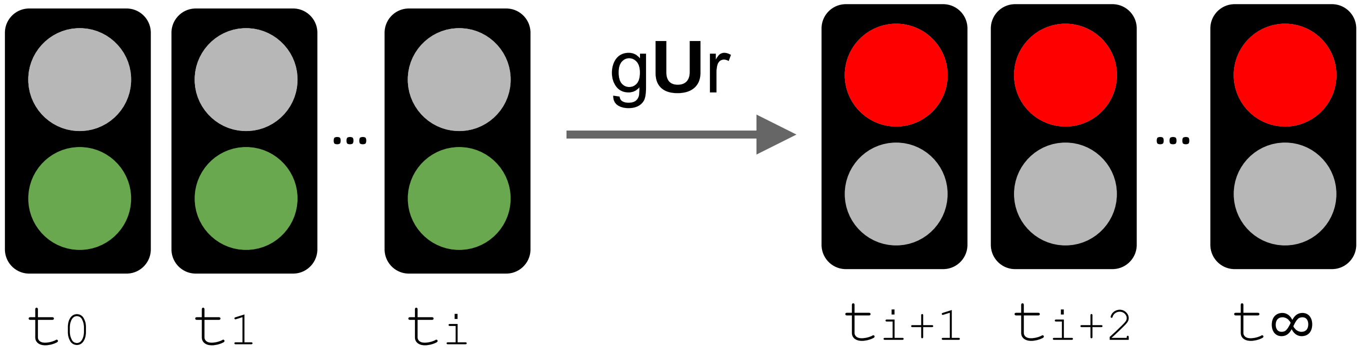
(within <num> <GD>)

(within <num> φ) would mean that φ must hold within <num> happenings

Expressing Temporal Logic in PDDL

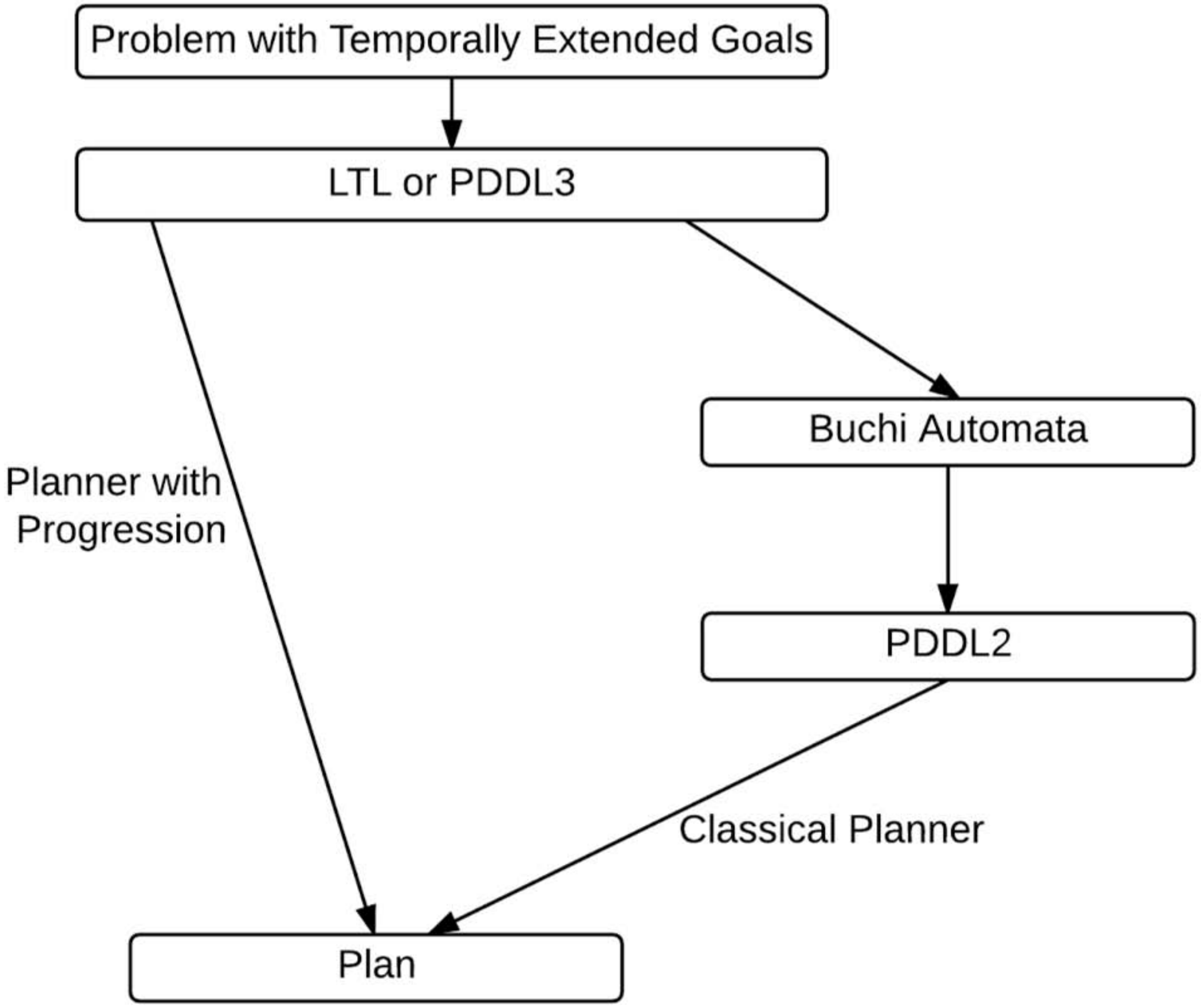
- The traffic light will be green until it turns red at which point it will be red forever

$$(g\ \mathbf{U}\ r) \wedge (r \rightarrow \mathbf{G}r)$$




```
(:goal
   (and
          (always-until (turn green) (turn
  red))
       (implies (turn red) (always (turn red)))
))
```

Application to Planning



Büchi Automata

Büchi Automata - *extension of finite automaton to infinite inputs (words)*

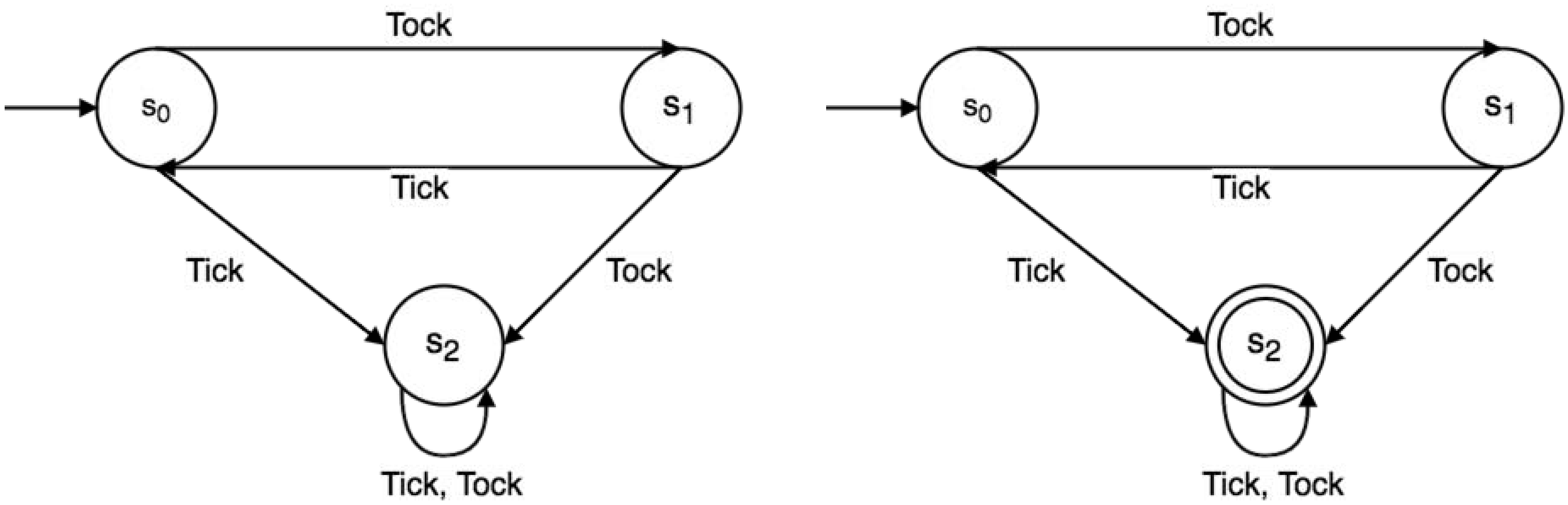
A Büchi automaton is 5-tuple $<S, s_0, T, F, \Sigma>$

- S is a finite set of states
- $s_0 \in S$ is an initial state
- $T \subseteq S \times \Sigma \rightarrow S$ is a transition relation
- $F \subseteq S$ is a set of accepting states
- Σ is a finite set of symbols ('alphabet')

An infinite sequence of states is accepted iff it visits the accepting state(s) infinitely often

Example Büchi Automata

Example: Model a clock



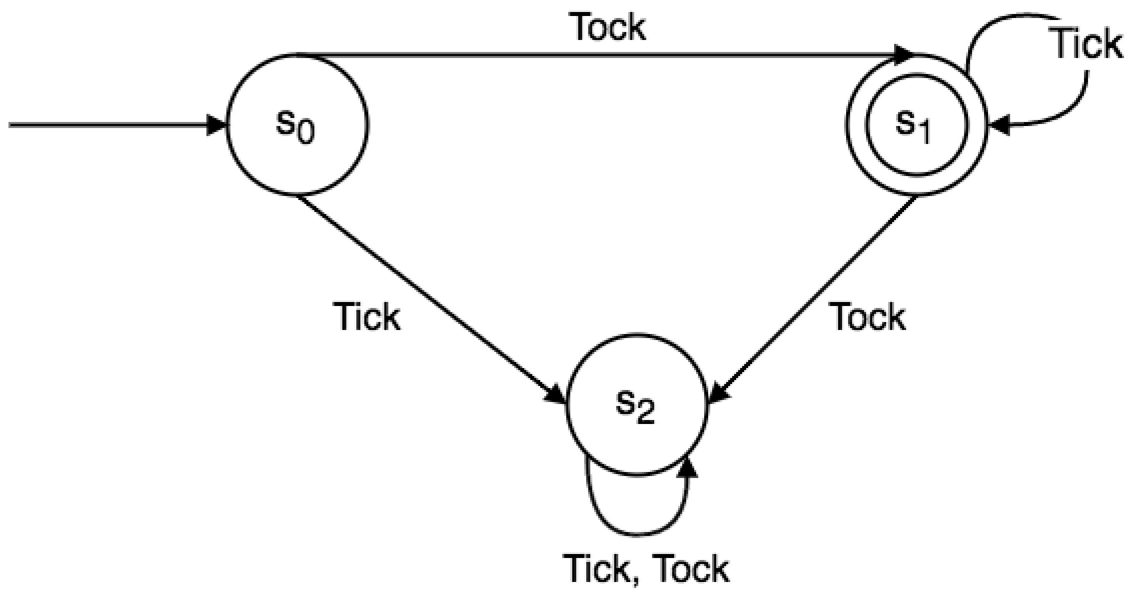
Accepted words:

TickTockTockTickTockTickTickTickTock...

TockTickTockTickTickTockTockTickTock...

Example Büchi Automata

Example: Model a clock

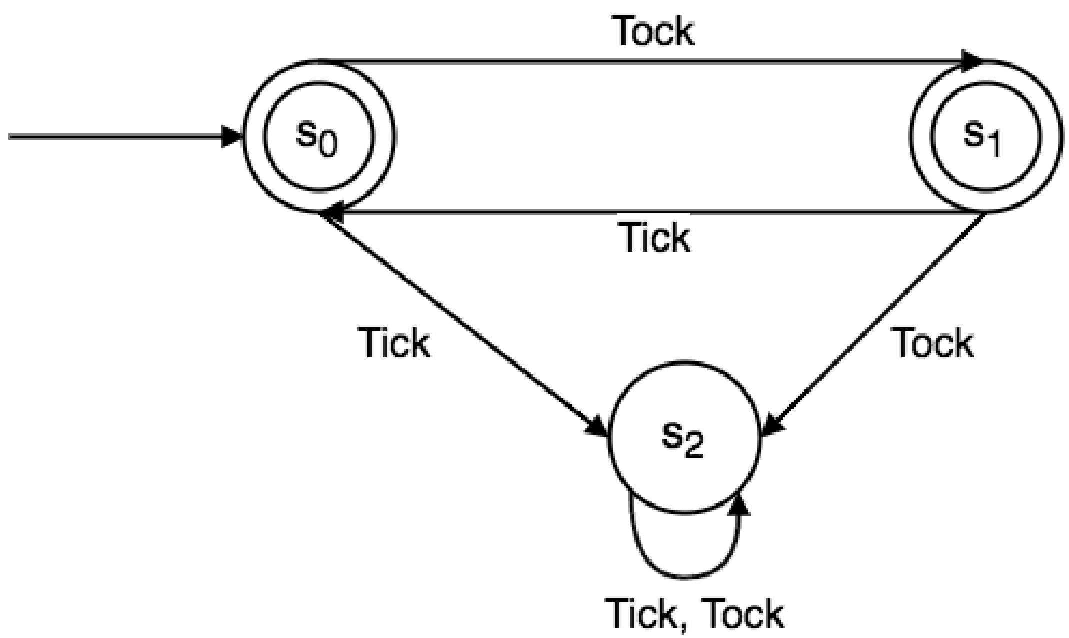


Accepted words:

TockTickTickTickTickTickTickTick...

Example Büchi Automata

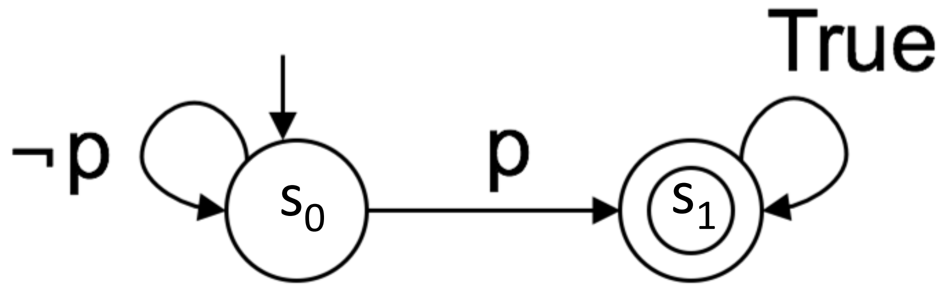
Example: Model a clock



Accepted words:

TockTickTockTickTockTickTockTick...

LTL to Büchi Automata



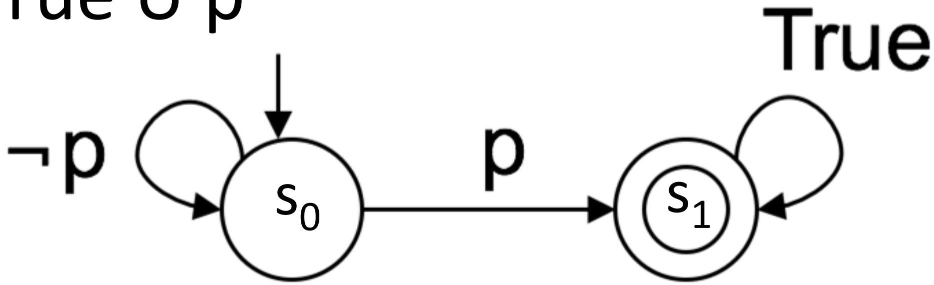
neXt?

Future/Eventually?

Globally?

LTL to Büchi Automata

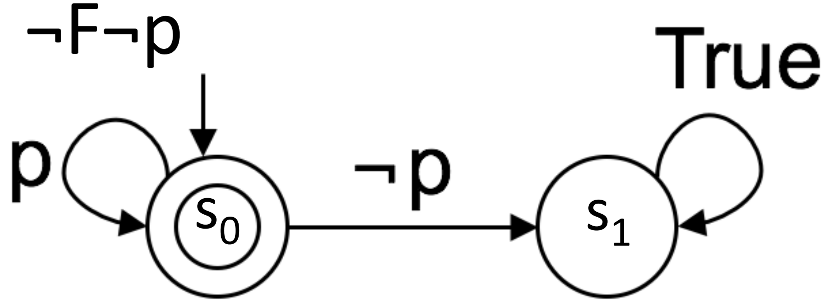
Future - **F**p ≡ True U p



Accepted word: ¬p ¬p ¬p p p ¬p …

Sequence of states: s_0 s_0 s_0 s_1 s_1 s_1…

Globally - **G**p ≡ ¬F¬p



Accepted word: p p p p p….

Sequence of states: s_0 s_0 s_0 s_0 s_0…

LTL to Büchi Algorithm

N – Node object

- N.curr – LTL formulas to be processed
- N.old – LTL formulas already processed
- N.next – LTL formulas to be processed in next node
- N.incoming – Incoming transitions from predecessor nodes

N_s – List of processed Nodes

N_i – Arbitrary node from N_s

```
expand (N,Ns)
        if N.curr is empty
                if N.curr = Ni.curr
                        Append N.curr to Ni.curr
                else
                        Append N to Ns
                        Create new node Nnew with Nnew.curr = N.next
                        expand (Nnew, Ns)
        else
                Remove an LTL formula f from N.curr and append to N.old
                Perform Progression on f
                Call expand on result of Progression
```

The result of this algorithm is a generalized Buchi automata which is then transformed into a simple Buchi automata.

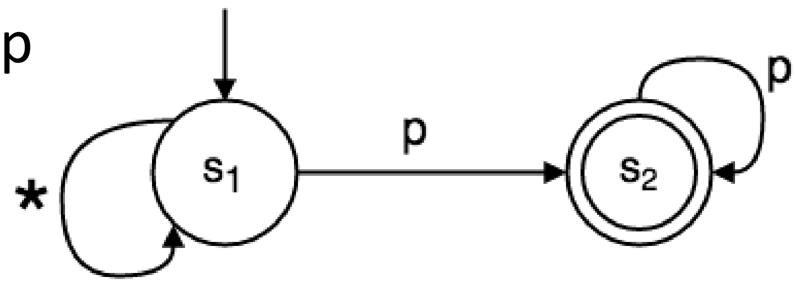
Progression Algorithm

```
progress(f,N, Δt = 1)  #Δt is time between successive states
    if f contains no temporal qualities:
        if N.curr entails f:
                f' = True
        else
                f' = False
    if f = f_1 ∧ f_2:
        progress(f_1, N, Δt) ∧ progress(f_2, N, Δt)
    if f = Xf_1:
        N.next.append(f_1)
    if f = f_1 U_[a,b] f_2:                #[a,b] is a time interval that could be infinite
        if b < a:
            f' = False
        else if 0 ∈ [a,b]:
                progress(f_2, N, Δt) ∨ (progress(f_1,N, Δt) ∧ N.next.append(f_1 U_[a,b] - Δt
        f))
        else
                progress(f_1,N, Δt) ∧ N.next.append(f_1 U_[a,b] - Δt f)
```

Büchi Automata to PDDL2

Büchi states are not equivalent to PDDL2 states. Consider:

Future**G**lobally - **FG**p



Two ways to transform temporally extended goals to PDDL2:

- Create new actions that encapsulate the allowable transitions in each state
- Introduce derived predicates
  - Do not depend on the actions
  - Used to determine which state the planner is in
  - Goal of the planner is to move from initial state to any accepting state

Planning with Preferences

Preference Based Planning

Classical Planning Problem

$$problem := (S, s_0, A, G)$$

S - set of states *s_0 - initial state* *A - set of operators* *G - set of goal states*

Preference-based Planning Problem

$$problem := (S, s_0, A, G, R)$$

R is a partial or total relation expressing preferences ($\preccurlyeq$) between plans

Preferences express properties of the plan that are desired but not required

Preference Expression Languages

- **Quantitative** - *assign numeric values to plans to compare them*
  - Markov Decision Processes (MDP's)
    - Find preferred policy using a reward function over conditional plans
  - PDDL3
    - Preferences expressed through reward function based on satisfying/violating logical formulas on the plan
- **Qualitative** - *relations compare plans based on properties of the plans that need not be numeric*
  - Ranked Knowledge Bases
    - Plan properties are ranked with preferred formulas ranked higher
  - Temporally Extended Preferences
    - Use LTL to express plan properties that are then ranked

Quantitative languages imply total comparibility while qualitative languages may allow incomparability

Expressing Preferences in PDDL3

Syntax for modeling preferences:

`(preference [name] <GD>)` - *label for fluents that represent preferences*

`is-violated` - *function that returns the number of times the preference was not satisfied in the plan*

Example:

Traffic light is green until it turns red

```
(preference gUr
        (always-until(turn green) (turn
red)))
```

Plan tries to not violate any preferred fluents

```
(metric minimize (is-violated gUr))
```

LPP Language Overview

- LPP is a quantitative language to express temporal preferences for planning
  - Preferences between different temporal goals can be expressed along with the strength of preference
    - i.e. Goal A is preferred twice as much as Goal B
- LPP is an extension of an older language PP
- Preference formulas in LPP are constructed hierarchically

See Bienvenu, Meghyn, Christian Fritz, and Sheila A. McIlraith. "Planning with Qualitative Temporal Preferences." KR 6 (2006): 134-144.

Constructing a Preference Formula

Basic Desire Formula (BDFs)

express temporally extended propositions

- At some point, will cook
  - $b_1 = \mathbf{F}(\text{cook})$
- At some point, will order takeout
  - $b_2 = \mathbf{F}(\text{orderTakeout})$
- At some point, will eat spaghetti
  - $b_3 = \mathbf{F}(\text{eatSpaghetti})$
- At some point, will eat pizza
  - $b_4 = \mathbf{F}(\text{eatPizza})$

Constructing a Preference Formula

Atomic Preference Formulas (APFs)

express preferences between BDFs

- In this example, weights associated with each BDF define preferences
  - Lower weight is preferred

- Prefer to cook over ordering takeout
  - $a_1 = b_1[0.2] \gg b_2[0.4]$
- Prefer eating spaghetti over eating pizza
  - $a_2 = b_3[0.3] \gg b_4[0.9]$

Constructing a Preference Formula

General Preference Formulas (GPFs)

allow conjunctions or disjunctions of APFs or qualification of BDFs with conditionals

- Satisfy the most preferred option among the APFs (satisfy APF with lowest weight)
  - $g_1 = a_1 \mid a_2$
- Choose the most preferred option that satisfies both APFs (minimize the maximum weight across both APFs)
  - $g_2 = a_1 \ \& \ a_2$

Constructing a Preference Formula

Aggregated Preferences Formulas (APFs)
define the order in which preferences should be relaxed

- Prefer that if both g_1 and g_2 from previous slide can't be met, that g_2 from previous slide is met
  - $g_1 \wedge g_2 \preccurlyeq g_2 \preccurlyeq g_1$
- Situations that aren't distinguished any other way can be sorted lexicographically (alphabetically)

LPP Formula Hierarchy Review

- **Basic Desire Formula (BDF)**
  - Express temporally extended propositions
- **Atomic Preference Formula (APF)**
  - Express preferences between BDFs
- **General Preference Formula (GPF)**
  - Allow conjunctions or disjunctions of APFs or qualification of BDFs with conditionals
- **Aggregated Preference Formula (APF)**
  - Define the order in which preferences should be relaxed

References

Gerevini, A., and D. Long. *Plan constraints and preferences in PDDL3: The language of the fifth international planning competition. University of Brescia*. Italy, Tech. Rep, 2005.

Patrizini, Fabio, et al. "Computing infinite plans for LTL goals using a classical planner." *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence; july 16-22, 2011; Barcelona. Menlo Park, California: AAAI Press; 2011. p. 2003-2008.*. Association for the Advancement of Artificial Intelligence (AAAI), 2011.

Baier, Jorge A., and Sheila A. McIlraith. "Planning with Temporally Extended Goals Using Heuristic Search." *ICAPS*. 2006.

Bacchus, Fahiem, and Froduald Kabanza. "Planning for temporally extended goals." *Annals of Mathematics and Artificial Intelligence* 22.1-2 (1998): 5-27.

Baier, Jorge A., and Sheila A. McIlraith. "Planning with first-order temporally extended goals using heuristic search." *Proceedings of the National Conference on Artificial Intelligence*. Vol. 21. No. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Baier, Jorge A., and Sheila A. McIlraith. "Planning with preferences." *AI Magazine* 29.4 (2009): 25.

Bienvenu, Meghyn, Christian Fritz, and Sheila A. McIlraith. "Planning with Qualitative Temporal Preferences." *KR* 6 (2006): 134-144.

Gerth, Rob, et al. "Simple on-the-fly automatic verification of linear temporal logic." *Protocol Specification, Testing and Verification XV*. Springer US, 1996. 3-18.

Appendix

Solving Planning Problems with Preferences

- PPLAN
  - implemented by Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith
- Solves planning problems with preferences expressed in LPP via bounded best-first search forward chaining planner
  - use of *progression* efficiently evaluates how well partial plans satisfy Φ (a general preference formula)
  - use of admissible *evaluation function* ensures best-first search is optimal

Quick Definitions

- Forward Chaining Planner - Forward chaining starts with the available data and uses inference rules to extract more data (from an end user, for example) until a goal is reached.
- A situation s is a *history* of the primitive actions $a \in A$ performed from an initial situation S0.

Progression

- Purpose of progression:
  - take in a situation and temporal logic formula (TLF)
  - evaluates the TLF with respect to the state of the situation
  - generates a new formula representing those aspects of the TLF that remain to be satisfied in subsequent situations.
- Weight of general preference formula with respect to a situation is equal to progressed preference formula with respect to final situation

Evaluation Function

- Evaluation Function
  - has optimistic and pessimistic weights to provide best and worst weights on a successor with respect to Φ.
  - the optimistic weight is non-decreases and does not over-estimate the actual weight
  - this allows PPLAN to define an optimal search algorithm

PPLAN Algorithm

optW = optimistic weight (Assumes all unfulfilled preferences are fulfilled)

pessW = pessimistic weight (Assumes all unfulfilled preferences are not fulfilled)

Algorithm

L = list of nodes sorted by optW, then pessW, then length

while L is not empty
    Remove first node from L
    If goal is achieved and optW = pessW
        return partial plan, optW
    Perform Progression
    Add new nodes to L and sort

PPLAN

- PPLAN is implemented with a
  - general preference formula fΦ they define is admissible and when used in best first search, the search is optimal
  - the best first search searches through the partial plans based on their weights
  - for full details see paper "Planning with Qualitative Temporal Preferences" by Fritz, Christian, Sheila A. McIlraith, and Meghyn Bienvenu.

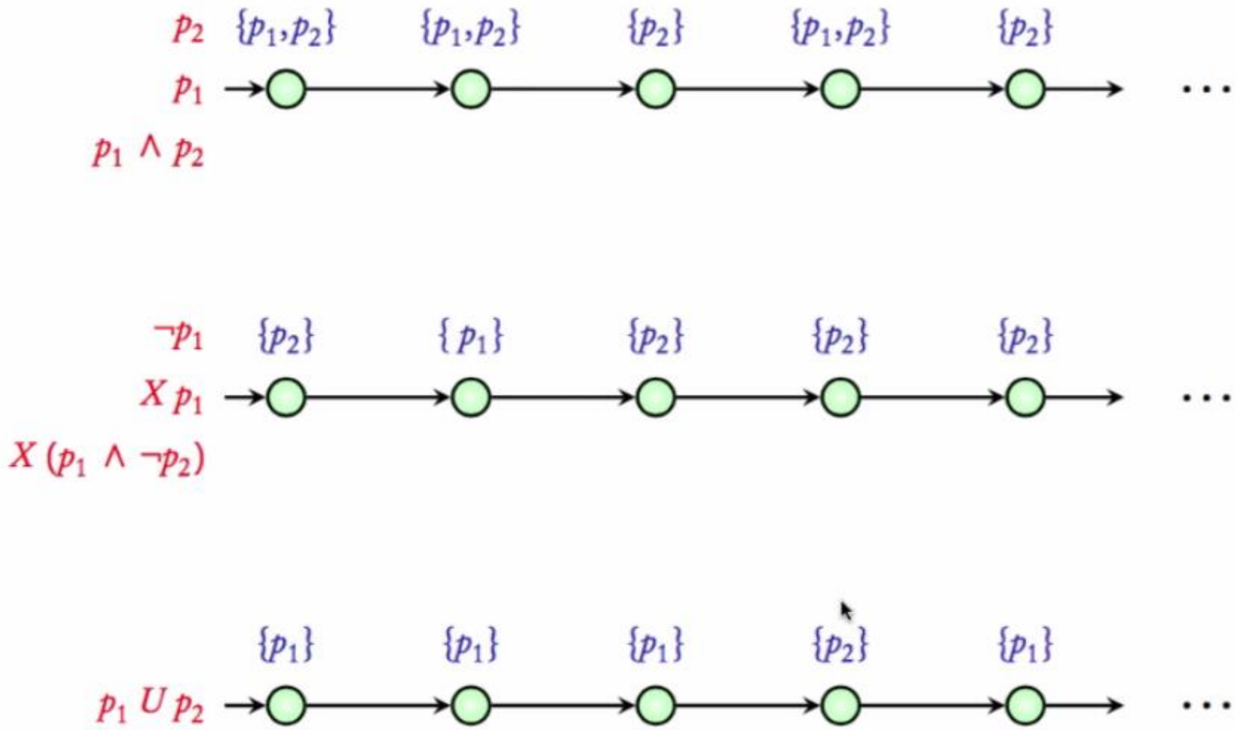
Additional Examples were taken from the youtube videos of NOC15 July-Oct CS12 : https://www.youtube.com/watch?v=W5Q0DL9plns

Example Formulations

- Traffic light is red: r
- Traffic light is green: g

- The traffic light will turn red in the next state
  - Xr
- The traffic light will be green until it turns red but it may not ever turn red
  - $(g \, U \, r) \vee Gg$ (Weak Until)
- The traffic light will be green until it turns red at which point it will be red forever
  - $(g \, U \, r) \wedge (r \rightarrow Gr)$

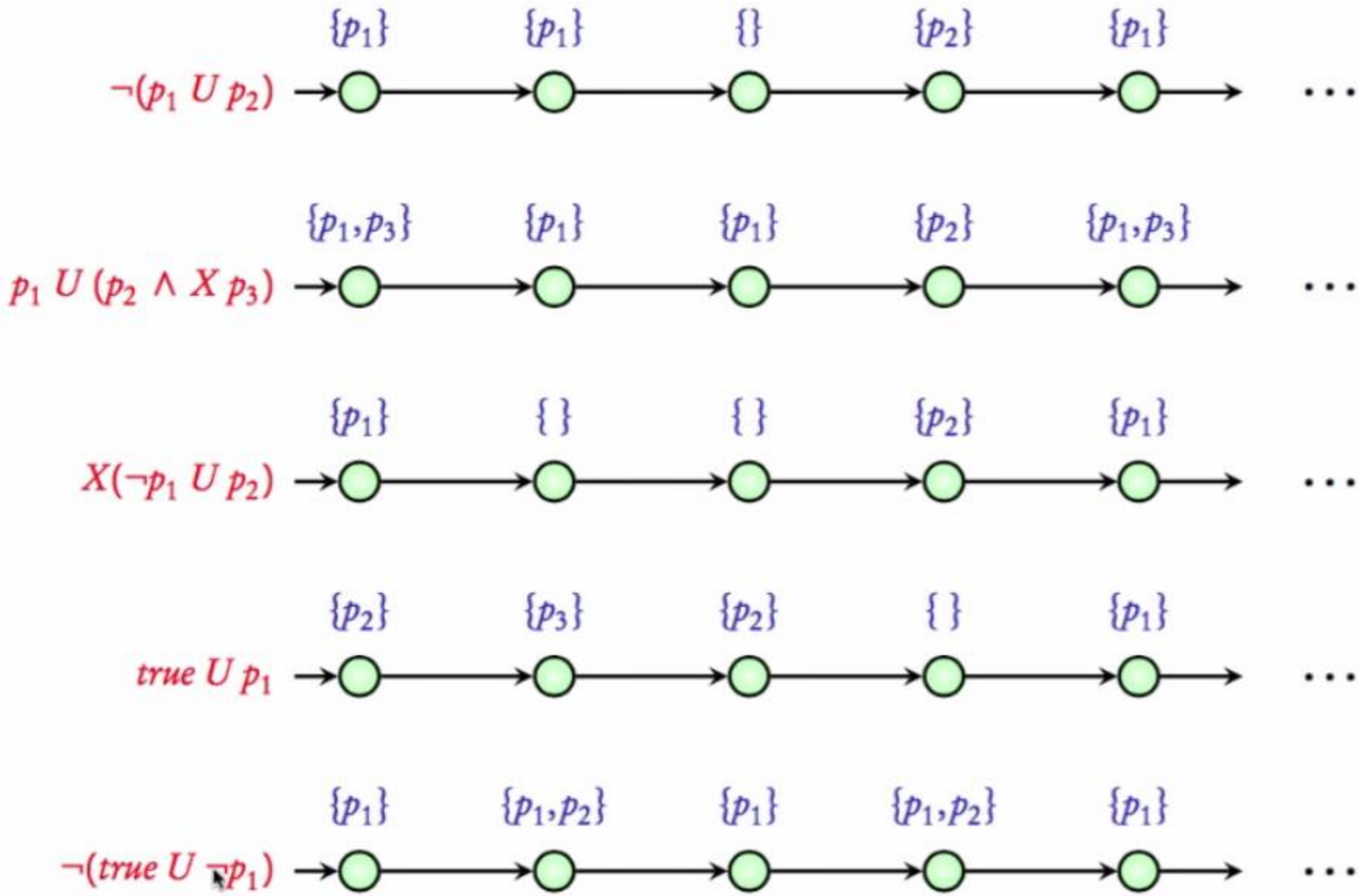
Additional Examples



$$\phi := \text{true} \mid p_i \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid X\,\phi \mid \phi_1\, U\, \phi_2$$

$p_i \in AP$ ϕ_1, ϕ_2 : LTL formulas

$$\phi := \text{true} \mid p_i \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid X\,\phi \mid \phi_1\, U\, \phi_2$$



16.412J / 6.834J Cognitive Robotics
Spring 2016