TEMPORAL PLANNING

Stefan Edelkamp

PUI - CTU

AI CENTER
FEE CTU

HISTORY

The development of PDDL is mainly driven by the **International Planning Competition (IPC):**

- **1998:** PDDL [McDermott and others (1998)]
  STRIPS and ADL.
- **2000:** "PDDL subset for the 2000 competition" [Bacchus (2000)]
  STRIPS and ADL.
- **2002:** PDDL2.1, Levels 1-3 [Fox and Long (2003)]
  Numeric and temporal planning.
- **2004:** PDDL2.2 [Hoffmann and Edelkamp (2005)]
  Derived predicates and timed initial literals.
- **2006:** PDDL3 [Gerevini *et al.* (2009)]
  Soft goals and trajectory constraints.

PDDL 2.1

Maria Fox and Derek Long promoted **numeric and temporal planning:**

- **PDDL2.1 level 1**: As in IPC'00.
- **PDDL2.1 level 2**: Level 1 plus numeric fluents. Comparisons between numeric expressions are allowed as logical atoms:
  `(>= (fuel) (* (dist ?x ?y) (consumption)))`
  Effects can modify fluents by numeric expressions:
  `(decrease (fuel) (* (dist ?x ?y) (consumption)))`
- **PDDL2.1 level 3**: Level 2 extended with action durations. Actions take an amount of time given by the value of a numeric expression:
  `(= ?duration (/ (dist ?x ?y) (speed))`
  Conditions/effects are applied at either start or end of action:
  `(at start (not (at ?x))) (at end (at ?y))`

DURATIVE ACTIONS

Respect different time spans to execute an action

```
(:durative-action refuel
 :parameters (?a - aircraft ?c - city)
 :duration (= ?duration (/ (- (capacity ?a) (fuel ?a)) (rate ?a)))
 :condition (and (at start (> (capacity ?a) (fuel ?a)))
                 (over all (at ?a ?c)))
 :effect (at end (assign (fuel ?a) (capacity ?a))))
```

Temporal Modalities:

(at start …) preconditions and change when action is initiated

(at end …), preconditions and effects when action terminates

(over all …), invariance condition to be true throughout the action

GENERAL OBJECTIVE FUNCTION – PLAN METRIC

In propositional planning, there are only two sensible metrics:

- number of operators in a total-ordered plan

- depth of operators in a partial-ordered plan

With introducing numbers and durations there are now more expressive options,

like the makespan (predicate implicitly used in the planner).

```
(:metric minimize total-time)
```

or arithmetic expressions over the numerical variable

```
(:metric minimize (+ (* 2 total-time) (* 4 total-fuel-consumed)))
```

PDDL 2.2

PDDL2.2:

- Derived predicates: Predicates that are not affected by the actions. Their value is instead derived via a set of derivation rules of the form **IF** $\varphi(\overline{x})$ **THEN** $P(\overline{x})$.
  Example: Flow of current in an electricity network.
  ```
  (:derived (fed ?x)
              (exists ?y (and (connected ?x ?y) (fed ?y))))
  ```
- Timed Initial Literals: Literals that will become true, independently of the actions taken, at a pre-specified point in time.
  Example: Opening/closing times.
  ```
  (at 9 (shop-open)) (at 18 (not (shop-open)))
  ```

TILS

Timed initial literals specified in the initial state

```
(at 25 (open-station city0))
(at 75 (not (open-station city0)))
```

leads to time window for the durative actions if timed initial literal is mentioned in the preconditions.

Can be compiled away in polynomial time to PDDL2.1 in a linear number of steps.

REPOSITORY

http://planning.domains

Includes huge number PDDL domains/instances, a PDDL editor, lectures explaining

PDDL etc.

International Planning Competitions:

http://icaps-conference.org/index.php/Main/Competitions

Latest (9th)

https://ipc2018.bitbucket.io/

Also includes results and participating planners in source code!

SUMMARY ON PDDL

- PDDL is the de-facto standard for classical planning, as well as extensions to numeric/temporal planning, soft goals, trajectory constraints.
- PDDL is used in the International Planning Competition (IPC).
- PDDL uses a schematic encoding, with variables ranging over objects similarly as in predicate logic. Most implemented systems use grounding to transform this into a propositional encoding.
- PDDL has a Lisp-like syntax.

TEMPORAL PLANNING: TIME DOES MATTER

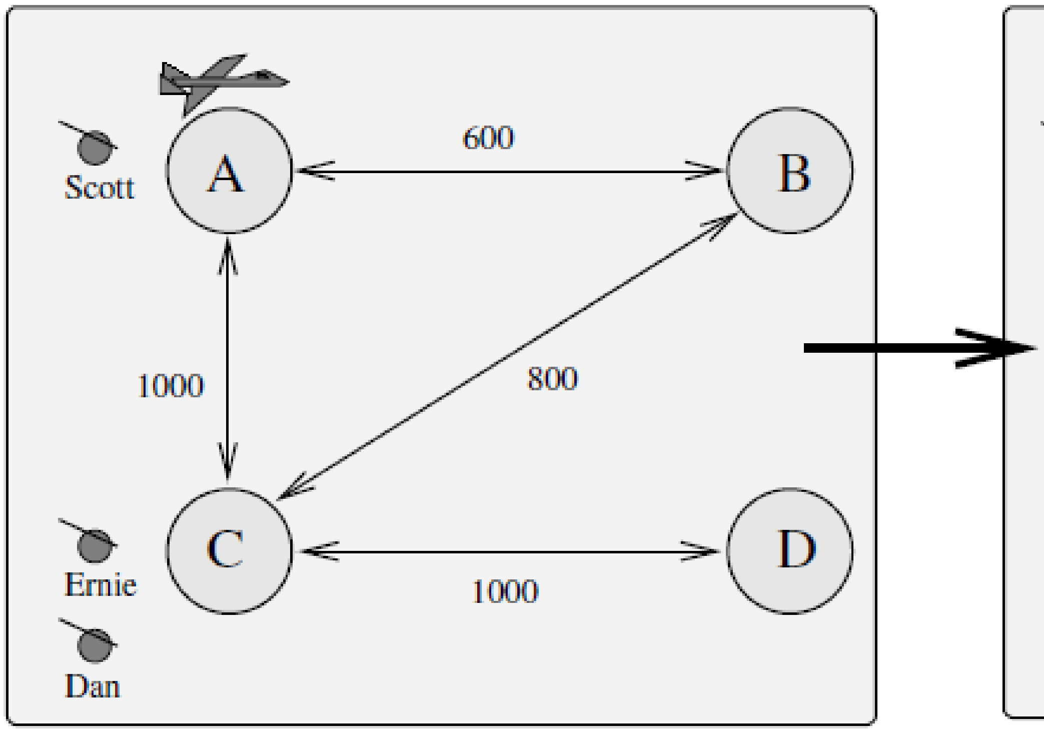
In general, activities have **varying durations**:

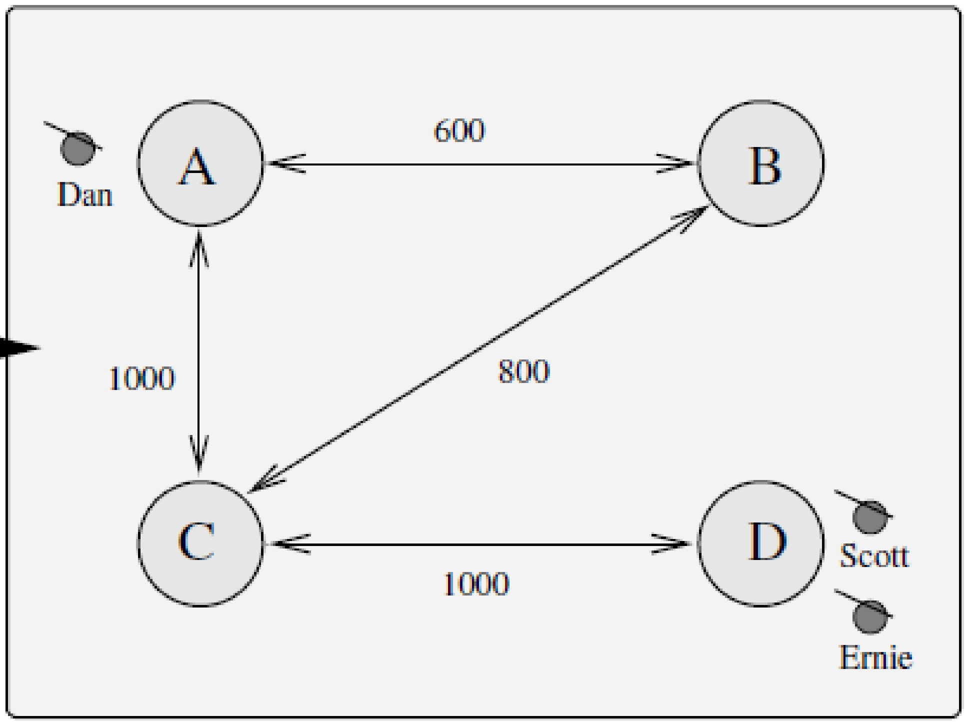
- Loading a package onto a truck is much quicker than driving the truck;
- Drinking a cup of tea takes longer than making it;
- Procrastinating tasks takes longer than doing them
- …

EXAMPLE: ZENO DOMAIN

INITIAL STATE

GOAL STATE





ZENO PDDL DOMAIN FILE

```
(define (domain zeno-travel)
(:requirements :durative-actions :typing :fluents)
(:types aircraft person city)
(:predicates (at ?x - (either person aircraft) ?c - city)
             (in ?p - person ?a - aircraft))
(:functions (fuel ?a - aircraft) (distance ?c1 - city ?c2 - city)
            (slow-speed ?a - aircraft) (fast-speed ?a - aircraft)
            (slow-burn ?a - aircraft)  (fast-burn ?a - aircraft)
            (capacity ?a - aircraft)   (refuel-rate ?a - aircraft)
            (total-fuel-used) (boarding-time) (debarking-time))
(:durative-action board
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration boarding-time)
 :condition (and (at start (at ?p ?c))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (at ?p ?c)))
              (at end (in ?p ?a))))
[...]
(:durative-action zoom
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :duration (= ?duration (/ (distance ?c1 ?c2) (fast-speed ?a)))
 :condition (and (at start (at ?a ?c1))
                 (at start (>= (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a)))))
 :effect (and (at start (not (at ?a ?c1)))
              (at end (at ?a ?c2))
              (at end (increase total-fuel-used
                           (* (distance ?c1 ?c2) (fast-burn ?a))))
              (at end (decrease (fuel ?a)
                           (* (distance ?c1 ?c2) (fast-burn ?a))))))
```

ZENO PDDL PROBLEM FILE

```
(define (problem zeno-travel-1)
    (:domain zeno-travel)
    (:objects plane - aircraft
              ernie scott dan - person
              city-a city-b city-c city-d - city)
    (:init (= total-fuel-used 0) (= debarking-time 20) (= boarding-time 30)
           (= (distance city-a city-b) 600)  (= (distance city-b city-a) 600)
           (= (distance city-b city-c) 800)  (= (distance city-c city-b) 800)
           (= (distance city-a city-c) 1000) (= (distance city-c city-a) 1000)
           (= (distance city-c city-d) 1000) (= (distance city-d city-c) 1000)
           (= (fast-speed plane) (/ 600 60)) (= (slow-speed plane) (/ 400 60))
           (= (fuel plane) 750)              (= (capacity plane) 750)
           (= (fast-burn plane) (/ 1 2))     (= (slow-burn plane) (/ 1 3))
           (= (refuel-rate plane) (/ 750 60))
           (at plane city-a) (at scott city-a) (at dan city-c) (at ernie city-c))
    (:goal (and (at dan city-a) (at ernie city-d) (at scott city-d)))
    (:metric minimize total-time)
)
```

SEQUENTIAL AND TEMPORAL PLAN

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c)     [30]
130: (board ernie plane city-c)   [30]
160: (refuel plane city-c)        [40]
200: (zoom plane city-c city-a) [100]
300: (debark dan plane city-a)    [20]
320: (board scott plane city-a)   [30]
350: (refuel plane city-a)        [40]
390: (zoom plane city-a city-c) [100]
490: (refuel plane city-c)        [40]
530: (zoom plane city-c city-d) [100]
630: (debark ernie plane city-d) [20]
650: (debark scott plane city-d) [20]
```

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c)     [30]
     (board ernie plane city-c)   [30]
100: (refuel plane city-c)        [40]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a)    [20]
     (board scott plane city-a)   [30]
     (refuel plane city-a)        [40]
280: (zoom plane city-a city-c) [100]
380: (refuel plane city-c)        [40]
420: (zoom plane city-c city-d) [100]
520: (debark ernie plane city-d) [20]
     (debark scott plane city-d) [20]
```

SNAG: SEQUENTIAL PLAN TIME VS. PARALLEL PLAN TIME

```
(zoom city-a city-c plane), (board dan plane city-c),
(refuel plane city-c), (zoom city-c city-a plane),
(board scott plane city-a), (debark dan plane city-a), (refuel plane city-a),
```

and

```
(board scott plane city-a), (zoom city-a city-c plane),
(board dan plane city-c), (refuel plane city-c),
(zoom city-c city-a plane), (debark dan plane city-a), (refuel plane city-a)
```

DIFFERENT PLAN OBJECTIVES

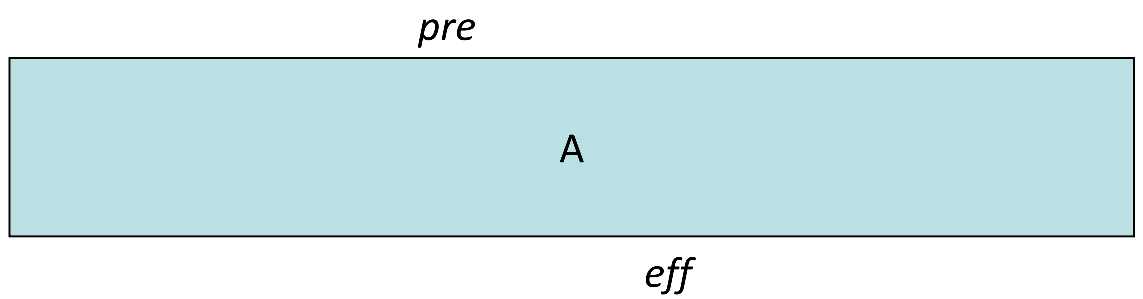
FUEL

```
     0: (board scott plane city-a) [30]
    30: (fly plane city-a city-c) [150]
   180: (board ernie plane city-c) [30]
        (board dan plane city-c)   [30]
   210: (fly plane city-c city-a) [150]
   360: (debark dan plane city-a)  [20]
        (refuel plane city-a)   [53.33]
413.33: (fly plane city-a city-c) [150]
563.33: (fly plane city-c city-d) [150]
713.33: (debark ernie plane city-d)[20]
        (debark scott plane city-d)[20]
```

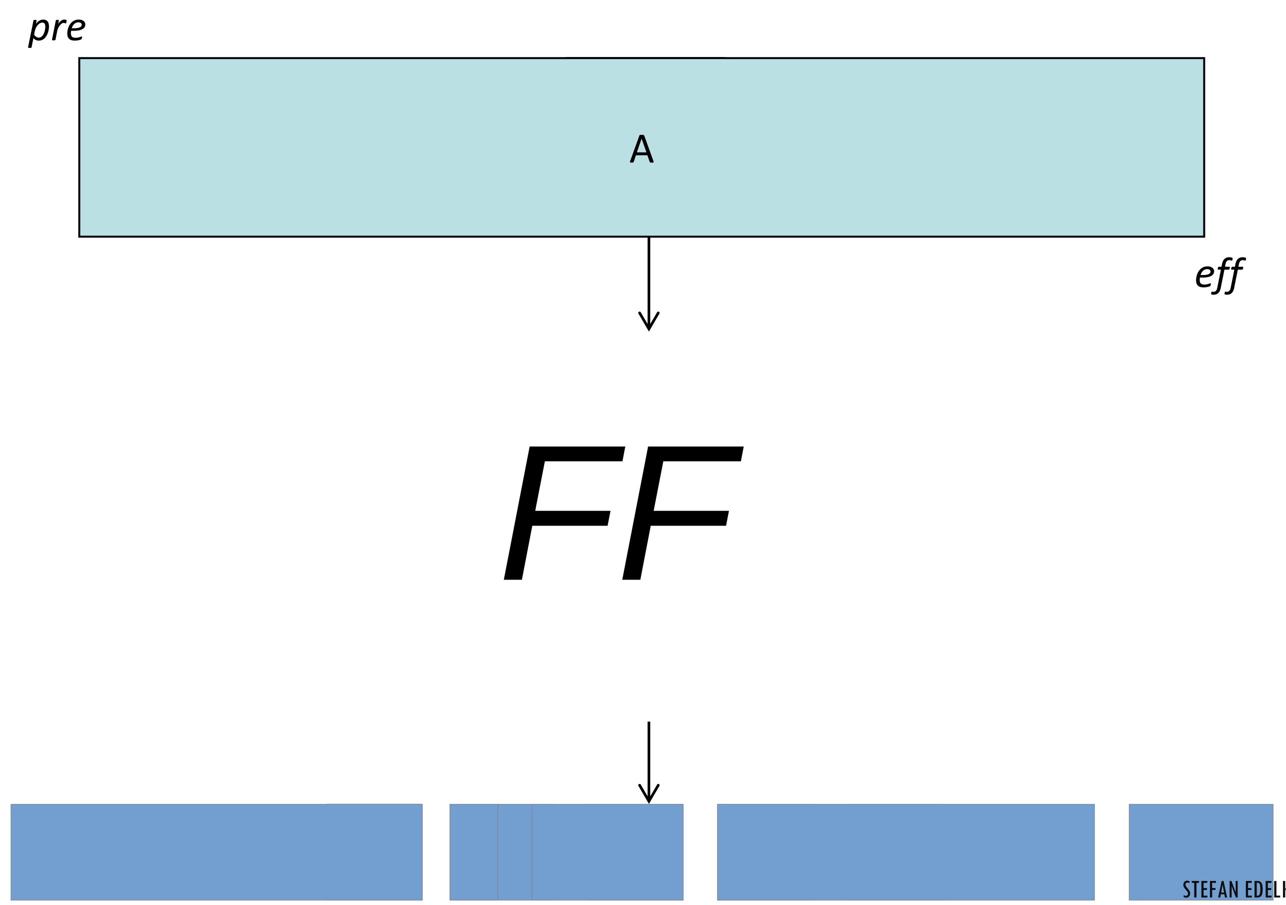
TIME

```
  0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c)    [30]
     (board ernie plane city-c)  [30]
     (refuel plane city-c)       [40]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a)   [20]
     (board scott plane city-a)  [30]
     (refuel plane city-a)       [40]
280: (fly plane city-a city-c)  [150]
430: (fly plane city-c city-d)  [150]
580: (debark ernie plane city-d) [20]
     (debark scott plane city-d) [20]
```

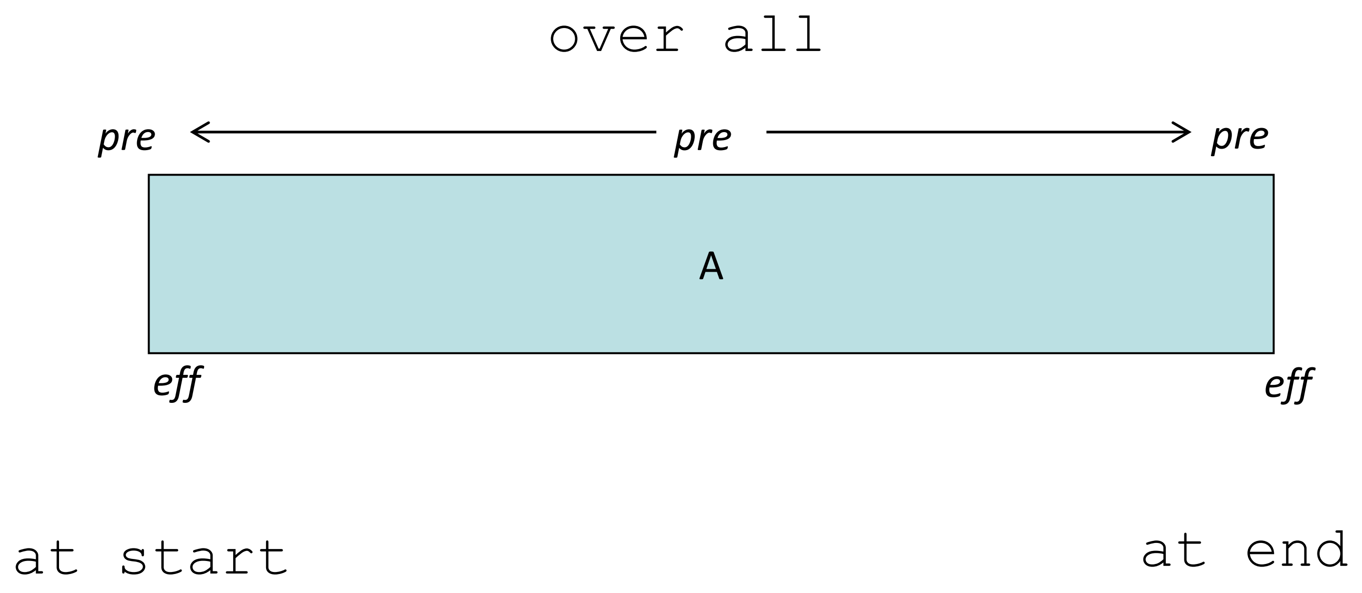
DURATIVE ACTIONS?



DURATIVE ACTIONS?



DURATIVE ACTIONS IN PDDL 2.1



PDDL EXAMPLE (I)

```
(           ·action LOAD-TRUCK
  :parameters
 (?obj - obj ?truck - truck ?loc - location)

 :precondition
   (and                (at ?truck ?loc))
                       (at ?obj ?loc)))
  :effect
   (and              (not (at ?obj ?loc)))
                     (in ?obj ?truck))))
```

Beware of self-overlapping actions!

PDDL EXAMPLE (II)

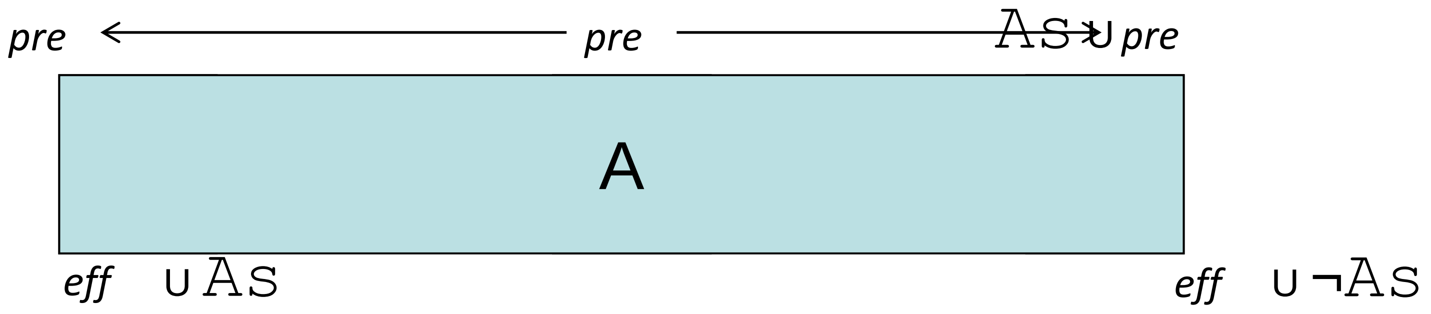
```
(:durative-action open-barrier
   :parameters
  (?loc - location ?p - person)
   :duration (= ?duration 1)
   :condition
     (and   (at start (at ?loc ?p)))
   :effect
     (and (at start (door-open ?loc))
          (at end (not (door-open ?loc))))
```

PDDL EXAMPLE (II)

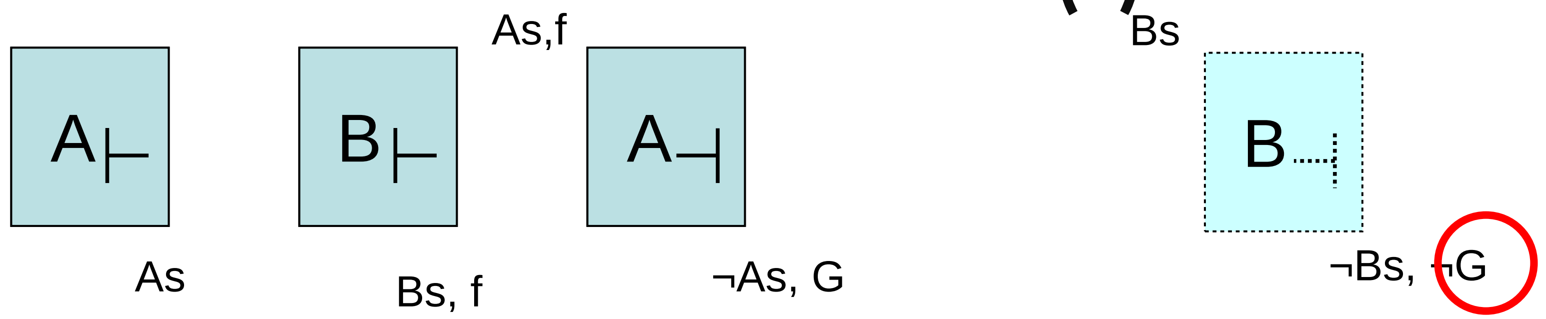
```
(:durative-action ope
  :parameters
 (?loc - location ?p
  :duration (= ?durat
  :condition
   (and   (at start (
  :effect
   (and (at start (do
        (at end (not (door-open ?loc))))
```

DURATIVE ACTIONS

(Fox and Long, ICAPS 2003)



PLANNING WITH SNAP ACTIONS (I)

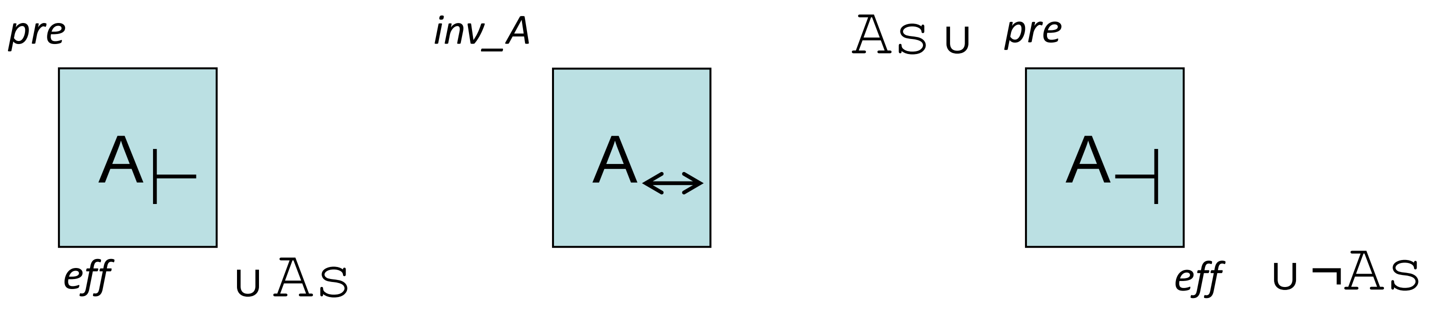


Challenge 1: What if B⊣ interferes with the goal?

@PDDL 2.1 semantics: **no actions can be executing in a goal state.**

Solution: add ¬As, ¬Bs, ¬Cs…. to the goal (or make this implicit in a temporal planner.)

PLANNING WITH SNAP ACTIONS (II)



Challenge 2: what about **over all** conditions?

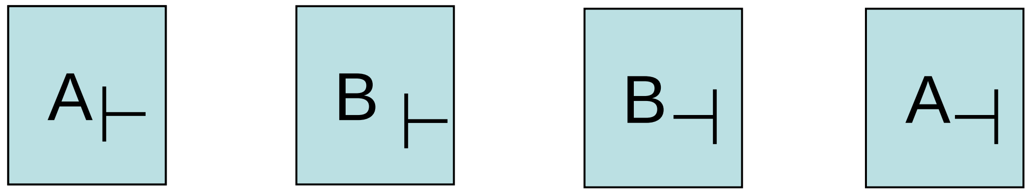
If A is executing, inv_A must hold.

Solution:
In every state where As is true: inv_A must also be true

Or: `(imply (As) inv_A)`

Violating an invariant then leads to a **dead-end**.

PLANNING WITH SNAP ACTIONS (III)



- Challenge 3: **where did the durations go?**
  - More generally, what are the temporal constraints?
  - **Logically sound ≠ temporally sound.**

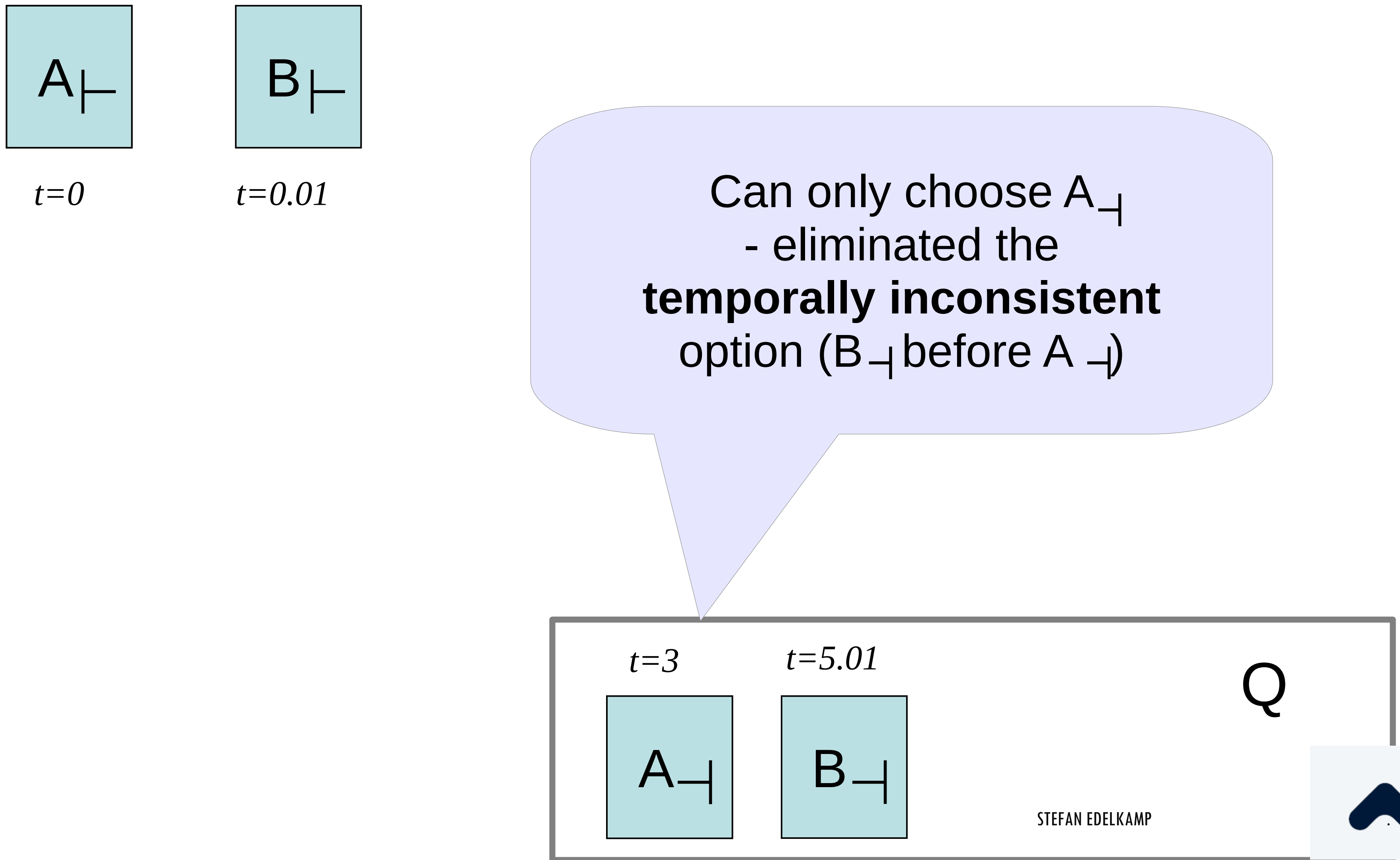
OPTION 1: DECISION EPOCH PLANNING

Term from Cushing et al, IJCAI 2007

- Search with **time-stamped states** and a **priority queue** of pending end snap-actions.
  - See Temporal Fast Downward (Eyerich, Mattmüller and Röger, ICAPS 2009); Sapa (Do and Kambhampati, JAIR 2003), and others.

- In a state S, at time t and with queue Q, either:
  - Apply a start snap-action $A_{\vdash}$ (at time t)
    - Insert $A_{\dashv}$ into Q at time $(t + dur(A))$
    - $S'.t = S.t + \varepsilon$
  - Remove and apply the first end snap-action from Q.
    - $S'.t$ set to the scheduled time of this, plus ε

RUNNING THROUGH OUR EXAMPLE...



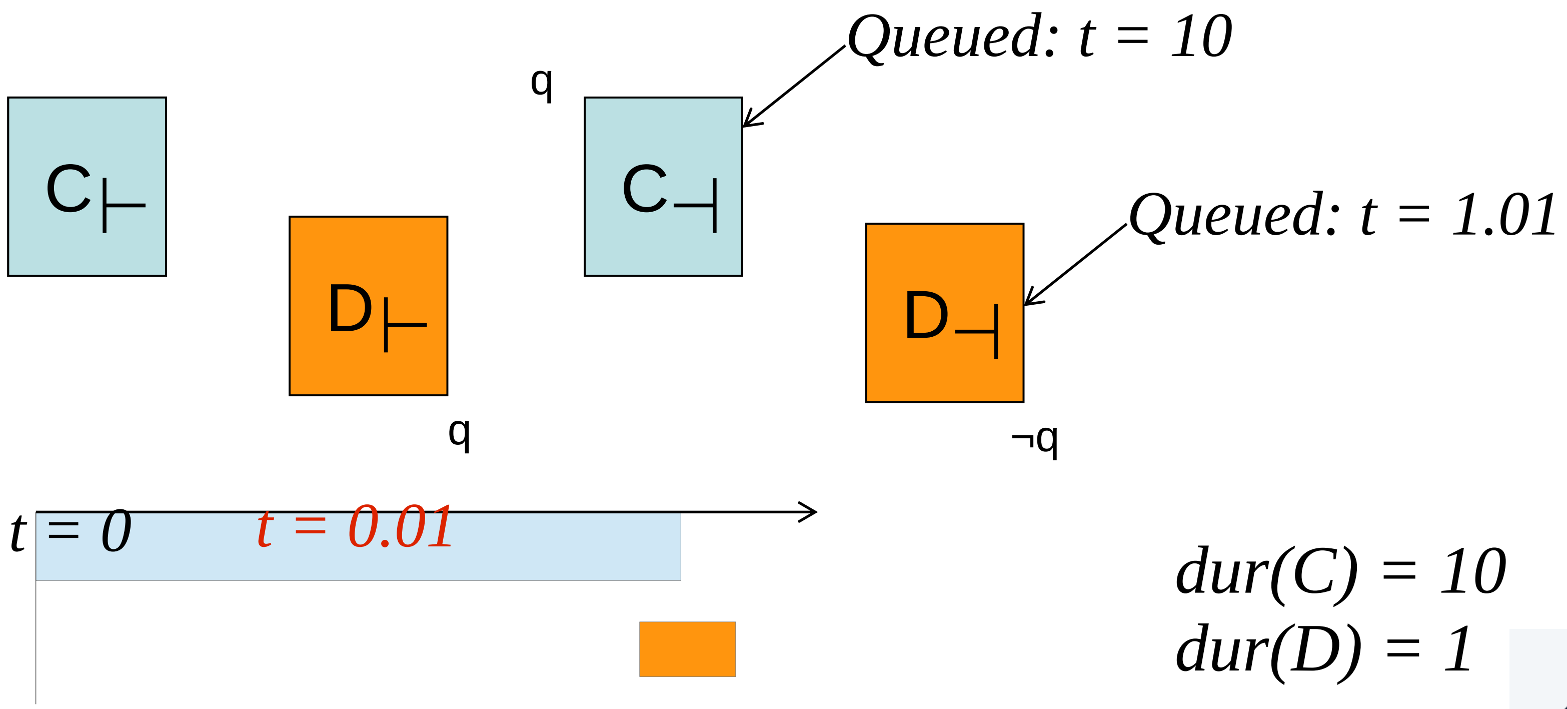
Can only choose $A_{\dashv}$ - eliminated the **temporally inconsistent** option ($B_{\dashv}$ before $A_{\dashv}$)

DECISION EPOCH PLANNING: THE SNAG

Must **fix start- and end-timestamps** at the point when the action is started.

- Used for the priority queue

Can we always do this?



$dur(C) = 10$
$dur(D) = 1$

OPTION 2: SIMPLE TEMPORAL NETWORKS

A SIMPLE TEMPORAL PROBLEM?

- All our constraints are of the form:
- $\varepsilon \le t(i+1) - t(i)$ *(c.f. sequence constraints)*
- $\mathrm{dur}_{min}(A) \le t(A_{\dashv}) - t(A_{\vdash}) \le \mathrm{dur}_{max}(A)$
- Or, more generally, $lb \le t(j) - t(i) \le ub$

- Is a **Simple Temporal Problem**
- "Temporal Constraint Networks", Dechter, Meiri and Pearl, AIJ, 1991

- Good news – is **polynomial**
- Bad news – in planning, we need to solve it a lot....

EXAMPLE

John travels to work either by car (30-40 min) or by bus (>= 60 min)

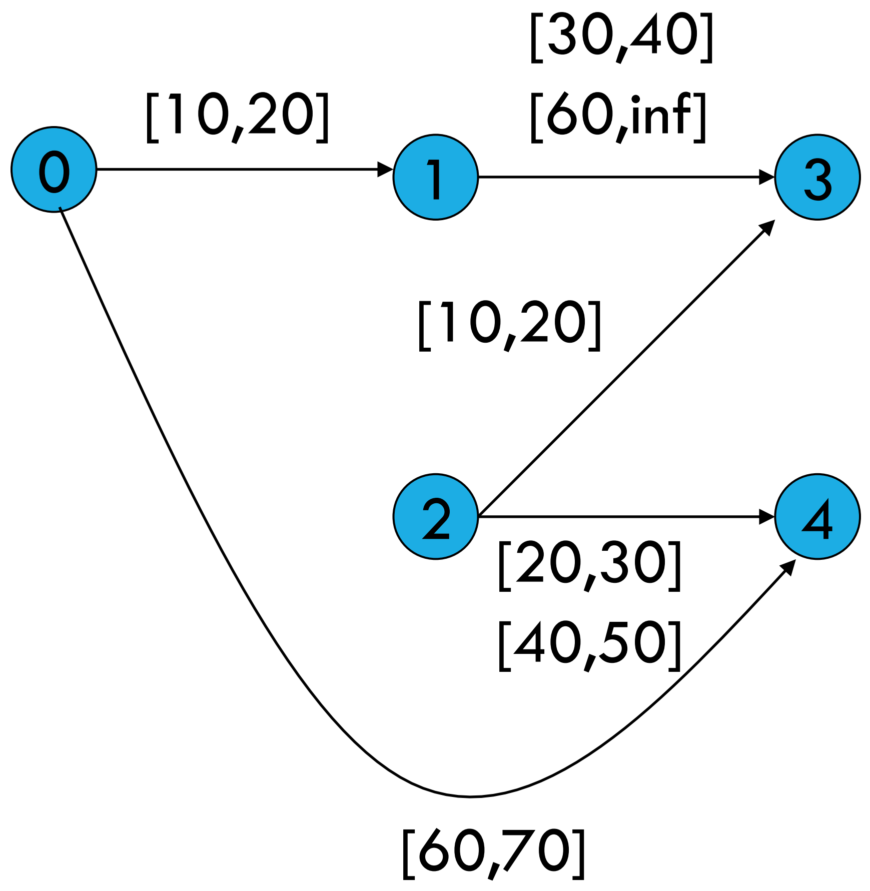
Fred travels to work either by car (20-30 min) or in a carpool (40-50 min)

Today John left between 7:10 and 7:30am.

Fred arrived at work between 8:00 and 8:10am.

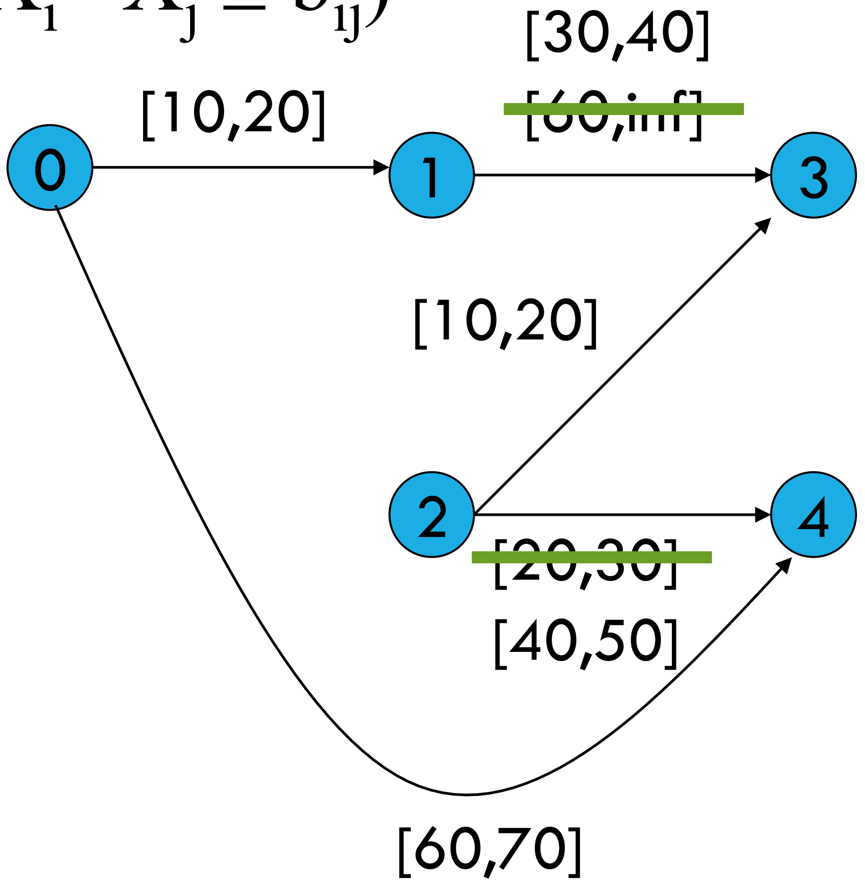
John arrived at work 10-20min after Fred left home.

VISUALIZE TCSP AS DIRECTED CONSTRAINT GRAPH



SIMPLE TEMPORAL NETWORK

- $T_{ij} = (a_{ij} \leq X_i - X_j \leq b_{ij})$



Simple Temporal Network:

A set of time points X_i at which events occur.

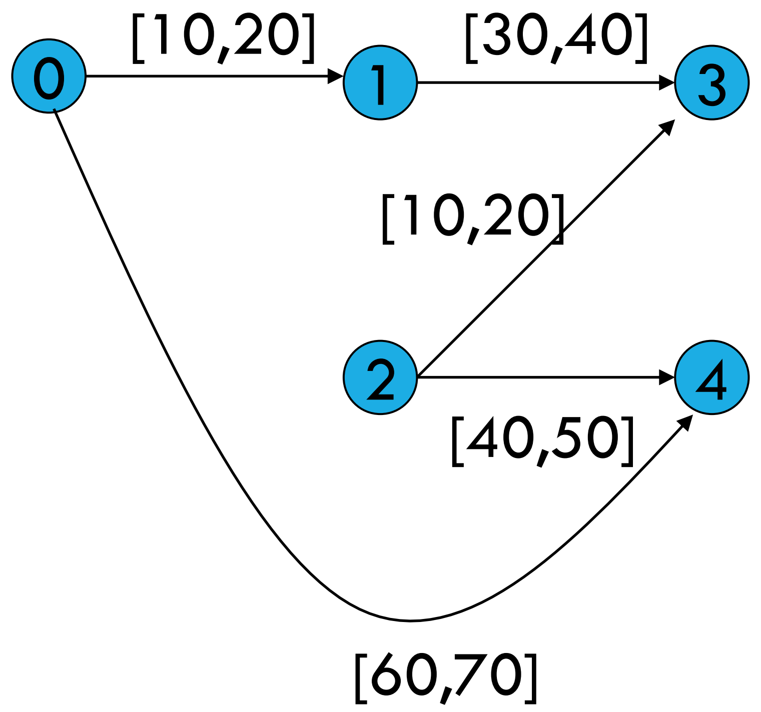
Unary constraints

$(a_0 \le X_i \le b_0)$ ~~or $(a_1 \le X_i \le b_1)$ or . .~~ .

Binary constraints

$(a_0 \le X_j - X_i \le b_0)$ ~~or $(a_1 \le X_j - X_i \le b_1)$ or~~ . . .

STN



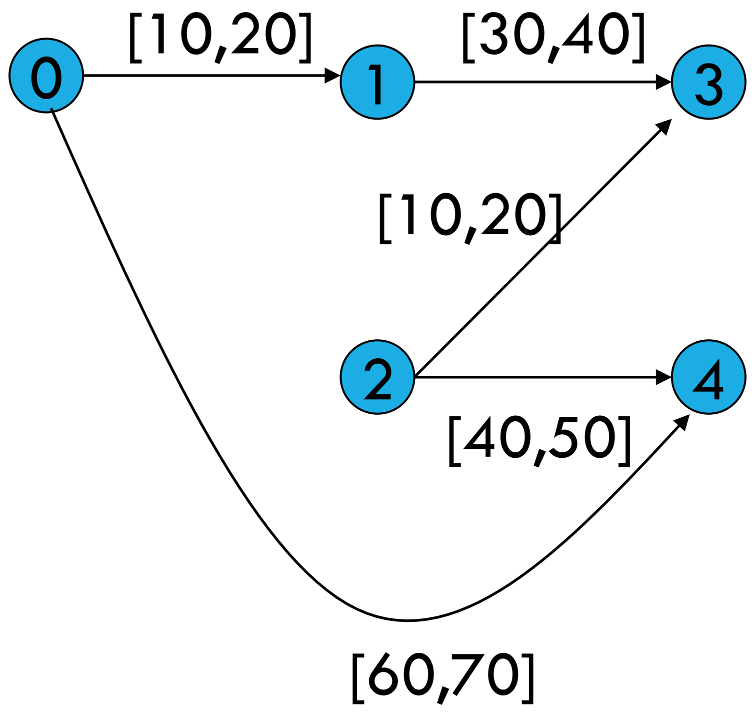
Shostak (1981) A simple temporal problem is consistent if and only if the distance graph has no cycles.
→ The consistency and the minimal network of an STP can be determined in cubic time using all-pairs shortest path search.

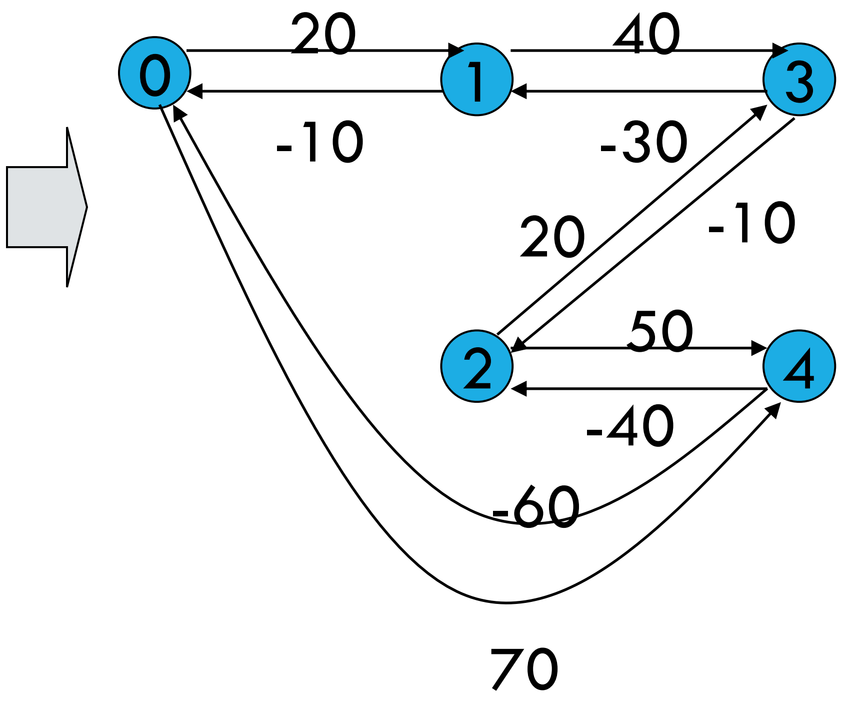
TO QUERY STN MAP TO DISTANCE GRAPH G_D

Edge encodes an upper bound on distance to target from source.

$$T_{ij} = (a_{ij} \leq X_j - X_i \leq b_{ij})$$

$$X_j - X_i \leq b_{ij}$$
$$X_i - X_j \leq - a_{ij}$$



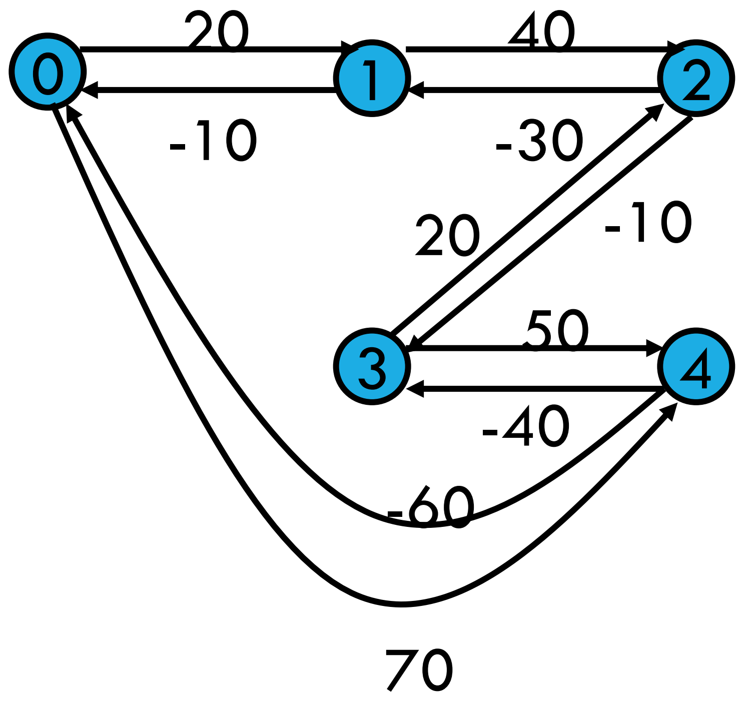


AI CENTER FEE CTU

SHORTEST PATHS OF G_D

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph



STN MINIMUM NETWORK

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph

	0	1	2	3	4
0	[0]	[10,20]	[40,50]	[20,30]	[60,70]
1	[-20,-10]	[0]	[30,40]	[10,20]	[50,60]
2	[-50,-40]	[-40,-30]	[0]	[-20,-10]	[20,30]
3	[-30,-20]	[-20,-10]	[10,20]	[0]	[40,50]
4	[-70,-60]	[-60,-50]	[-30,-20]	[-50,-40]	[0]

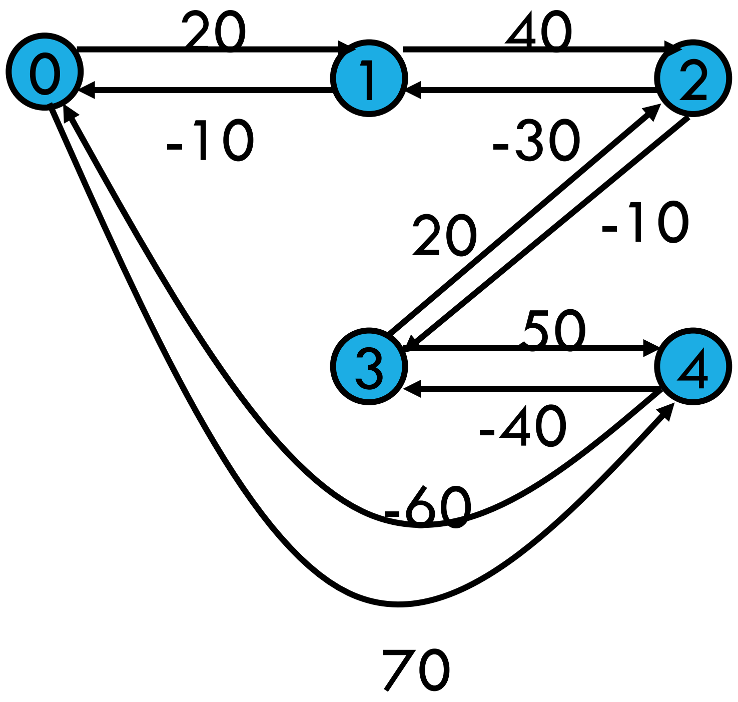
STN minimum network

AI CENTER FEE CTU

TEST CONSISTENCY: NO NEGATIVE CYCLES

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph

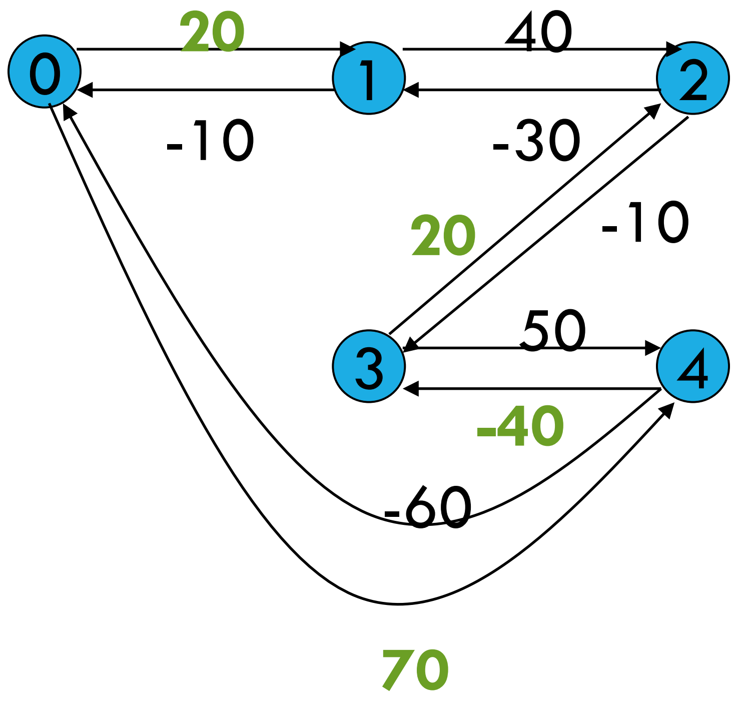


LATEST SOLUTION

Node 0 is the reference.

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph

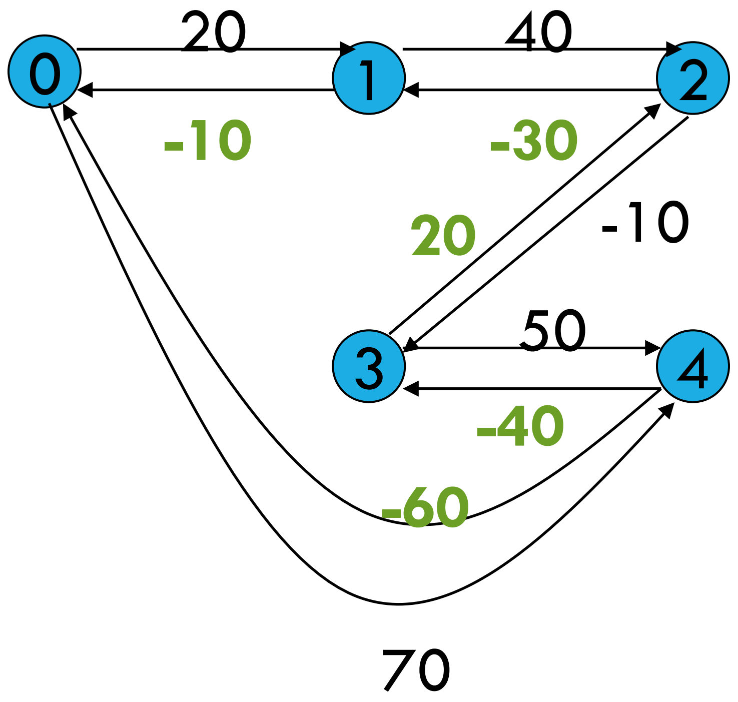


EARLIEST SOLUTION

Node 0 is the reference.

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph



FEASIBLE VALUES

	0	1	2	3	4
0	0	20	50	30	70
1	-10	0	40	20	60
2	-40	-30	0	-10	30
3	-20	-10	20	0	50
4	-60	-50	-20	-40	0

d-graph

- X_1 in [10, 20]
- X_2 in [40, 50]
- X_3 in [20, 30]
- X_4 in [60, 70]

BACK TO PLANNING: LATEST POSSIBLE TIMES? (MAXIMUM SEPARATION)

$t(A) - t(Z) <= 4$
$t(B) - t(Z) <= 8$
$t(C) - t(Z) <= 10$

('A comes no more than 4 time units after Z')

LATEST POSSIBLE TIMES?

t(A) – t(Z) <= 4
t(B) – t(Z) <= 8
t(C) – t(Z) <= 10

t(B) – t(A) <= 2
t(C) – t(B) <= 1

('B comes no more than 2 time units after A')

EARLIEST POSSIBLE TIMES? (MINIMUM SEPARATION)

- For latest possible time: find the **shortest path**
- For earliest possible times…?

EARLIEST POSSIBLE TIMES?

$2 <= t(A) – t(Z)$
$4 <= t(B) – t(Z)$

$3 <= t(B) – t(A)$
$1<= t(C) – t(B)$

HACKING ALGORITHMS

- Longest path from Z to C?
- = Shortest **negative** path from C to Z

$$2 <= t(A) – t(Z)$$

Multiply both sides by -1:

$$-2 > - t(A) + t(Z)$$

p >= q is the same as q <= p:

$$- t(A) + t(Z) < -2$$

Rearrange LHS:

$$t(Z) – t(A) < -2$$

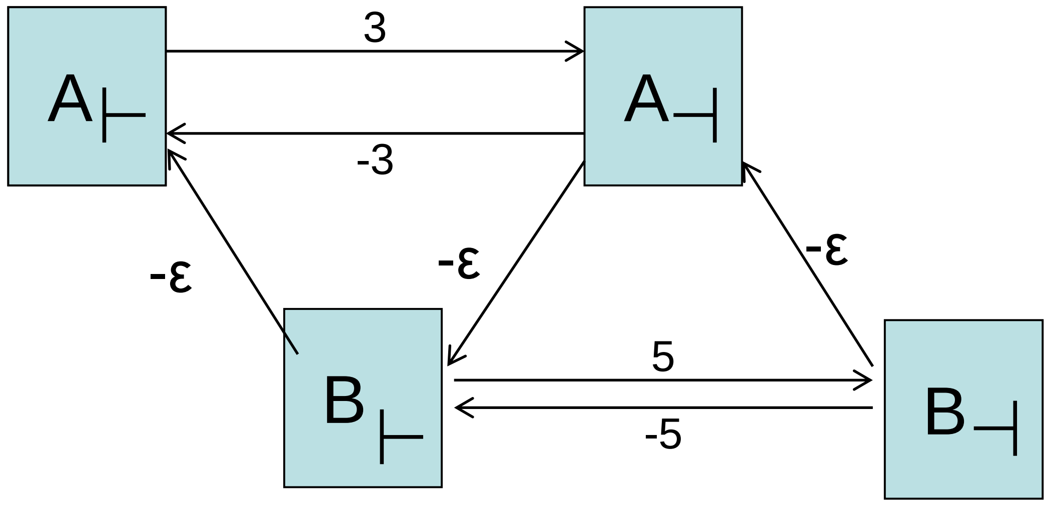
EARLIEST POSSIBLE TIMES?

$2 <= t(A) – t(Z)$	$-2 >= – t(A) + t(Z)$	$t(Z) – t(A) <= -2$
$4 <= t(B) – t(Z)$	$-4 >= – t(B) + t(Z)$	$t(Z) – t(B) <= -4$
$3 <= t(B) – t(A)$	$-3 >= – t(B) + t(A)$	$t(A) – t(B) <= -3$
$1<= t(C) – t(B)$	$-1 >= – t(C) + t(B)$	$t(B) – t(C) <= -1$

SIMPLE TEMPORAL NETWORKS (I)

- Can map STPs to an equivalent digraph:
  - One vertex per time-point (and one for 'time zero');
  - For $lb \leq t(j) - t(i) \leq ub$:
  - An edge (i $\rightarrow$ j) with weight *ub*.
  - An edge (j $\rightarrow$ i), with weight *-lb*
  - (c.f. $lb \leq t(j) - t(i) \quad \rightarrow \quad t(j) - t(i) \leq -lb$)

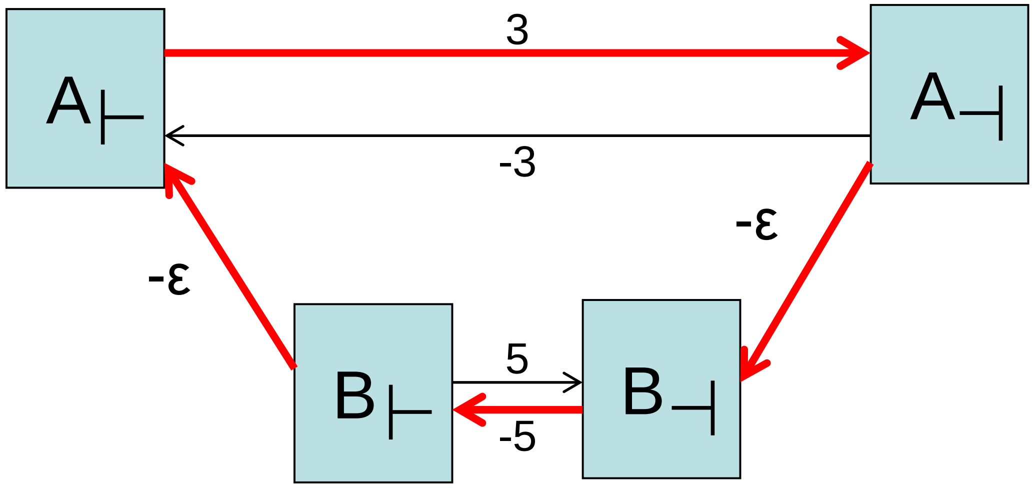
EXAMPLE STN



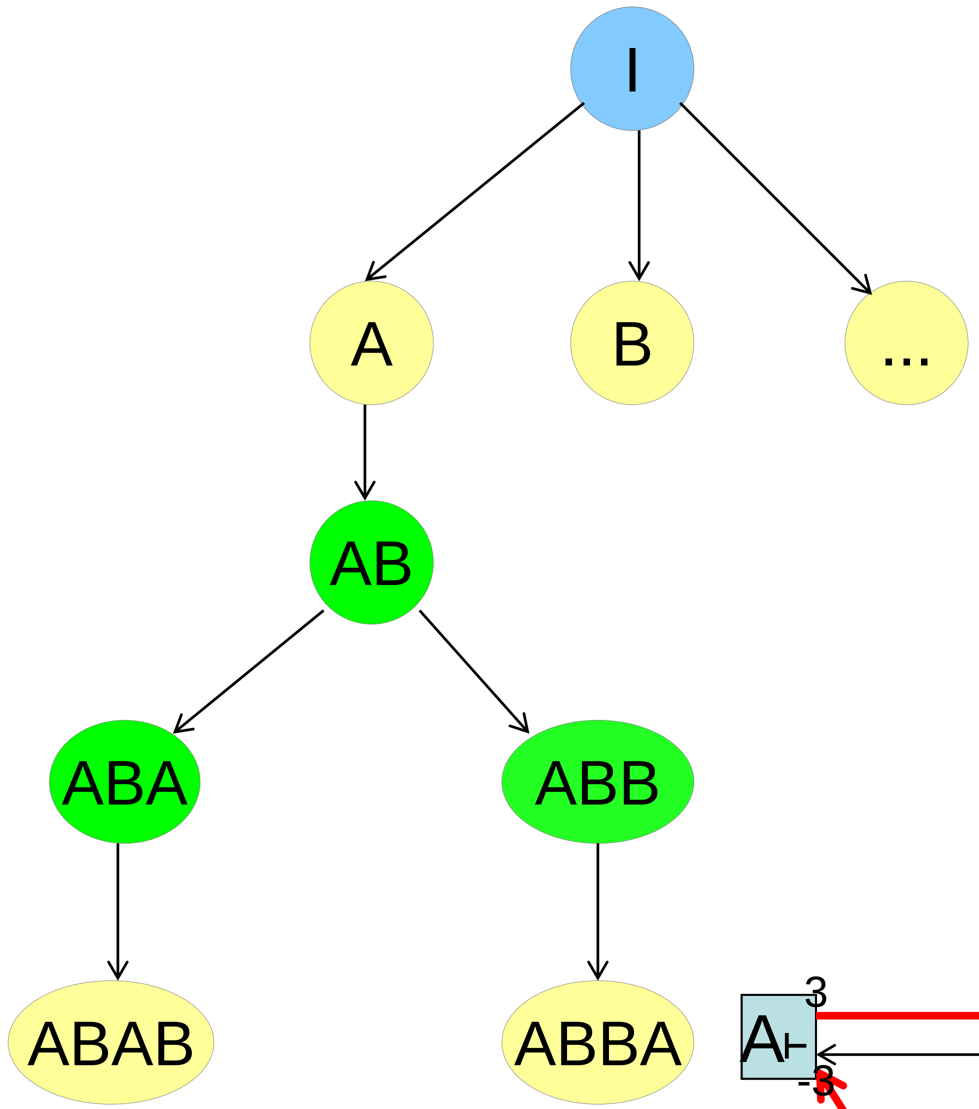
```
0.00: (A) [3]
0.01: (B) [5]
```

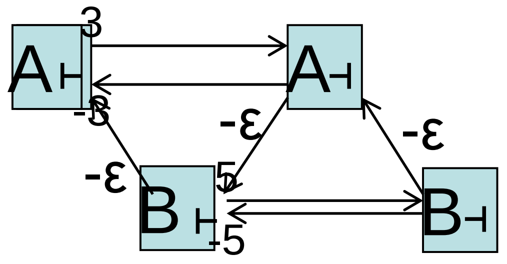
SIMPLE TEMPORAL NETWORKS (II)

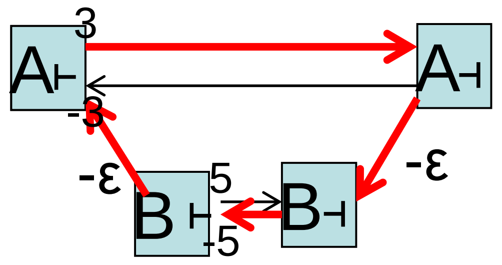
- Solve the shortest path problem (e.g. using Bellman-Ford) from/to zero
  - dist(0,j)=x → maximum timestamp of j = x
  - dist(j,0)=y → minimum timestamp of j = -y
- If we find a **negative cycle** then the temporal constraints are inconsistent:



AI CENTER FEE CTU







AI CENTER
FEE CTU

PUBLIC TRANSPORT EXAMPLE

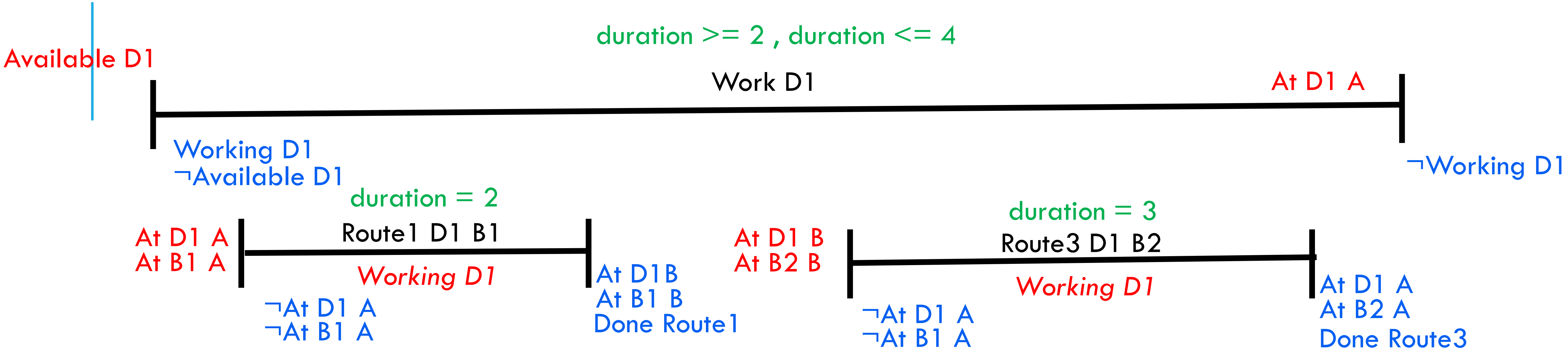
Drivers have working hours;

Bus routes have fixed durations and start and end locations.

Goals are that each bus route is done.

The routes have timetables that they must follow.

TEMPORAL PLANNING: PUBLIC TRANSPORT



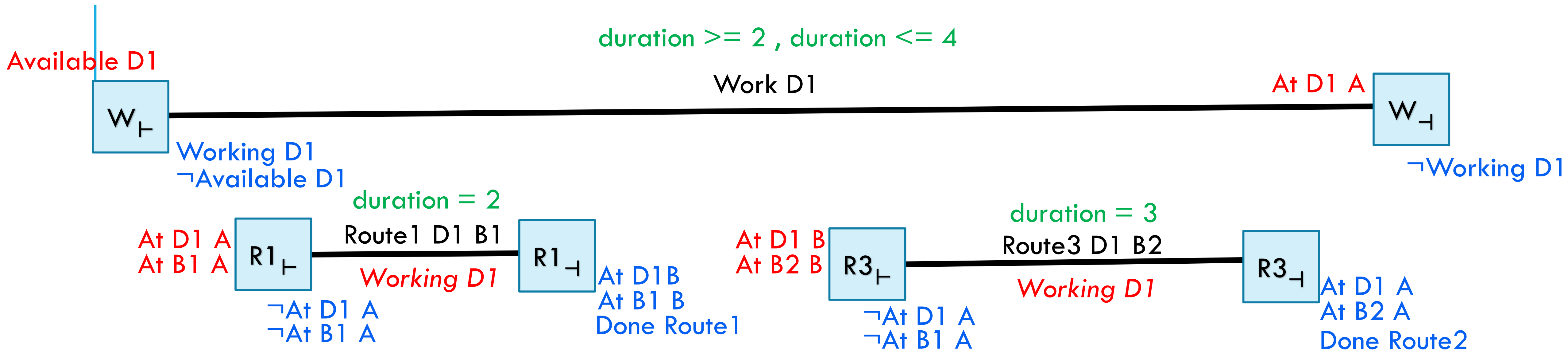
Actions have:

- Conditions and Effects at the start and at the end;
- Invariant/overall conditions;
- Durations constraints:

  (= ?duration 4)

  (and (>= ?duration 2) (<= ?duration 4))

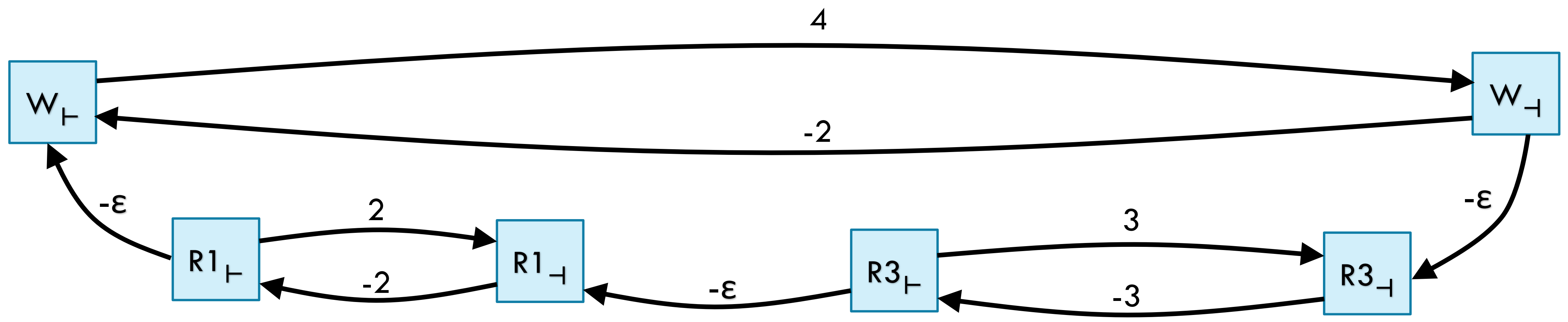
PLANNING WITH SNAP ACTIONS



Three Challenges:

- Make sure ends can't be applied unless starts have.
- Overall Conditions.
- Duration constraints.

PLANNING WITH SNAP ACTIONS AND STNS



Constraints:

$W_{\dashv} - W_{\vdash} >= 2$

$W_{\dashv} - W_{\vdash} <= 4$

$R1_{\vdash} >= W_{\vdash} + \varepsilon$

$R1_{\dashv} - R1_{\vdash} = 2$

$R3_{\vdash} >= R1_{\vdash} + \varepsilon$

$R3_{\dashv} - R3_{\vdash} = 3$

$W_{\dashv} >= R3_{\dashv} + \varepsilon$

TIMED INITIAL LITERALS

Introduced in PDDL 2.2 (IPC 2004);

Allow us to model facts that become true, or false, at a specific time.

Can use them to model deadlines or time windows.

Cannot be done directly, but we can achieve this by adding more facts to the domain.

MODELLING DEADLINES USING TILS

Make sure the action achieving the desired fact has a condition that ensures it takes place before the deadline (over all or at start/end).

Make that fact true in the initial state.

And a TIL to delete it at the deadline.

Note that we could have multiple deadlines for different objects.

```
(:durative-action unload-truck
  :parameters (?p - obj ?t- truck ?l- location)
  :duration (= ?duration 2)
  :condition  (and (over all (at ?t ?l))
                   (at start (in ?p?t)))
                   (at end (can-deliver ?p)))
  :effect   (and (at start (not (in ?p ?t)))
                 (at end (at ?p ?l))))
Init:
(can-deliver package1)
(at 9 (not (can-deliver package1)))
(can-deliver package2)
(at 11 (not (can-deliver package2)))
```

REASONING WITH TILS IN FORWARD SEARCH

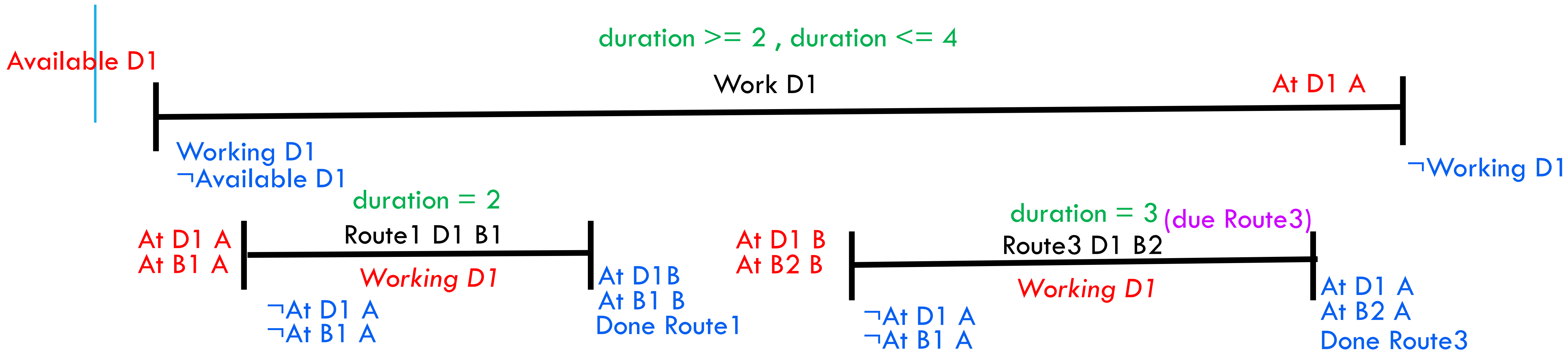
Order the TILs chronologically;

At each state we have a choice:

- Apply an action that is applicable in that state;
- Apply the next Available TIL.

This allows us to leave the choice to search about whether the TIL will appear before or after a given action.

TEMPORAL PLANNING: PUBLIC TRANSPORT

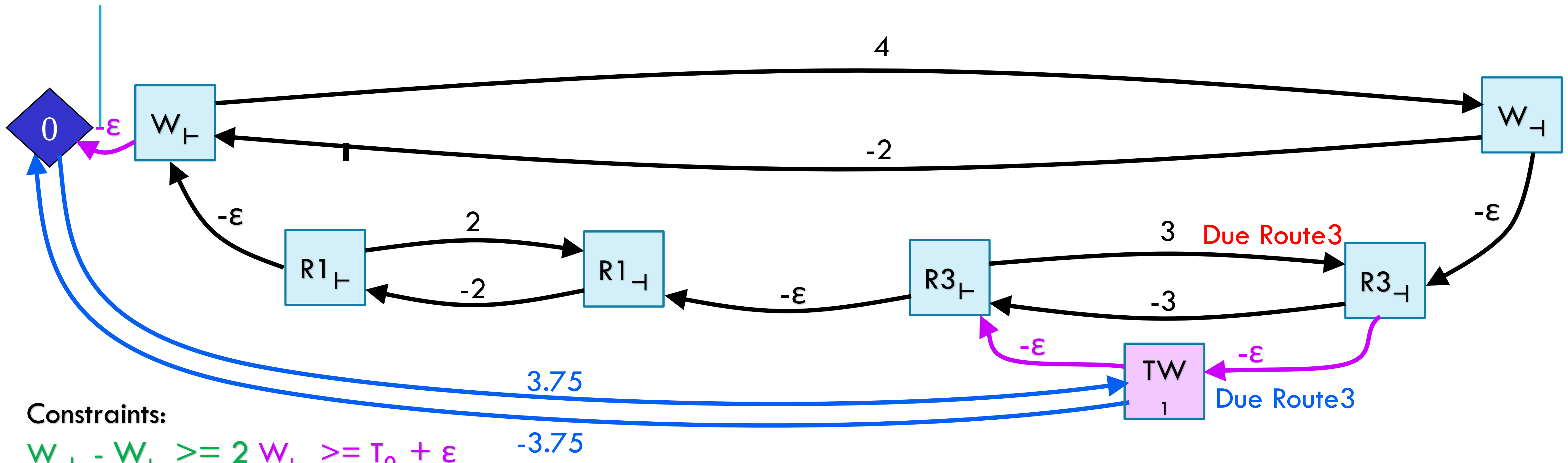


Actions have:

- Conditions and Effects at the start and at the end;
- Invariant/overall conditions;
- Durations constraints:
  (= ?duration 4)
  (and (>= ?duration 2) (<= ?duration 4))
- We can also have windows of opportunity: (at 3.75 (due Route3)) (at 4 (not (due Route3)))

AI CENTER FEE CTU

PLANNING WITH TIME WINDOWS



Constraints:

$W_\dashv - W_\vdash >= 2$ $W_\vdash >= T_0 + \varepsilon$

$W_\dashv - W_\vdash <= 4$ $TW_1 = T_0 + 3.75$

$R1_\vdash >= W_\vdash + \varepsilon$ $TW1_1 >= R3_\vdash +$

$R1_\dashv - R1_\vdash = 2$ ε

$R3_\vdash >= R1_\vdash + \varepsilon$ $R3_\dashv >= TW_1 + \varepsilon$

$R3_\dashv - R3_\vdash = 3$

$W_\dashv >= R3_\dashv + \varepsilon$

Planning with Preferences

- Find: **A** sequence of actions: any plan will do?
  - Even the shortest might not be necessarily the best.
  - We might care about *how* the goal is achieved.

- What if we can't reach the goal:
  - Report 'No Plan Exists'.
  - Search Indefinitely.
  - Maybe we could satisfy some of it?

Link to Numerical Metric Planning

Given:

- An initial state: a set of propositions and assignments to numeric variables, `e.g. (at rover waypoint1) (= (energy rover) 10).`
- A goal: a desired set of propositions/assignments, `e.g. (at rover waypoint4) (have-soil-sample waypoint3).`
- A set of actions each with:
  - Preconditions on execution;
  - Effects that describe how the world changes upon their execution.

```
(:action navigate
:parameters
(?r - rover ?y - waypoint ?z - waypoint)
:precondition (and
             (available ?r)
             (at ?r ?y)
                          (visible ?y ?z)
             (>= (energy ?r) 8))
:effect (and
             (decrease (energy ?r) 8)
             (not (at ?x ?y))
             (at ?x ?z)))
```

Find:

- A sequence of actions that when applied in the initial state leads to a state that satisfies the goal condition.

Planning with **Preferences**

- Simple Preferences (*soft goals and preconditions):*
  - *(p0 (at end (at rover waypoint3)))*
- *Trajectory preferences (Conditions on the plan):*
  - *(p1 (always (>= (energy rover) 2)*
  - *(p2 (sometime (at driver costa-coffee))*
  - *(p3 (at-most-once (at truck Birmingham)))*
  - *(p4 (sometime-after (at Birmingham) (at Glasgow)))*
  - *(p5 (sometime-before (at Birmingham) (had-lunch)))*
- *Temporal Preferences (not covered).*
- *Metric Function:*
- *(minimze (+ (fuel-used) (*2 (is-violated p0)) (*5 (is-violated p1)))*

EXAMPLES

Blocksworld:

- a fragile block can never have something above it, or
- it can have at most one block on it;
- we would like that the blocks forming the same tower always have the same colour;
- in some state of the plan, all blocks should be on the table.

Transportation:

- we would like that every airplane is used (instead of using only a few airplanes, because it is better to distribute the workload among the available resources and limit heavy usage);
- whenever a ship is ready at a port to load the containers it has to transport, all such containers should be ready at that port;
- we would like that at the end of the plan all trucks are clean and
- at their source location ;we would like no truck to visit any destination more than once

MODALITIES

(:constraints ...)

plus

:constraints flag in the: requirements list.

BNF

<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> |(hold-after <num> <GD> | ...

GOALS FOR TODAY

- Get to work
- If I do that, get a coffee on the way
- If I get a coffee, go to the loo after

GOALS FOR TODAY

- Get to work
- If I do that, get a coffee on the way
- If I get a coffee, go to the loo after
- Play the piano

AI CENTER FEE CTU

AI CENTER
FEE CTU

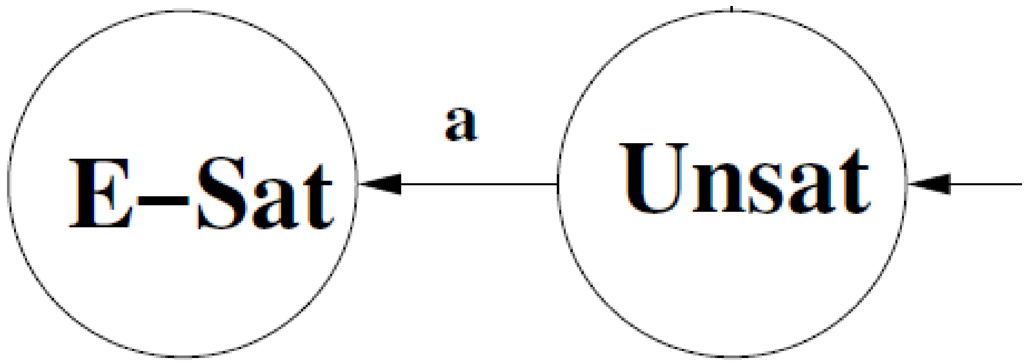
GOALS FOR *THE END* OF TODAY?

```
(:goals  (and
                    (at stefan work)
          )
)
```

WORK/LIFE BALANCE

(sometime (at stefan work))

(sometime (at stefan piano))



(sometime a)

SELLING OUT...

(preference p0 (sometime (at stefan work)))

(preference p1 (sometime (at stefan piano)))

$cost(p0) = 100$

$cost(p1) = 5$

WITH GOAL(S)

```
(:goal (and
        (at stefan mybed)
))

(:constraints (and
    (preference p0 (sometime (at stefan work)))
    (preference p1 (sometime (at stefan piano)))
))
```

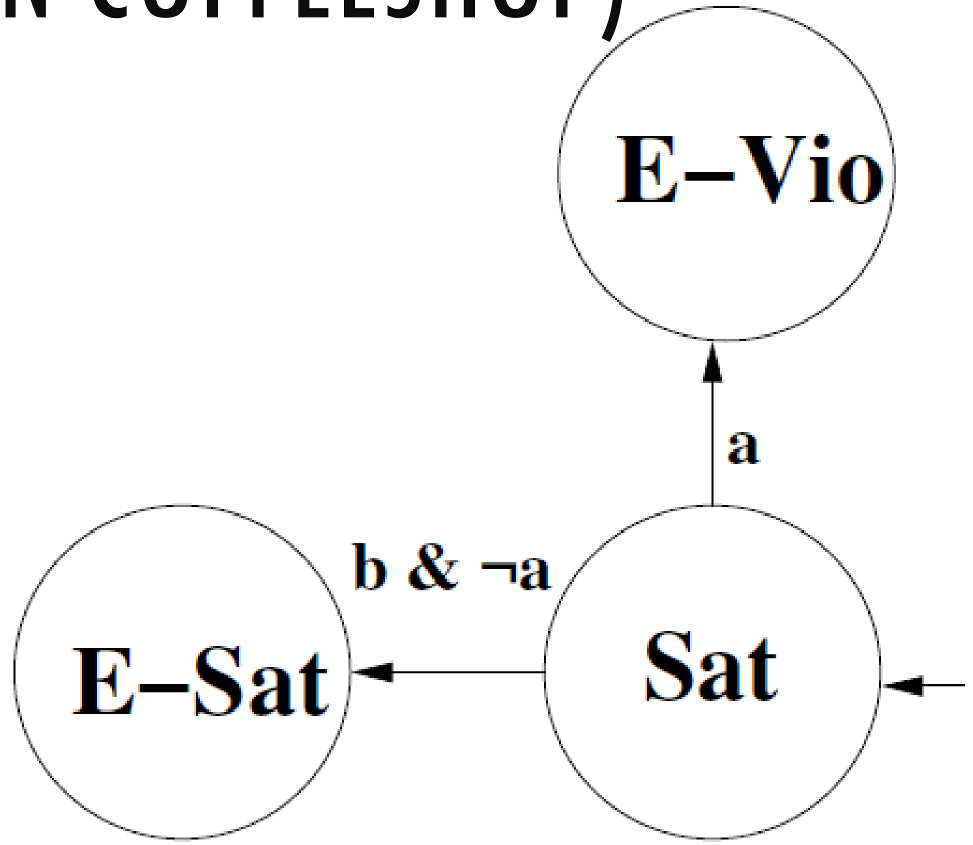
IF I DO THAT, GET A COFFEE ON THE WAY

```
(sometime (at stefan coffeeshop))  ?


(sometime-before (at stefan work)
                 (at stefan coffeeshop))
```

AI CENTER
FEE CTU

A = (AT STEFAN WORK)
B = (AT STEFAN COFFEESHOP)



`(sometime-before a b)`

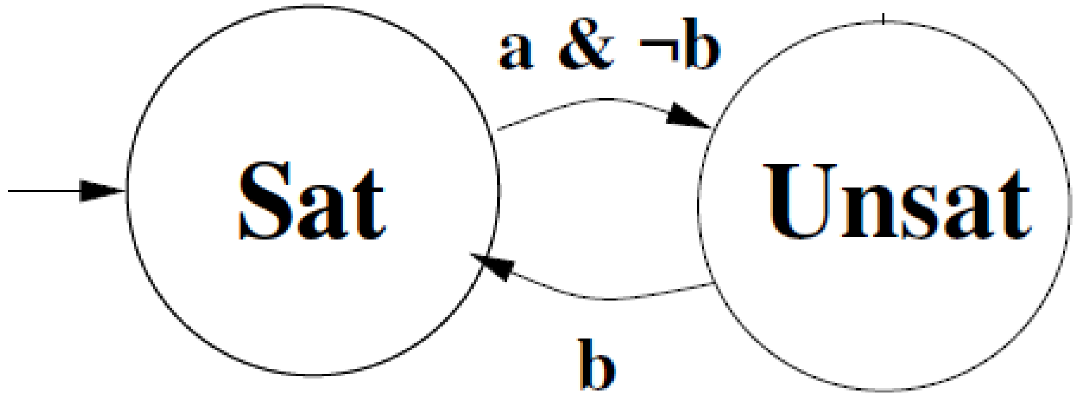
IF I GET A COFFEE, GO TO THE LOO

(sometime (at stefan loo)) ???

IF I GET A COFFEE, GO TO THE LOO

```
(sometime-before (at stefan work)
                              (at stefan coffeeshop))
(sometime-after (at stefan coffeeshop)
                           (at stefan loo))
```

A = (AT STEFAN COFFEESHOP)
B = (AT STEFAN LOO)



(sometime-after a b)

WATCH OUT!

What if playing the piano adds (played stefan piano):

```
        (sometime-after (at stefan work)
                        (played stefan piano))
```

Order a then b then c?

```
        (and    (sometime-before (b) (a))
                (sometime-after (b) (c))
```

We can satisfy this with b, c, a, b not just a, b, c!

ONE LAST SORT OF PREFERENCE

```
(:goal (and
    (at stefan mybed)
    (preference p4 (switched-off phone))
))
```

Summary

- It's not just about where you get to but how you get there;
- We can't always achieve everything, but want to achieve all we can;
- Sophisticated domain-independent planners exist to deal with these concerns for us:
  - Much of this success is down to heuristics;
- Treat summary stats with care!