Temporal Networks: STNs, STNUs, CSTNs and CSTNUs

Luke Hunsberger

Computer Science Department, Vassar College, Poughkeepsie, NY USA

November 2018

An earlier version of this presentation "Temporal Networks for Dynamic Scheduling" was given by Luke Hunsberger at the *1st Summer School on Cognitive Robotics,* MIT, 2017. Additonal material was added by Roberto Posenato and Luke Hunsberger

Outline

Simple Temporal Networks (STN)

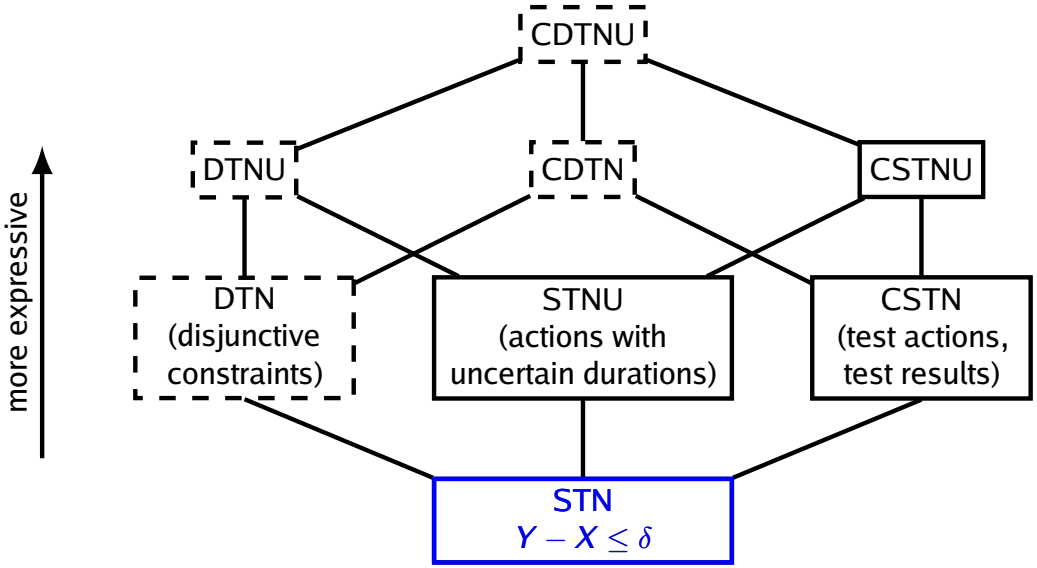
Simple Temporal Network
Motivating Example

- Given a plan of flying from New York to Rome and back.
- Determine if the plan is **consistent**.
- If it is consistent, determine a **temporal schedule**.

Fly from New York to Rome and back

- Leave New York after 4 p.m., June 8
- Return to New York before 10 p.m., June 18
- Away from New York no more than 7 days
- In Rome at least 5 days
- Return flight lasts no more than 7 hours

Simple Temporal Network (STN)*
Description



- STN = Time–points and
  simple temporal constraints.
- Flexible: Time–points may "float";
  not "nailed down" until they are *executed.*
- Efficient algorithms for determining consistency, managing real–time execution, and managing new constraints.

* [Dechter et al., 1991]

Simple Temporal Network*
Definition

Definition 1 (Simple Temporal Network)

A *Simple Temporal Network (STN)* is a pair, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, where:

- $\mathcal{T}$ is a set of real–valued variables called *time–points*; and
- $\mathcal{C}$ is a set of binary constraints, each of the form:

$$Y - X \leq \delta$$

where $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$.

* [Dechter et al., 1991]

Simple Temporal Network
The *Zero* Time–Point, Z

- A special time–point, Z, whose value is fixed at 0.
- Binary constraints involving Z are equivalent to unary constraints.

Example 1

$$X - Z \leq 7 \quad \Longleftrightarrow \quad X \leq 7$$

$$Z - X \leq -3 \quad \Longleftrightarrow \quad X \geq 3$$

- A *solution* to an STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ is a complete set of assignments to the time–points in $\mathcal{T}$:

$$\{X_1 = w_1,\ X_2 = w_2,\ \ldots,\ X_n = w_n\}$$

  that together satisfy all of the constraints in $\mathcal{C}$.

- An STN with at least one solution is *consistent.*
- STNs with identical solution sets are *equivalent*.

Simple Temporal Network
STN for Travel Example

$\mathcal{T} = \{Z, X_1, X_2, X_3, X_4\}$, where $Z =$ Noon, June 8

$$\mathcal{C} = \begin{cases} Z - X_1 \leq -4 & \text{(Leave NYC after 4 p.m., June 8)} \\ X_4 - Z \leq 250 & \text{(Return NYC by 10 p.m., June 18)} \\ X_4 - X_1 \leq 168 & \text{(Gone no more than 7 days)} \\ X_2 - X_3 \leq -120 & \text{(In Rome at least 5 days)} \\ X_4 - X_3 \leq 7 & \text{(Return flight less than 7 hrs)} \end{cases}$$

Simple Temporal Network
Graph for an STN*

The *graph* for an STN, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, is a graph, $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, where:

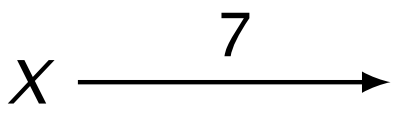
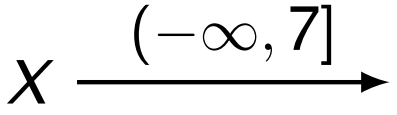
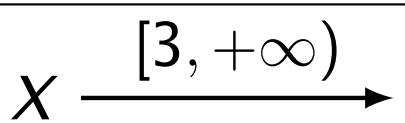
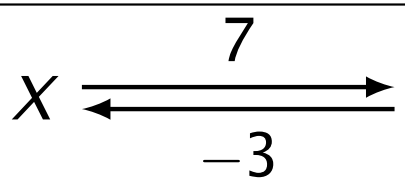
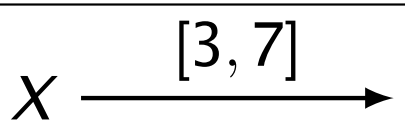
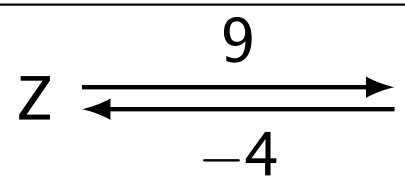
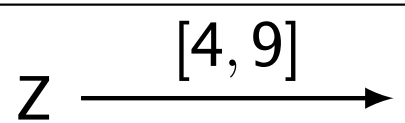
$$\text{Time–points in } \mathcal{S} \iff \text{nodes in } \mathcal{G}$$

$$\text{Constraints in } \mathcal{C} \iff \text{edges in } \mathcal{E}\text{:}$$

$$Y - X \leq \delta \qquad X \xrightarrow{\delta} Y$$

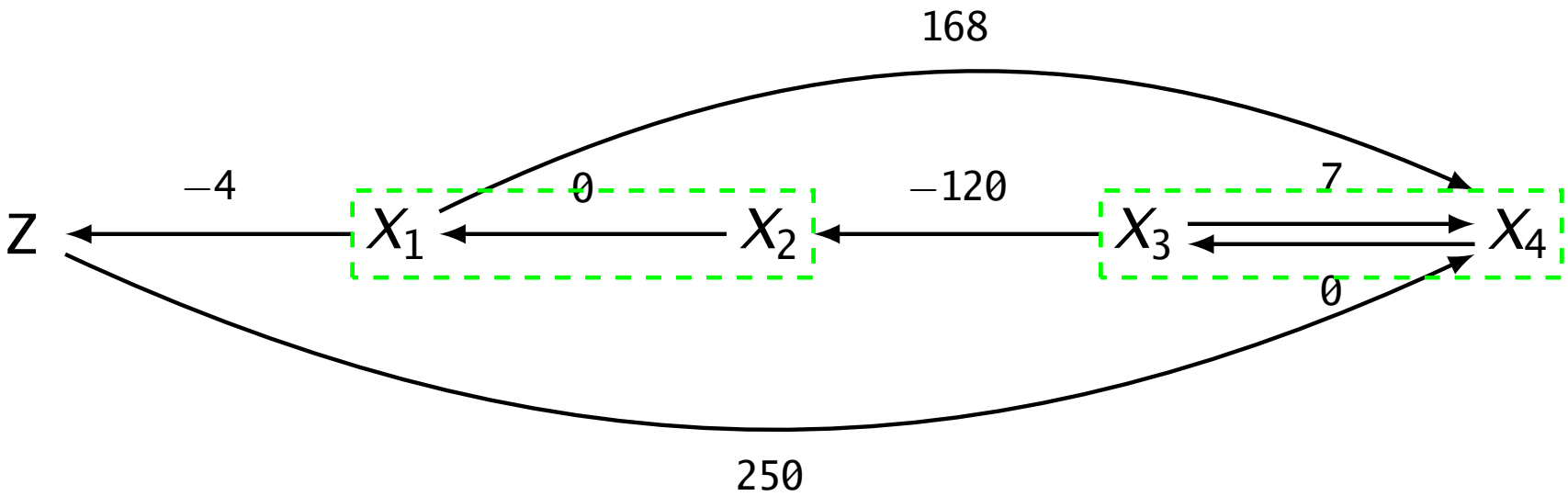
* [Dechter et al., 1991]

Simple Temporal Network
Graphical Representation

Constraint(s)	Edge(s)	Interval Notation
$Y - X \leq 7$		
$X - Y \leq -3$ (equiv: $Y - X \geq 3$)		
$3 \leq Y - X \leq 7$		
$4 \leq X \leq 9$		

Simple Temporal Network
Graph for Airline Scenario

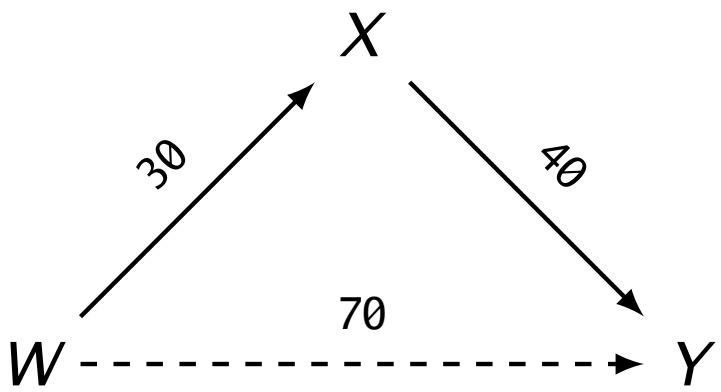
$$\left\{\begin{array}{rcrrcr} Z - X_1 & \leq & -4, & X_4 - Z & \leq & 250 \\ X_4 - X_1 & \leq & 168, & X_2 - X_3 & \leq & -120 \\ X_4 - X_3 & \leq & 7, & X_1 - X_2 & \leq & 0 \\ X_3 - X_4 & \leq & 0 & & & \end{array}\right\}$$



Simple Temporal Network
Implicit Constraints

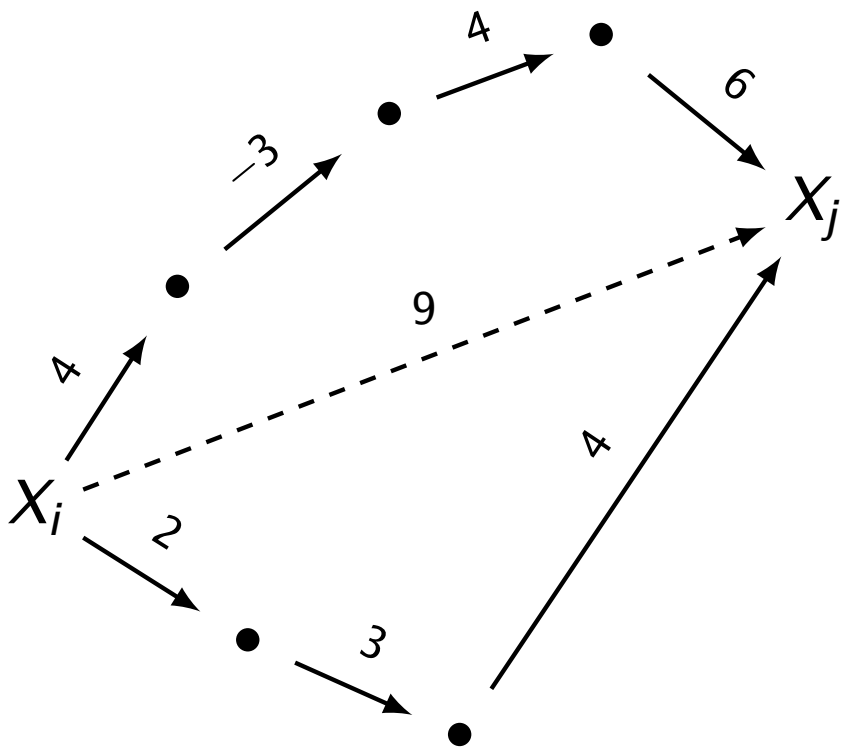
Explicit constraints combine (propagate) to form implicit constraints:

$$\begin{array}{rrcl} & X - W & \leq & 30 \\ + & Y - X & \leq & 40 \\ \hline & Y - W & \leq & 70 \end{array}$$



Simple Temporal Network
Chains of Constraints as Paths

- Chains of constraints correspond to **paths** in the graph.
- **Stronger** constraints correspond to **shorter** paths.

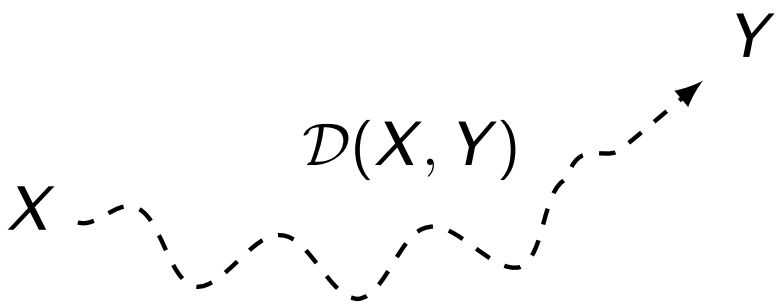


Simple Temporal Network
Distance Matrix*

Definition 2 (Distance Matrix)

The *Distance Matrix* for an STN, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$, is a matrix $\mathcal{D}$ defined by:

$$\mathcal{D}(X, Y) = \text{ Length of Shortest Path from } X \text{ to } Y \text{ in the graph for } \mathcal{S}$$

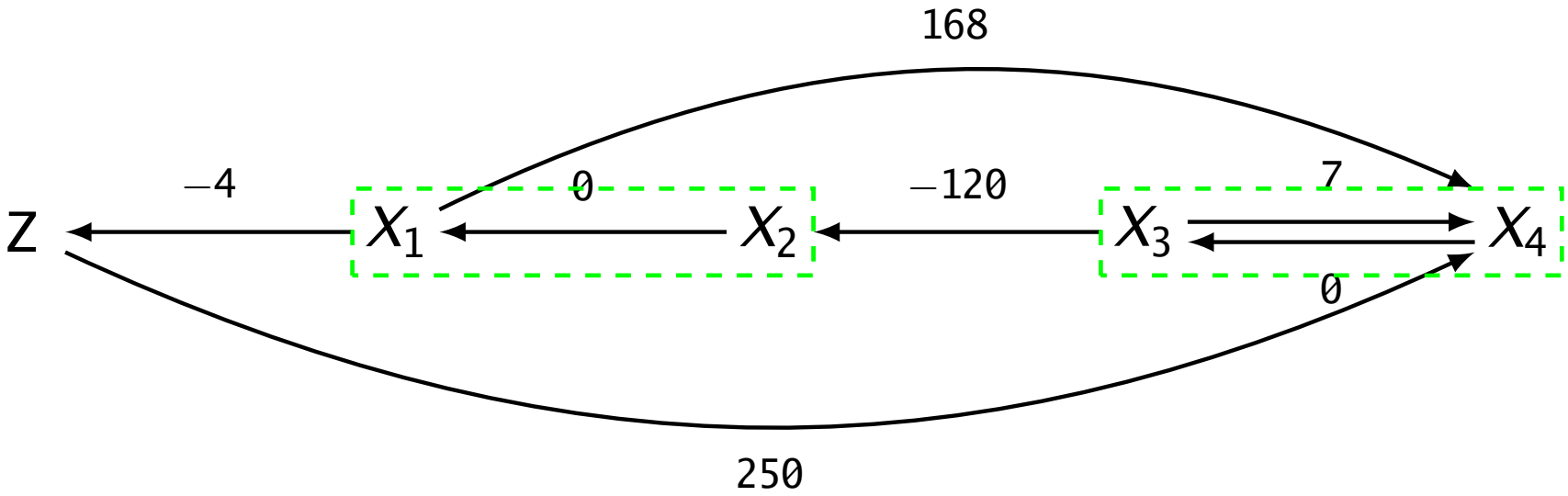


- The strongest implicit constraint on X and Y in $\mathcal{S}$ is:

$$Y - X \leq \mathcal{D}(X, Y)$$

* [Dechter et al., 1991]

Simple Temporal Network
Travel Scenario's Distance Matrix



$\mathcal{D}$	Z	X_1	X_2	X_3	X_4
Z	0	130	130	250	250
X_1	−4	0	48	168	168
X_2	−4	0	0	168	168
X_3	−124	−120	−120	0	7
X_4	−124	−120	−120	0	0

Gray cells correspond to explicit edges.

Simple Temporal Network
Fundamental Theorem of STNs*

For an STN $\mathcal{S}$, with graph $\mathcal{G}$, and distance matrix $\mathcal{D}$, the following are equivalent:

- $\mathcal{S}$ is consistent
- $\mathcal{D}$ has non–negative values down its main diagonal
- $\mathcal{G}$ has no negative–length loops

* [Dechter et al., 1991; Hunsberger, 2014]

Simple Temporal Network
Finding a solution for a consistent STN

- $\mathcal{D}$ has all necessary information.
- *Time window* for any X: $[-\mathcal{D}(X, Z), \mathcal{D}(Z, X)]$
- Two easy–to–find solutions:
  - *Earliest–times* solution:
    $X_1 = -\mathcal{D}(X_1, Z),\ X_2 = -\mathcal{D}(X_2, Z),\ \ldots,\ X_n = -\mathcal{D}(X_n, Z)$;
  - *Latest–times* solution:
    $X_1 = \mathcal{D}(Z, X_1), X_2 = \mathcal{D}(Z, X_2), \ldots, X_n = \mathcal{D}(Z, X_n)$.
- Simple algorithm to find a different solution:
  - Pick any time–point that doesn't yet have a value;
  - Give it a value from its time–window;
  - Update $\mathcal{D}$; $\Leftarrow$ expensive ...
  - Repeat until all time–points have values.

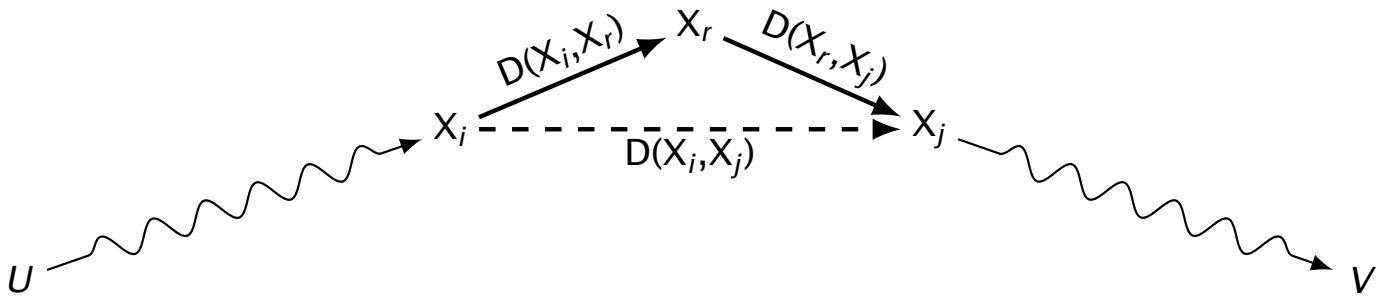
* [Dechter et al., 1991]

Simple Temporal Network
Computing $\mathcal{D}$ from Scratch*

- $\mathcal{D}$ is the *All–Pairs, Shortest–Paths* (APSP) Matrix for $\mathcal{G}$.
- If $\mathcal{S}$ has n time–points and m constraints:
  - Floyd–Warshall Algorithm: $O(n^3)$
  - Johnson's Algorithm: $O(n^2 \log n + mn)$
    — uses Bellman–Ford and Dijkstra

* [Cormen et al., 2001]

Floyd–Warshall Algorithm*

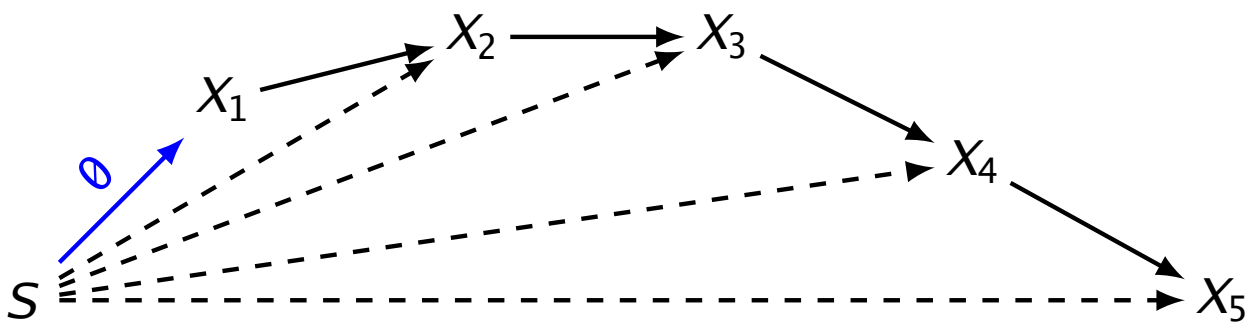



```
Initialize D(_,_) using edge-weights
for r=1 to n
   for i=1 to n
      for j=1 to n
         D(Xi,Xj) := min{D(Xi,Xj), D(Xi,Xr) + D(Xr,Xj)}
```

If a shortest path from U to V contains X_r as an interior point, then after the r^{th} round, that shortest path can ignore X_r.

* [Cormen et al., 2001]

Bellman–Ford Algorithm*




```
for each X, d(X) := 0

for i=1 to (n-1),
   for each edge (U,δ,V) in graph,
      d(V) := min{d(V), d(U) + δ}

for each edge (U,δ,V) in graph,
   if (d(V) > d(U) + δ) return false

return true
```

* [Cormen et al., 2001]

Dijkstra's SSSP Algorithm*

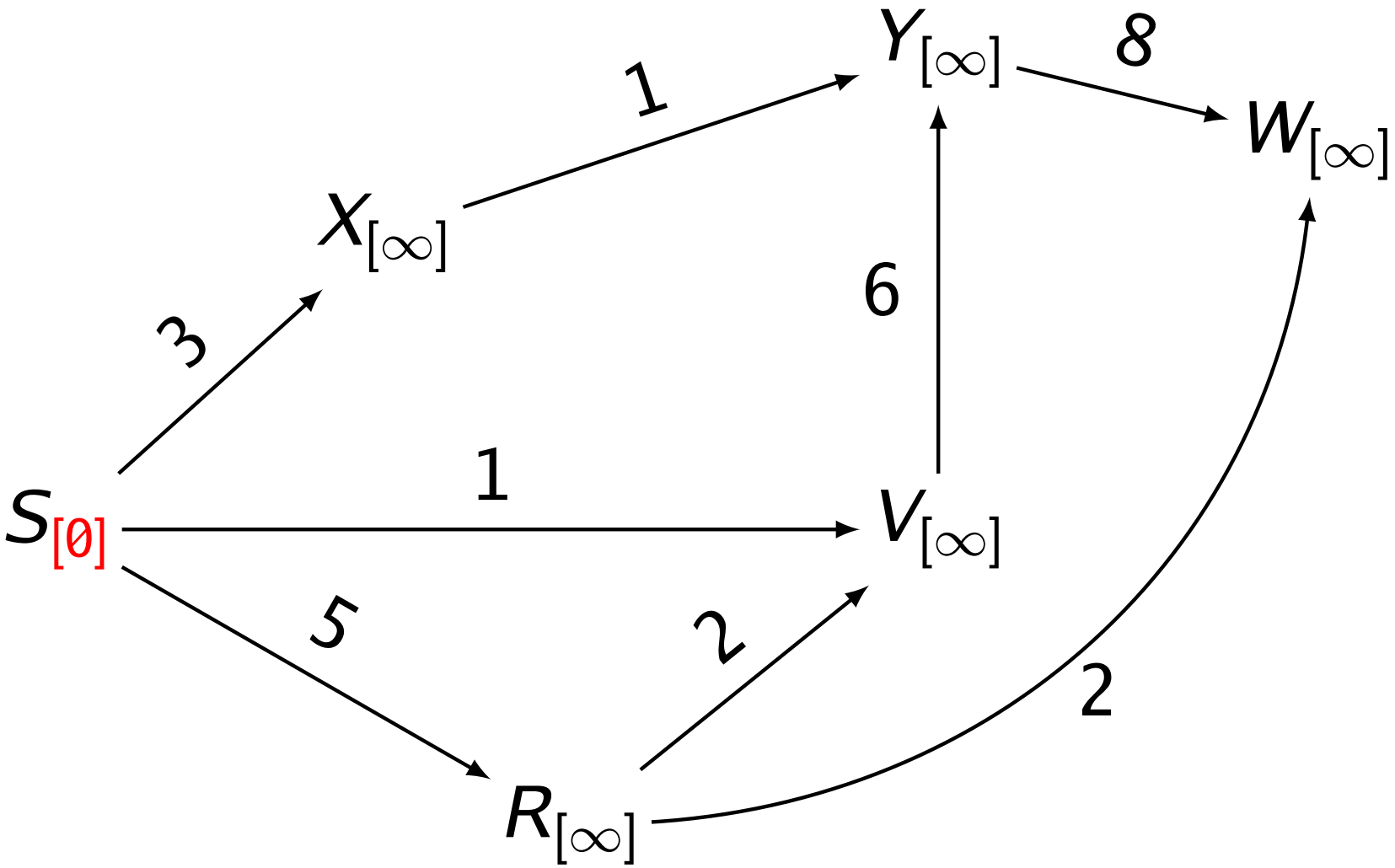
Single–Source Shortest–Paths

- Works on STN graphs with non–negative edges
- For a given source node S, computes $\mathcal{D}(S, X)$ for all X
- $O(m + n \log n)$ if using *fibonacci heap* for priority queue

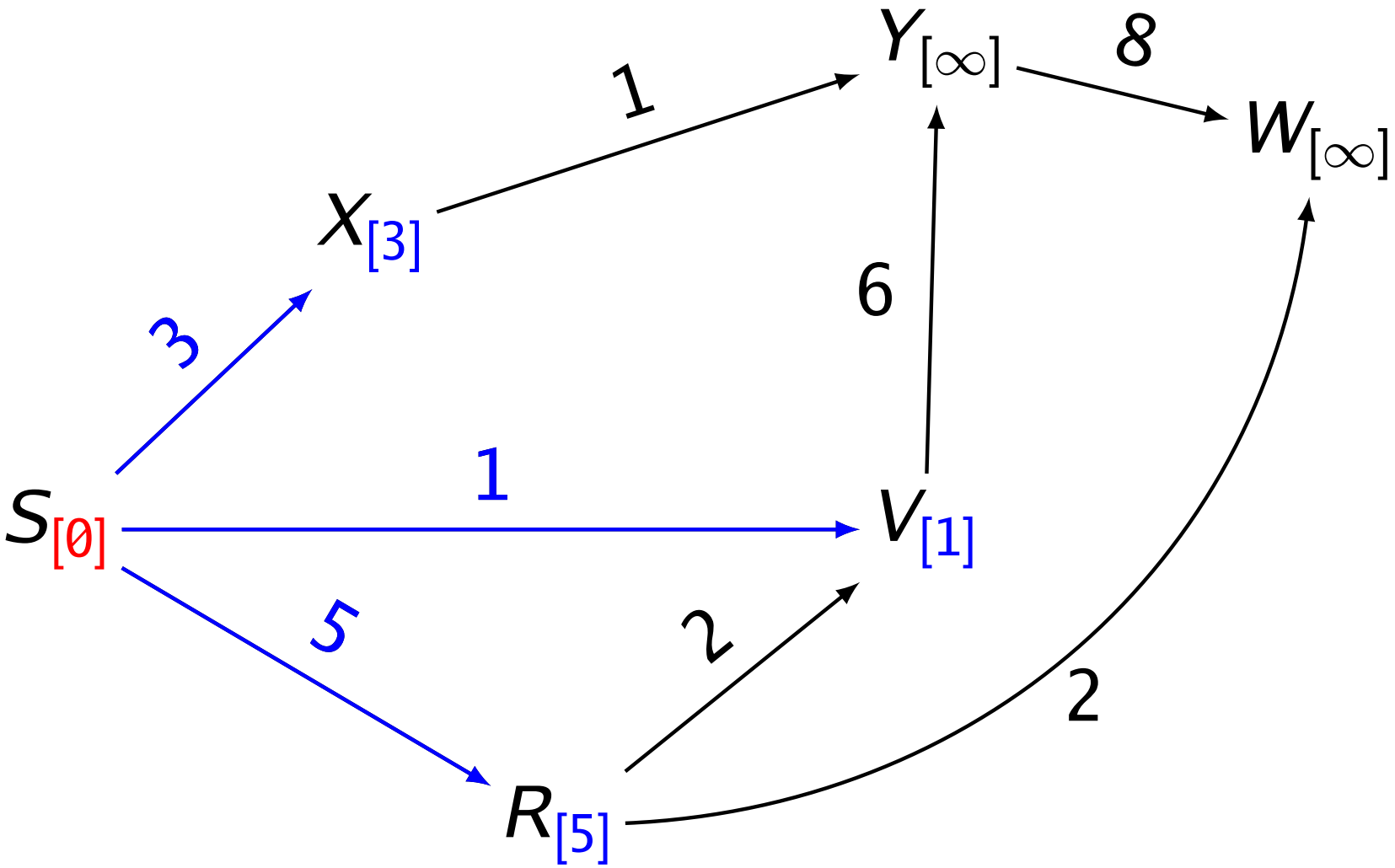
```
Let D(S, X) := ∞ for all X, except D(S, S) := 0
Let Q be an empty priority queue
Insert S into Q with priority 0
while Q non-empty,
   U := ExtractMinFrom(Q)
   for each successor edge (U, δ, V),
      D(S, V) := min{D(S, V), D(S, U) + δ}
return D
```

* [Cormen et al., 2001]

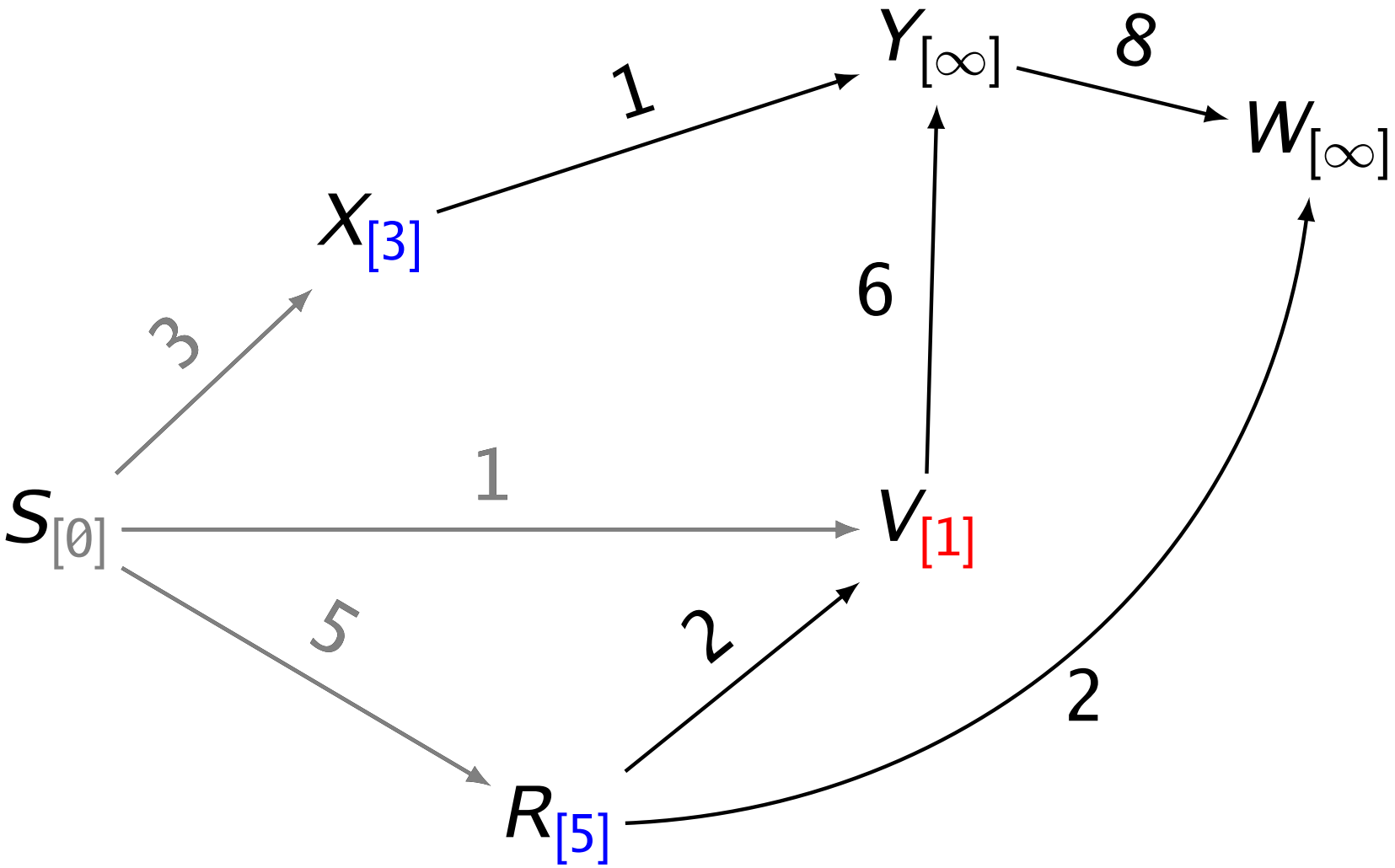
Dijkstra Example



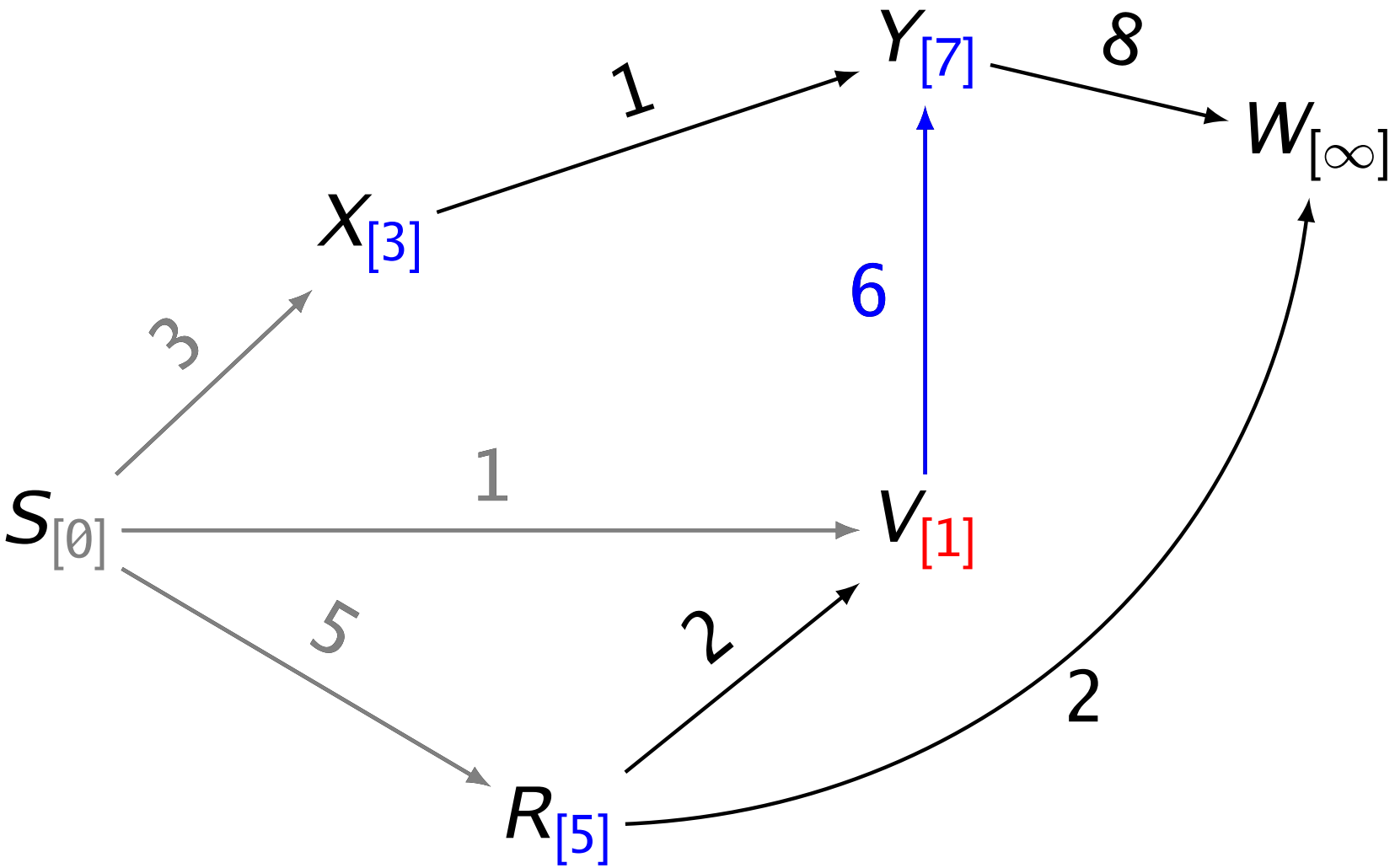
Dijkstra Example



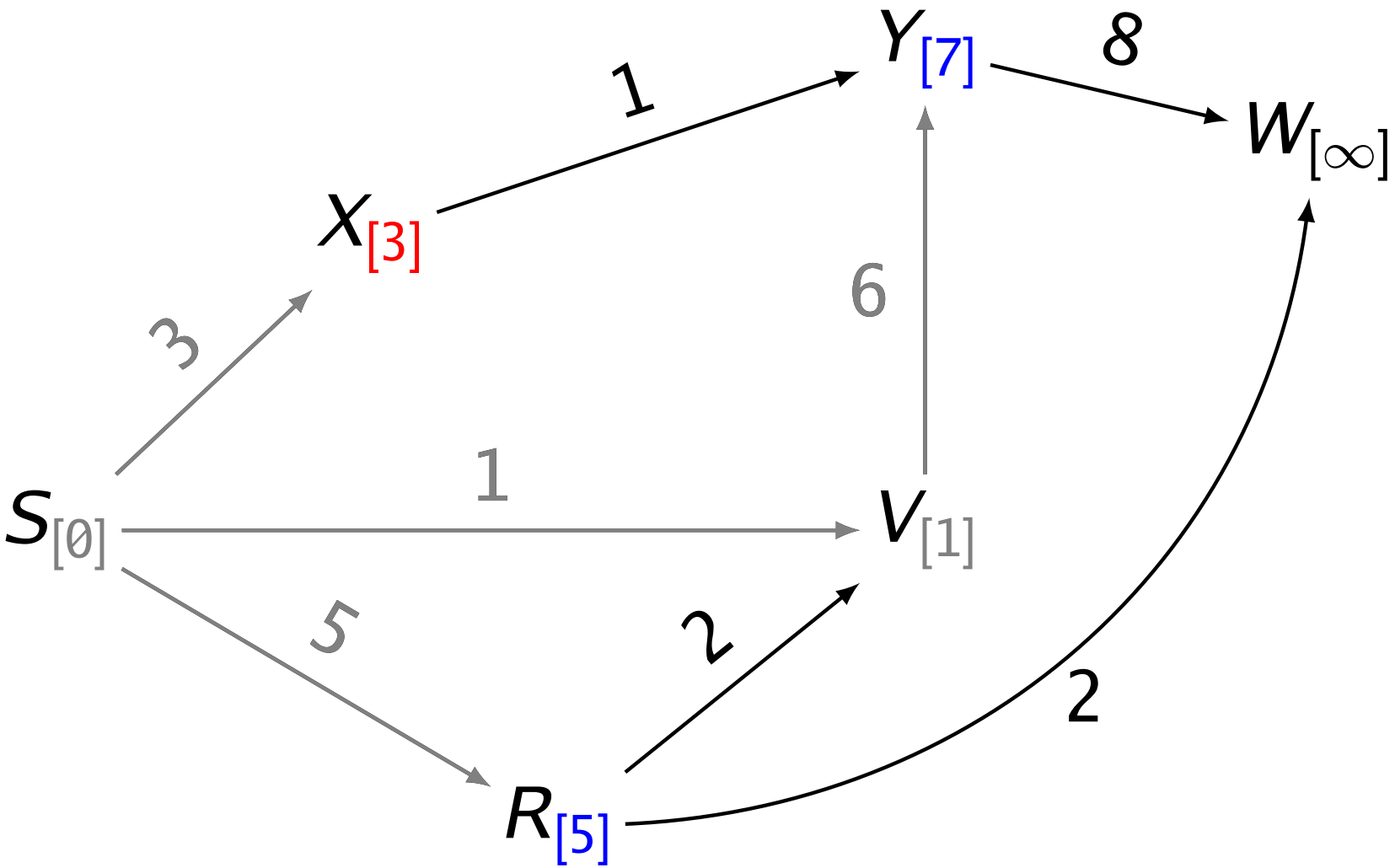
Dijkstra Example



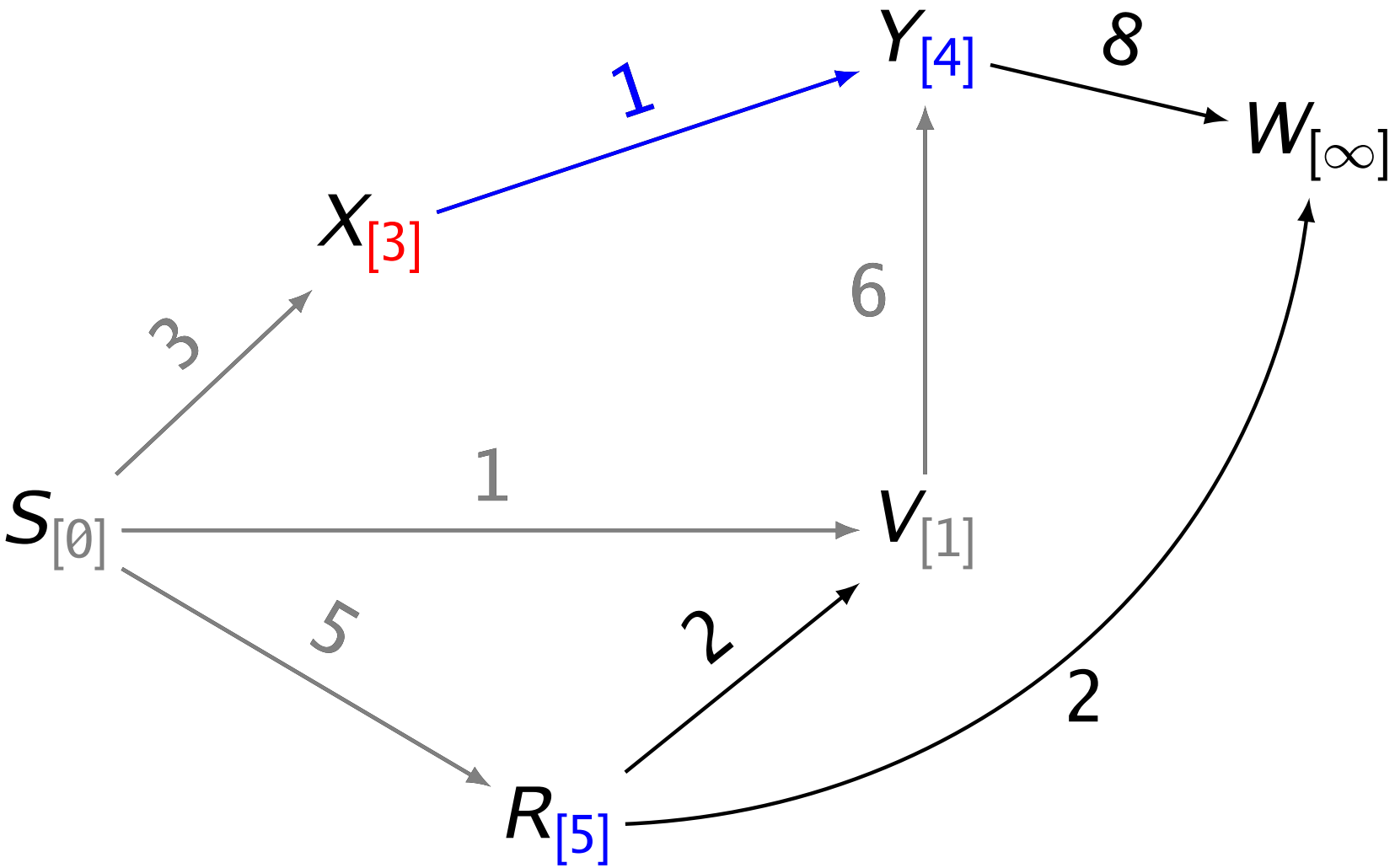
Dijkstra Example



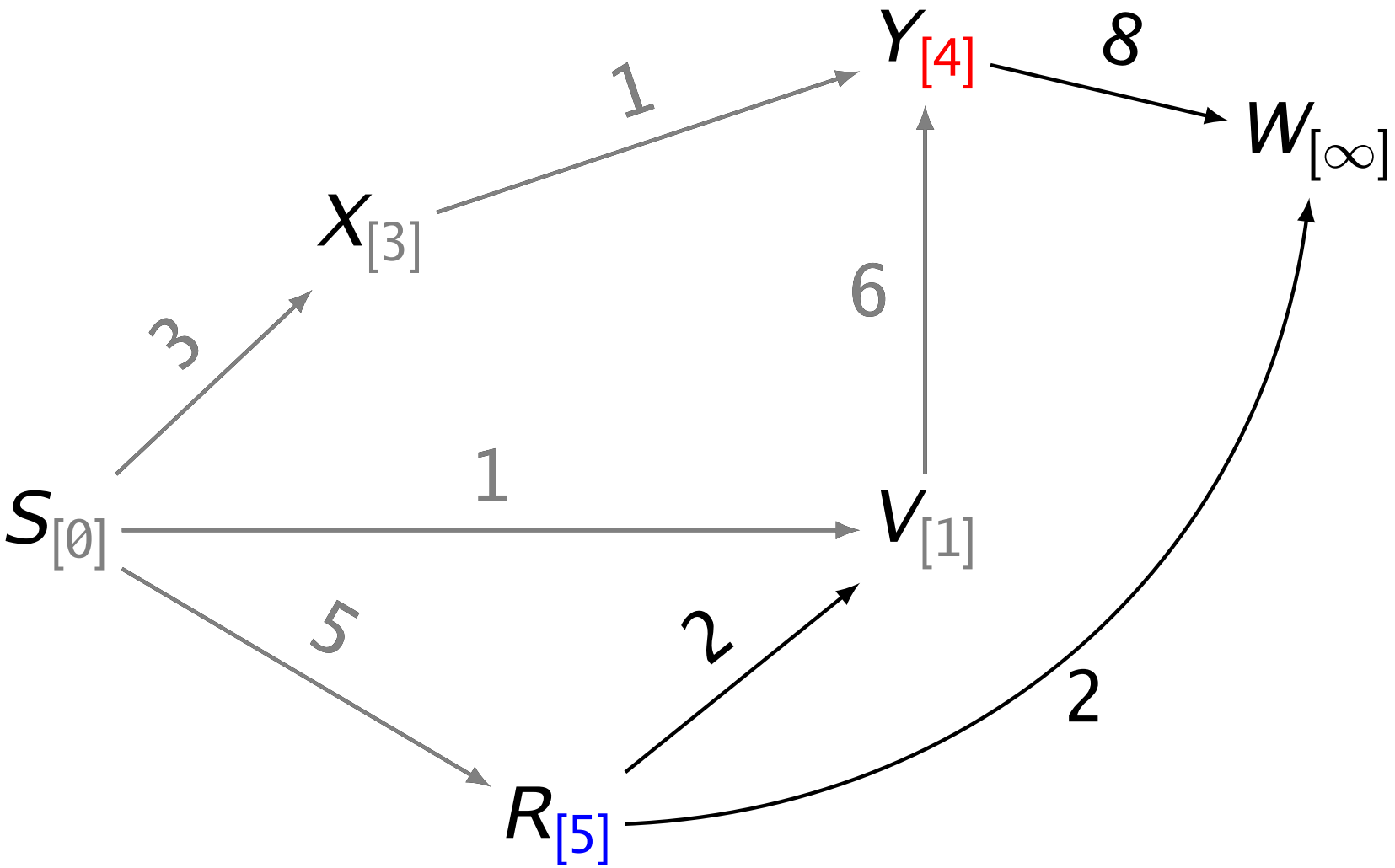
Dijkstra Example



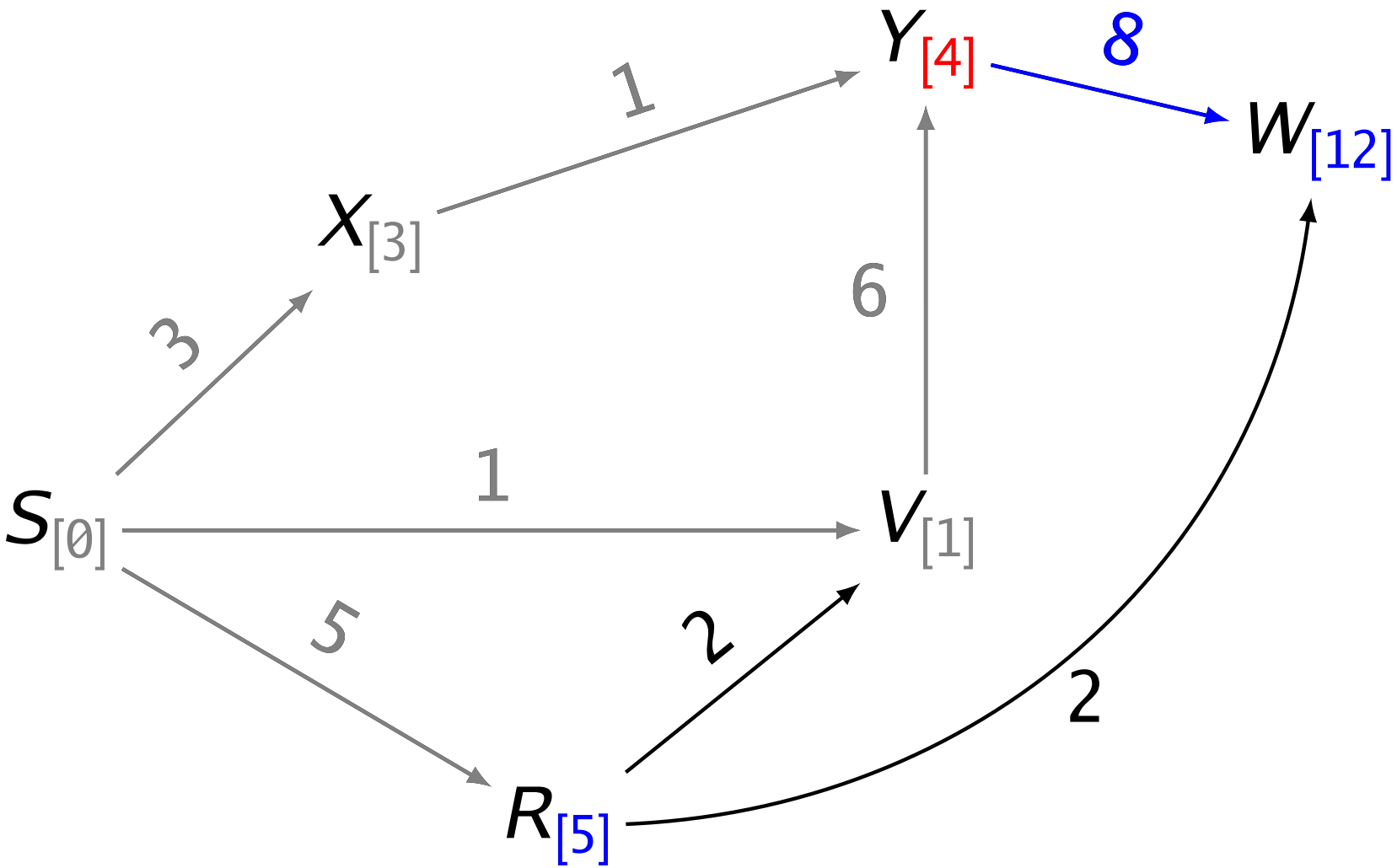
Dijkstra Example



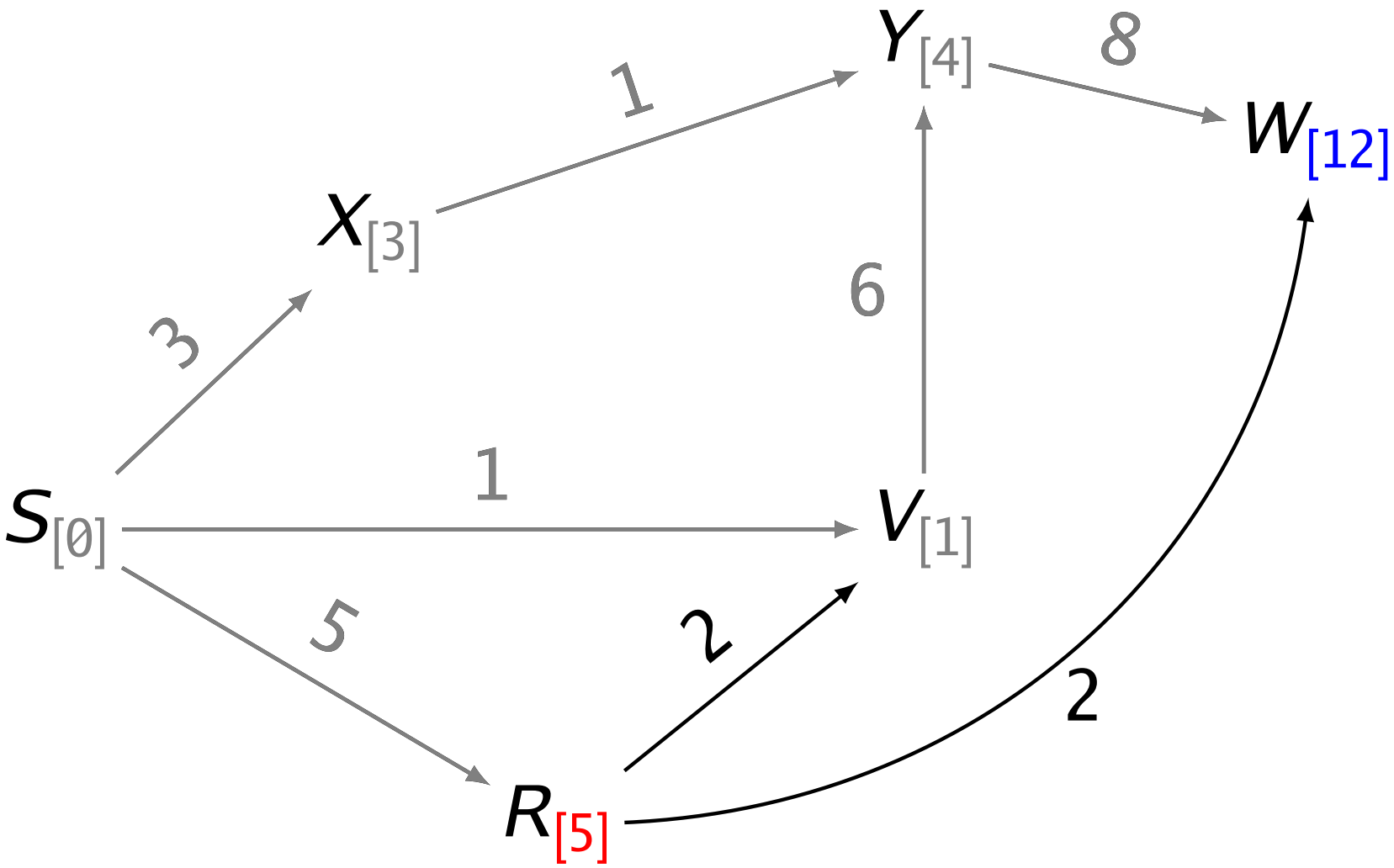
Dijkstra Example



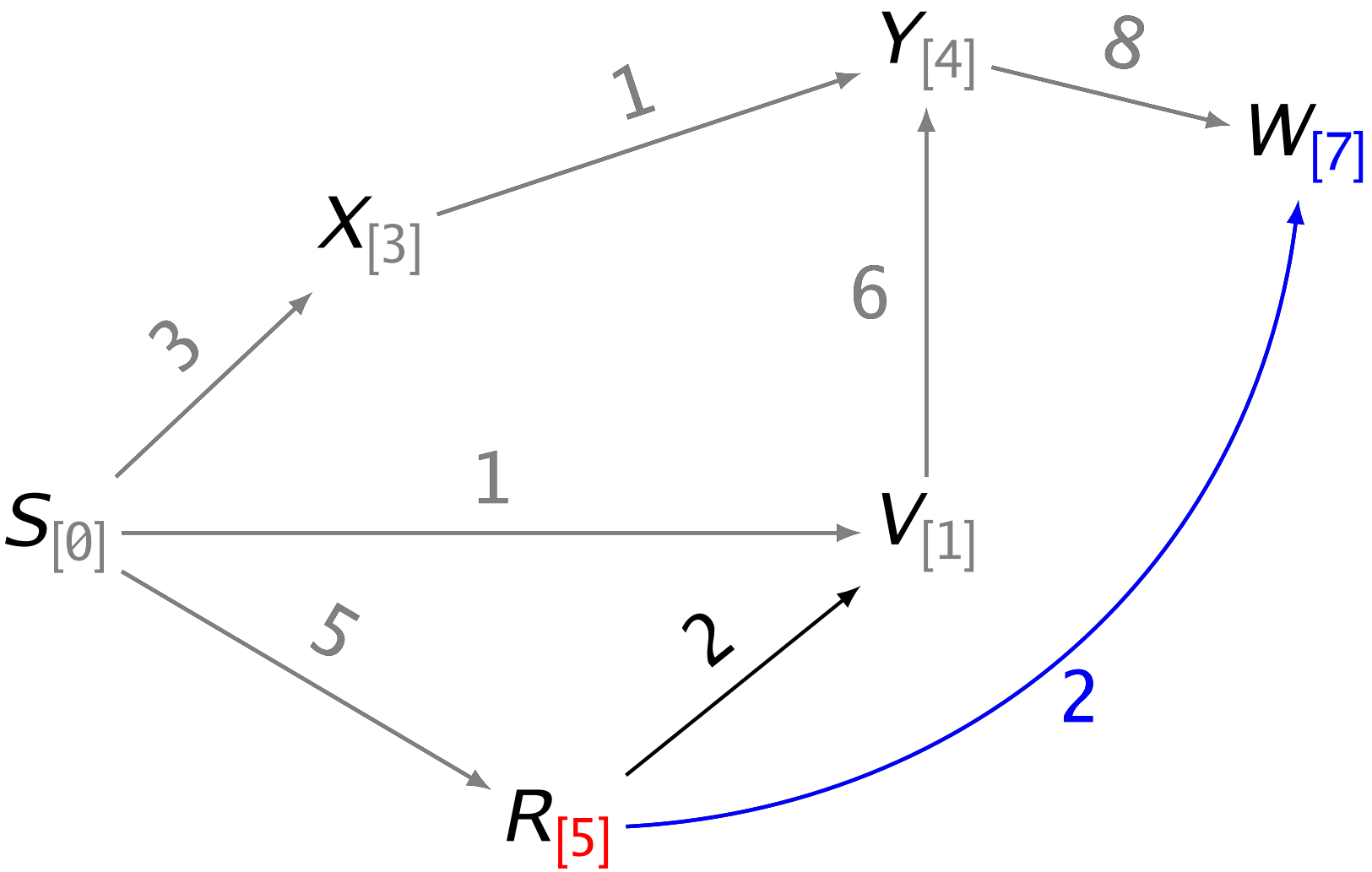
Dijkstra Example



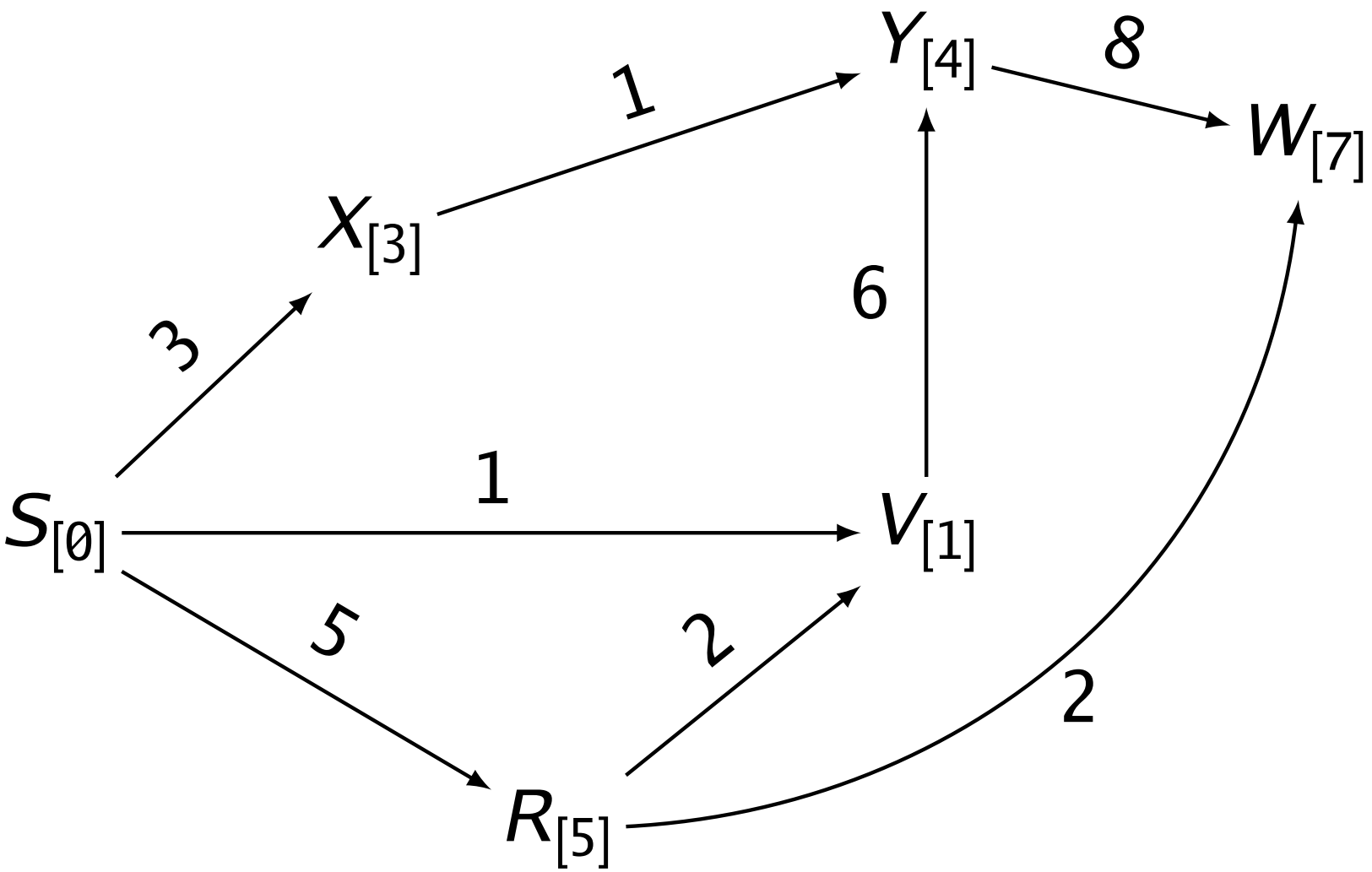
Dijkstra Example



Dijkstra Example



Dijkstra Example



Potential Functions & Re–weighted Graphs

- Let $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ be any consistent STN.
- Let $f : \mathcal{T} \rightarrow \mathbb{R}$ be any solution for $\mathcal{S}$.
  - Then $f(Y) - f(X) \leq \delta$ for each constraint $(Y - X \leq \delta) \in \mathcal{C}$
  - In other words, $0 \leq f(X) + \delta - f(Y)$
  - Let $\mathcal{C}' = \{(X, \delta', Y) \mid (X, \delta, Y) \in \mathcal{C}\}$, where $\delta' = f(X) + \delta - f(Y)$
  - Then $\mathcal{S}' = (\mathcal{T}, \mathcal{C}')$ has only non–negative edges.
  - Therefore, can use Dijkstra's SSSP algorithm on $\mathcal{S}'$

Johnson's Algorithm*

Given: an STN, $\mathcal{S} = (\mathcal{T}, \mathcal{C})$

Run Bellman–Ford SSSP with *new* source node S

Then $f : \mathcal{T} \to \mathbb{R}$ given by $f(X) = D(S, X)$ is a solution for $\mathcal{S}$.

Let $\mathcal{S}' = (\mathcal{T}, \mathcal{C}')$ be re–weighted graph based on f:

$\delta' = f(X) + \delta - f(Y) \geq 0$ for each $(X, \delta, Y) \in \mathcal{C}$.

For each $X \in \mathcal{T}$, run Dijkstra on $\mathcal{S}'$ with X as source node

— computes $\mathcal{D}'(X, Y)$ for all $Y \in \mathcal{T}$.

Reverse the re–weighting to obtain $\mathcal{D}$ for $\mathcal{S}$:

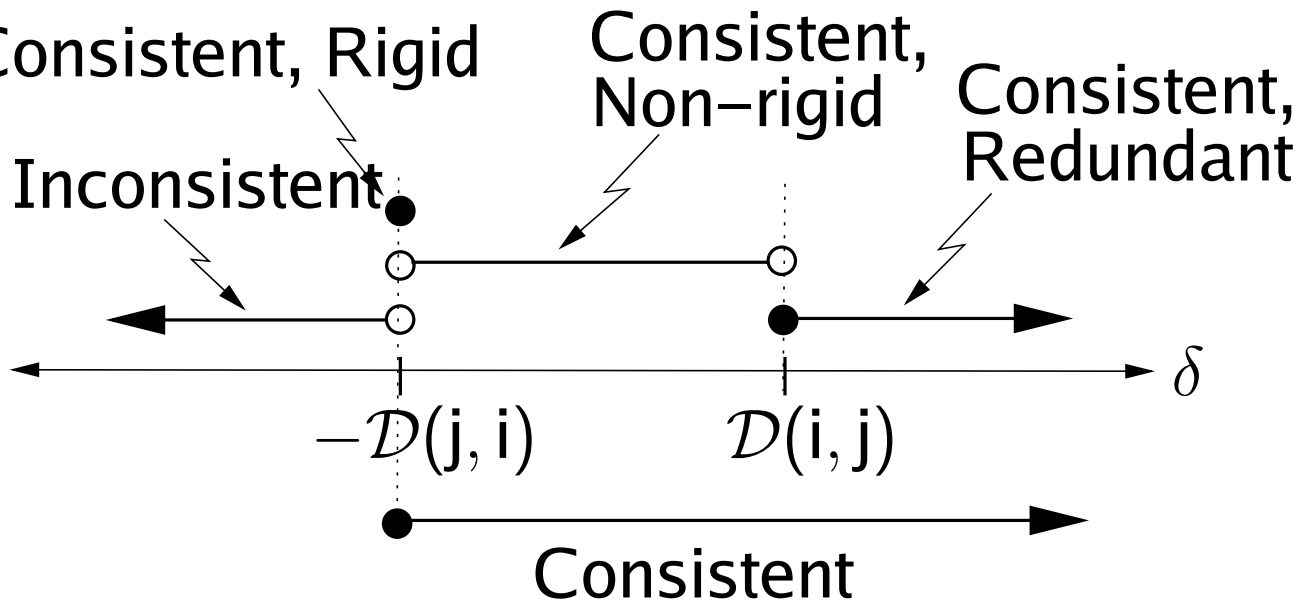
$\mathcal{D}(X, Y) = -f(X) + \mathcal{D}'(X, Y) + f(Y)$.

Complexity: $O(mn) + n * O(m + n \log n) = O(mn + n^2 \log n)$

* [Cormen et al., 2001]

Simple Temporal Network
Inserting new constraint/edge

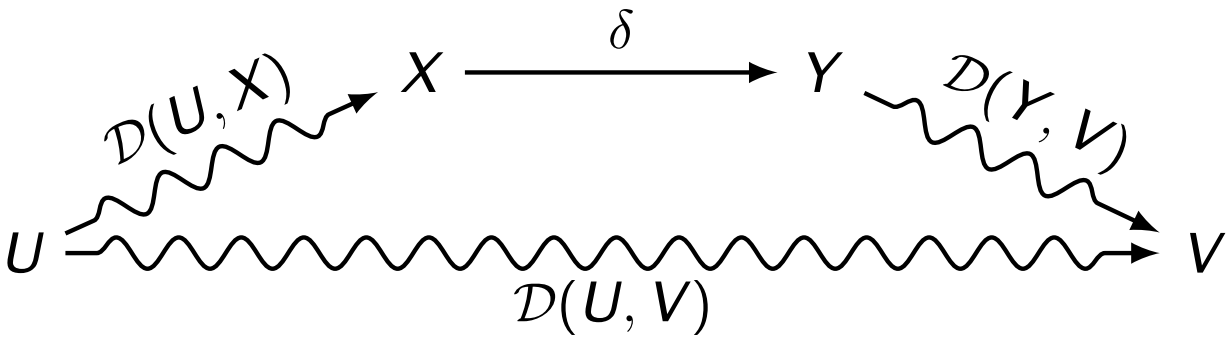
Result of inserting new constraint, $Y - X \leq \delta$:



Simple Temporal Network
Incrementally updating $\mathcal{D}$

- Given a consistent STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ with distance matrix $\mathcal{D}$.
- Insert new constraint $(Y - X \leq \delta)$.
- How to update $\mathcal{D}$?
  - Re–compute from scratch: $O(mn + n^2 \log n)$.
  - "Naïve" algorithm: $O(n^2)$: For each $U, V \in \mathcal{T}$,

$$\mathcal{D}(U, V) := \min\{\mathcal{D}(U, V),\ \mathcal{D}(U, X) + \delta + \mathcal{D}(Y, V)\}$$



Simple Temporal Networks
Incremental Update Algorithm

- Propagate updates to $\mathcal{D}$ along edges in graph.
- Only propagate along *tight* edges.
  $(Y - X \leq \delta)$ is tight iff $\mathcal{D}(X, Y) = \delta$.
- Phase I: propagate forward
- Phase II: propagate backward
- Checks no more than $b*\Delta$ cells of $\mathcal{D}$, where:

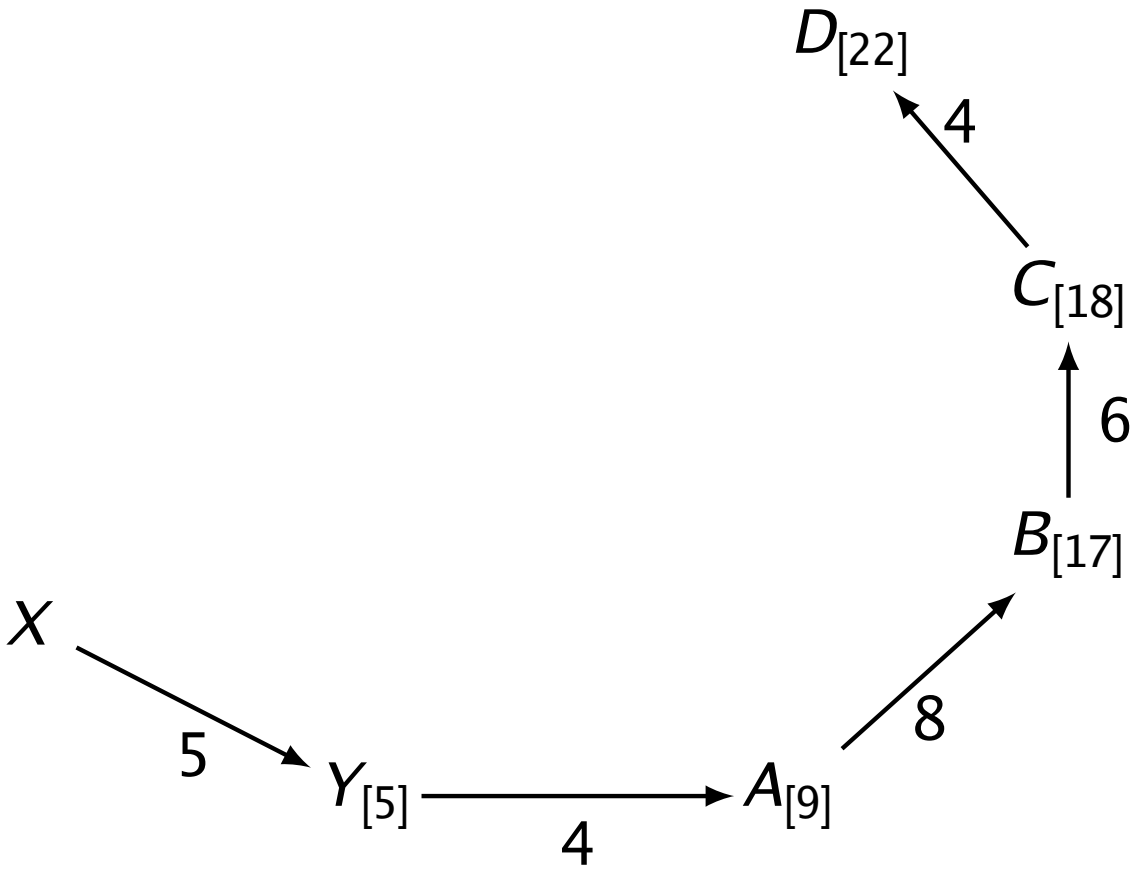
  $\Delta =$ number of cells needing updating;

  $b =$ max. number of edges incident to any node.

* [Even and Gazit, 1985; Ramalingam and Reps, 1996; Rohnert, 1985]

Incremental Update Algorithm

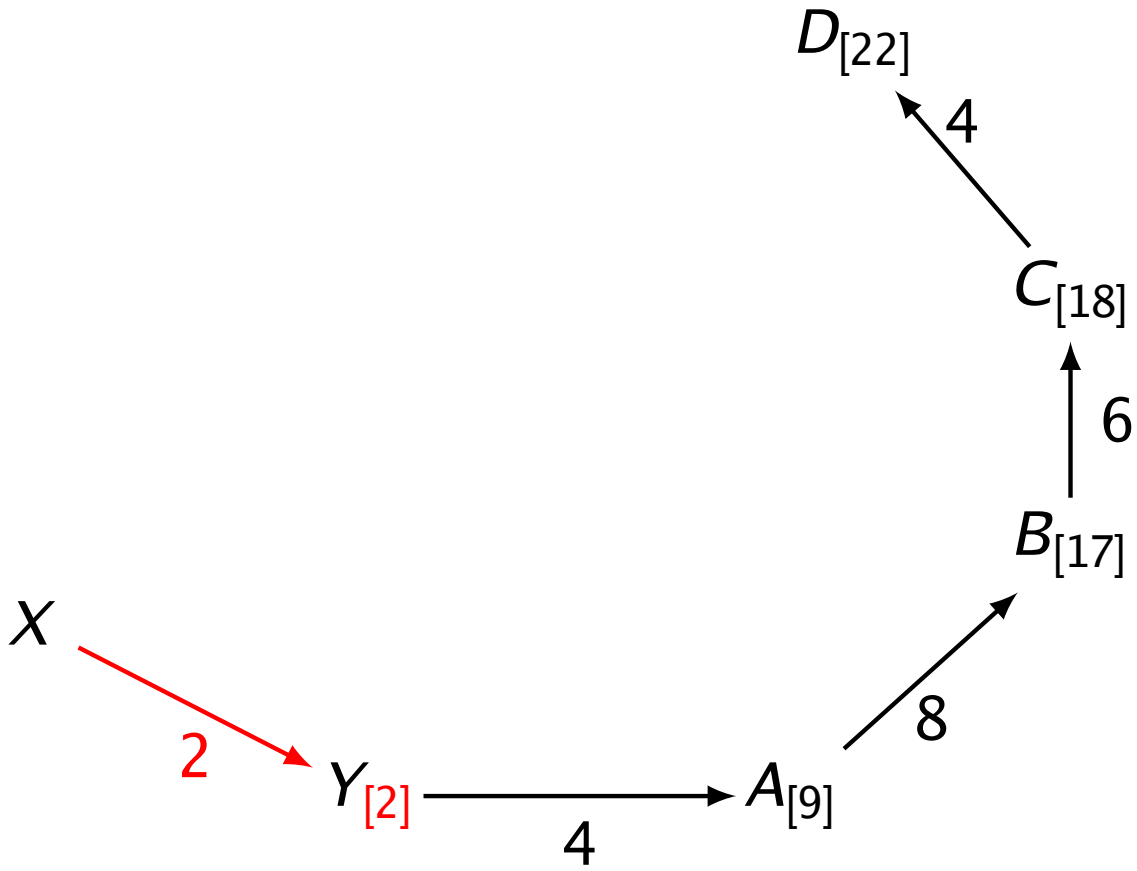
Propagate forward from XY as long as new values obtained



Numbers in brackets are current values of $\mathcal{D}(X, \_)$.

Incremental Update Algorithm

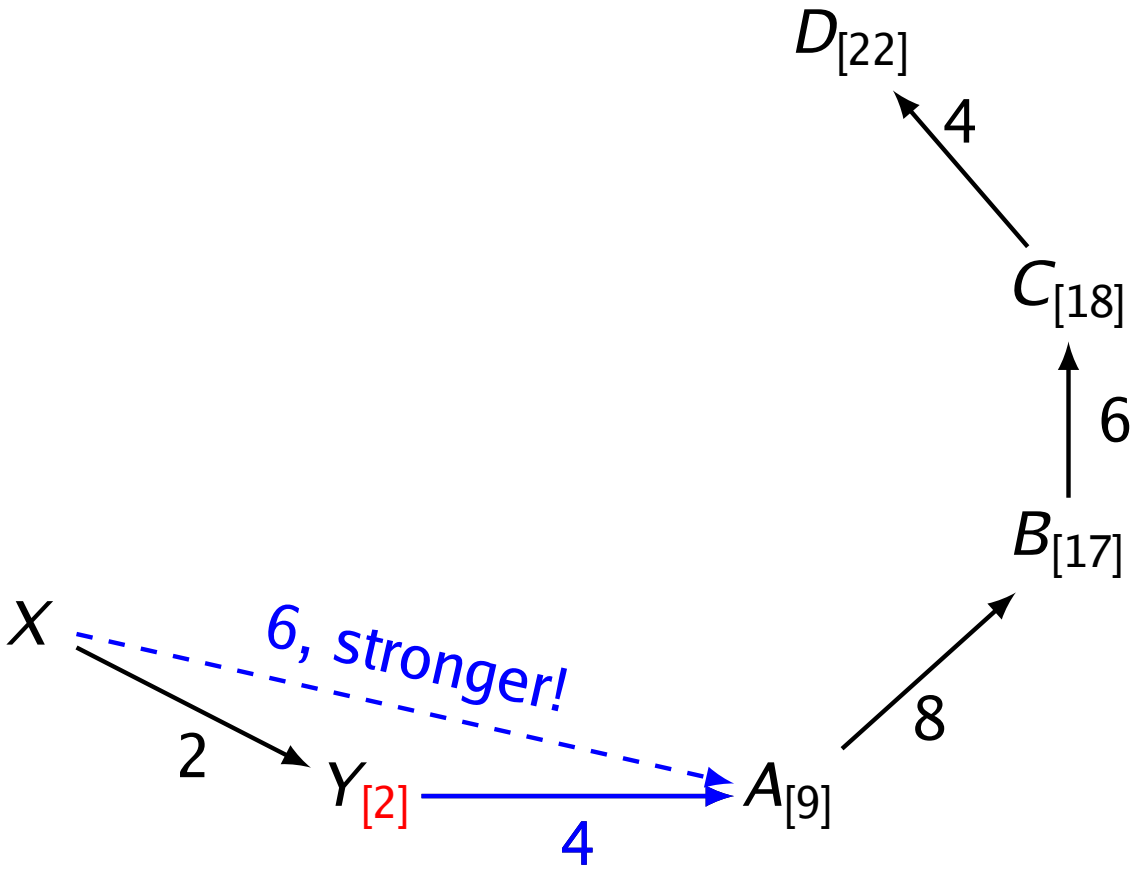
Propagate forward from XY as long as new values obtained



Numbers in brackets are current values of $\mathcal{D}(X, \_)$.

Incremental Update Algorithm

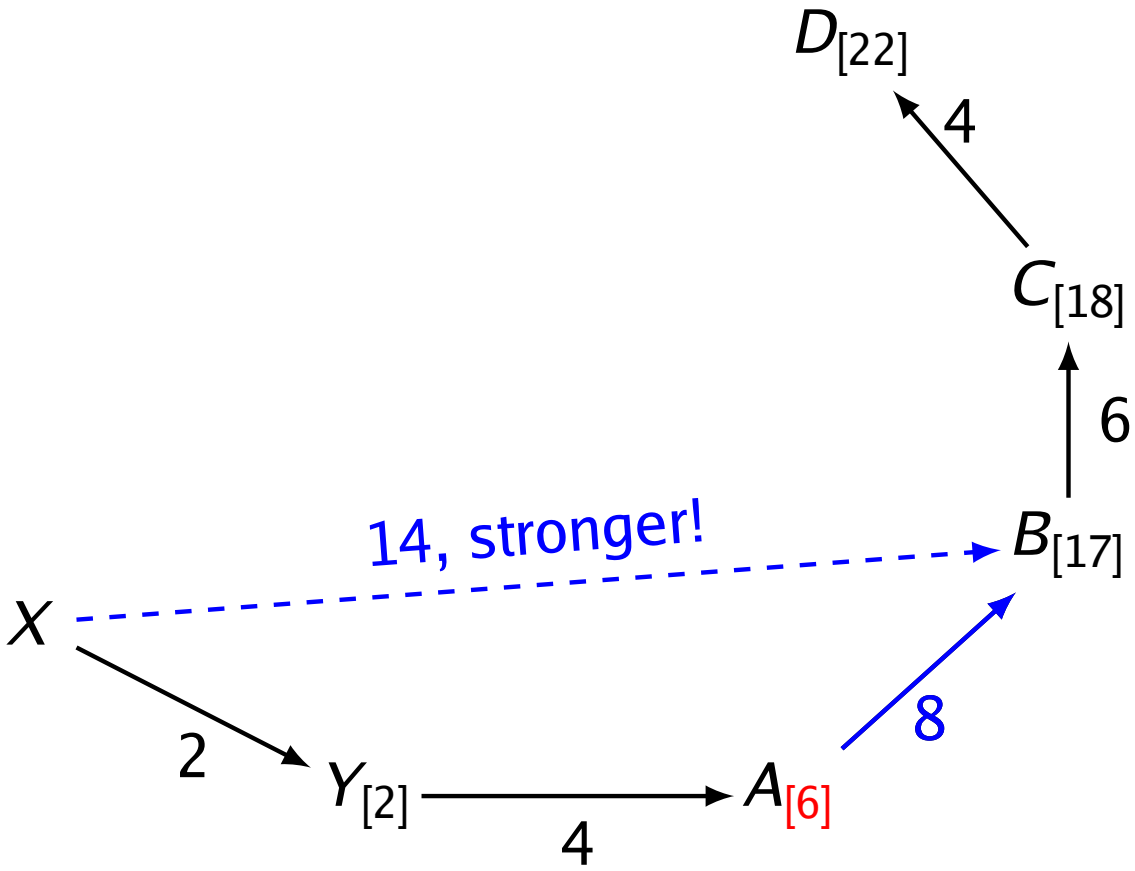
Propagate forward from XY as long as new values obtained



Numbers in brackets are current values of $\mathcal{D}(X, \_)$.

Incremental Update Algorithm

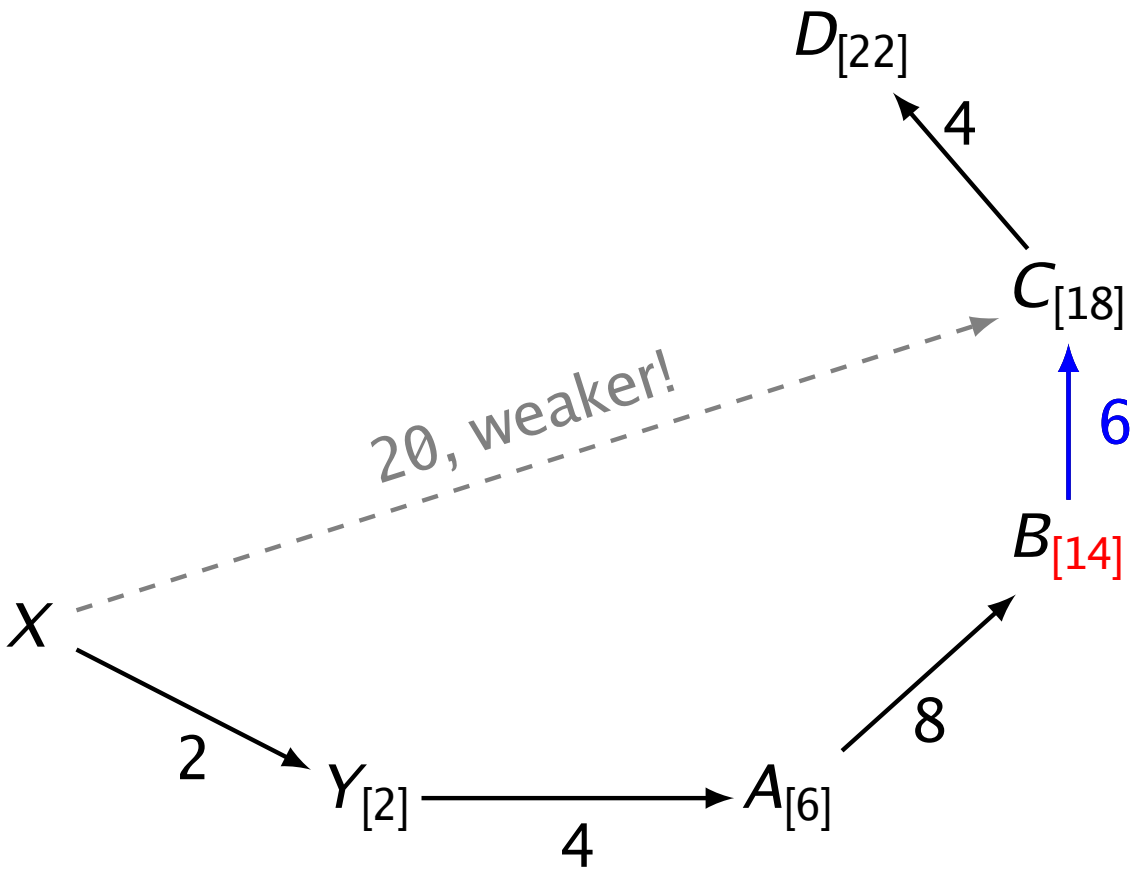
Propagate forward from XY as long as new values obtained



Numbers in brackets are current values of $\mathcal{D}(X, \_)$.

Incremental Update Algorithm

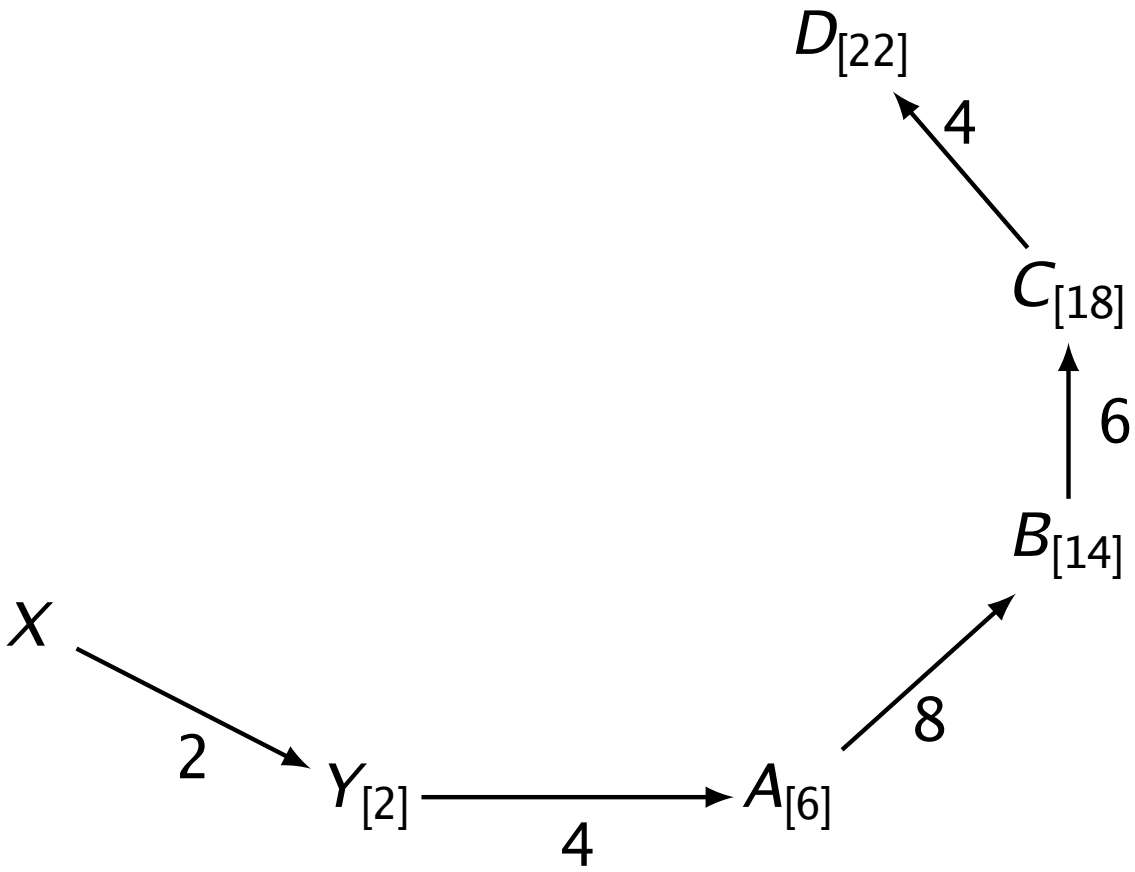
Propagate forward from XY as long as new values obtained



Propagation stops!

Incremental Update Algorithm

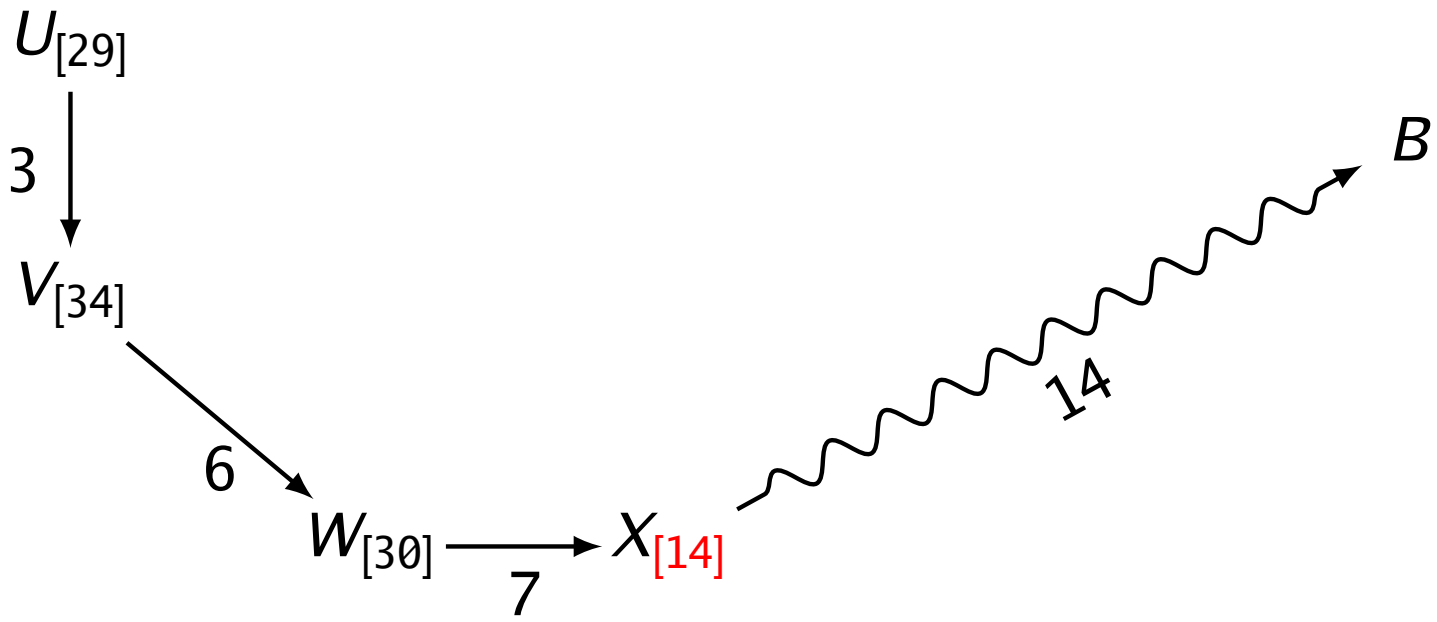
Propagate forward from XY as long as new values obtained



Forward propagtion done along this path.

Incremental Update Algorithm

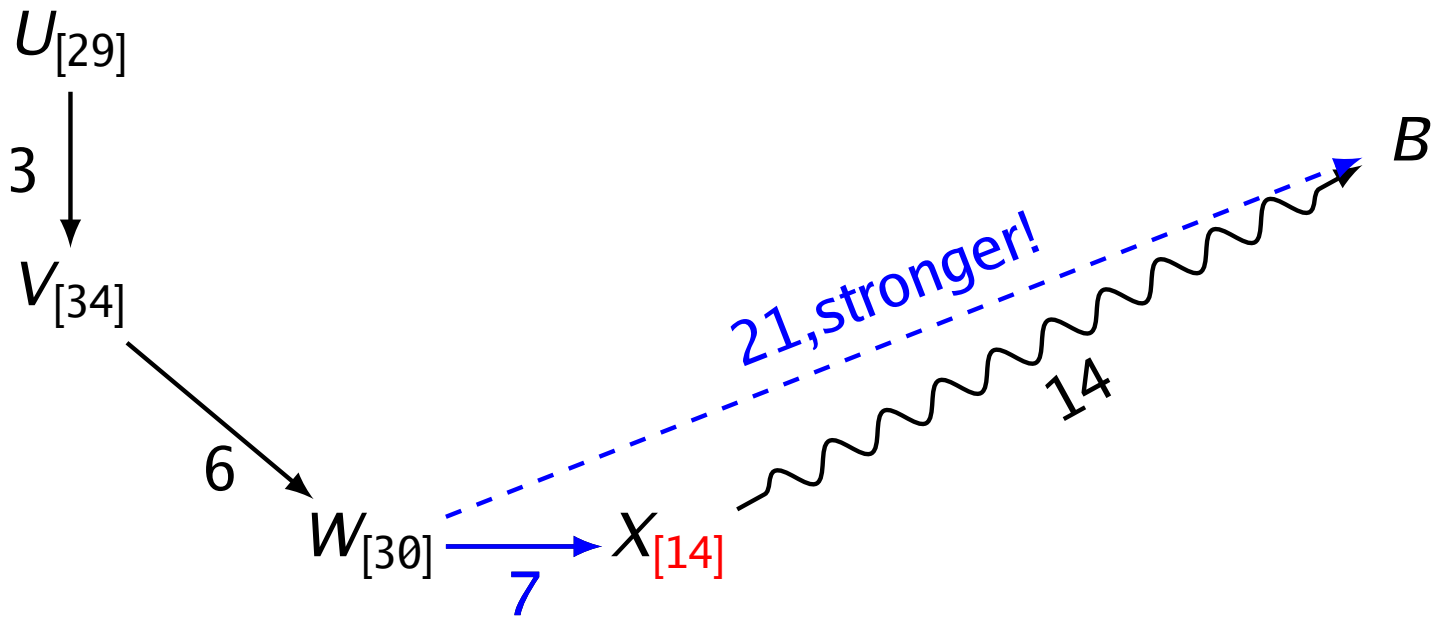
For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



Numbers in brackets are $\mathcal{D}(\_, B)$ values.

Incremental Update Algorithm

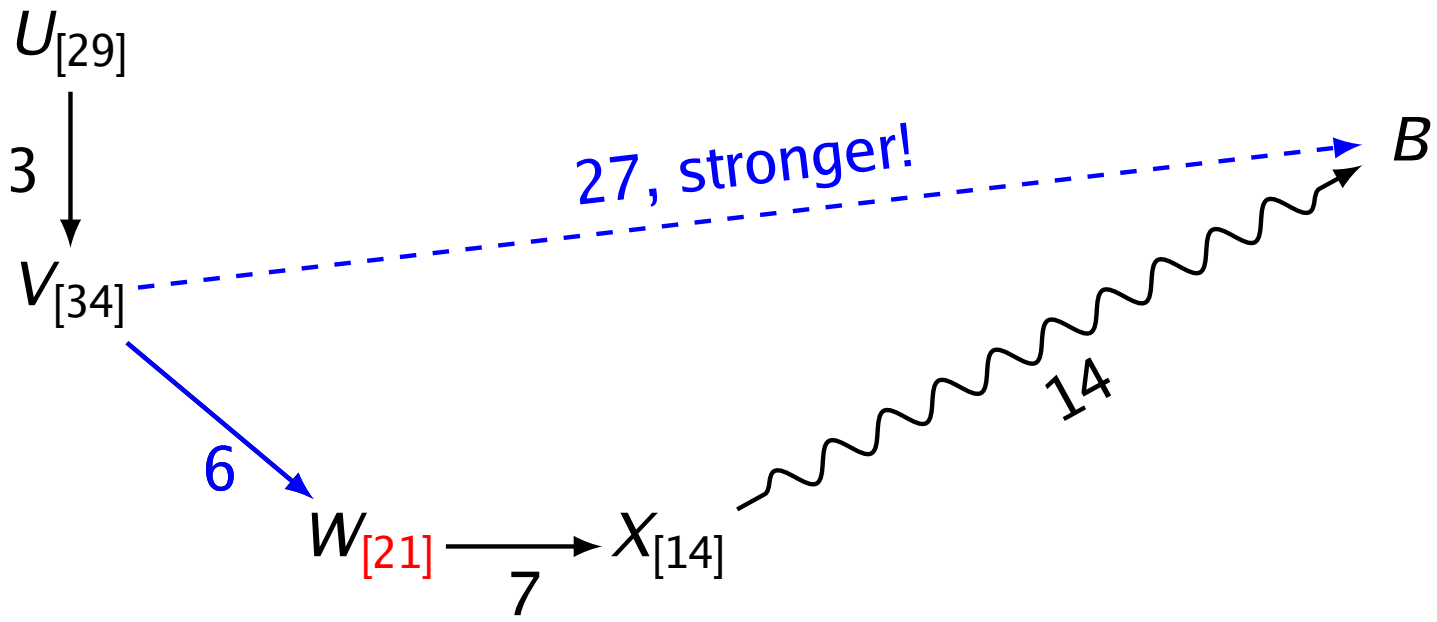
For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



Numbers in brackets are $\mathcal{D}(\_, B)$ values.

Incremental Update Algorithm

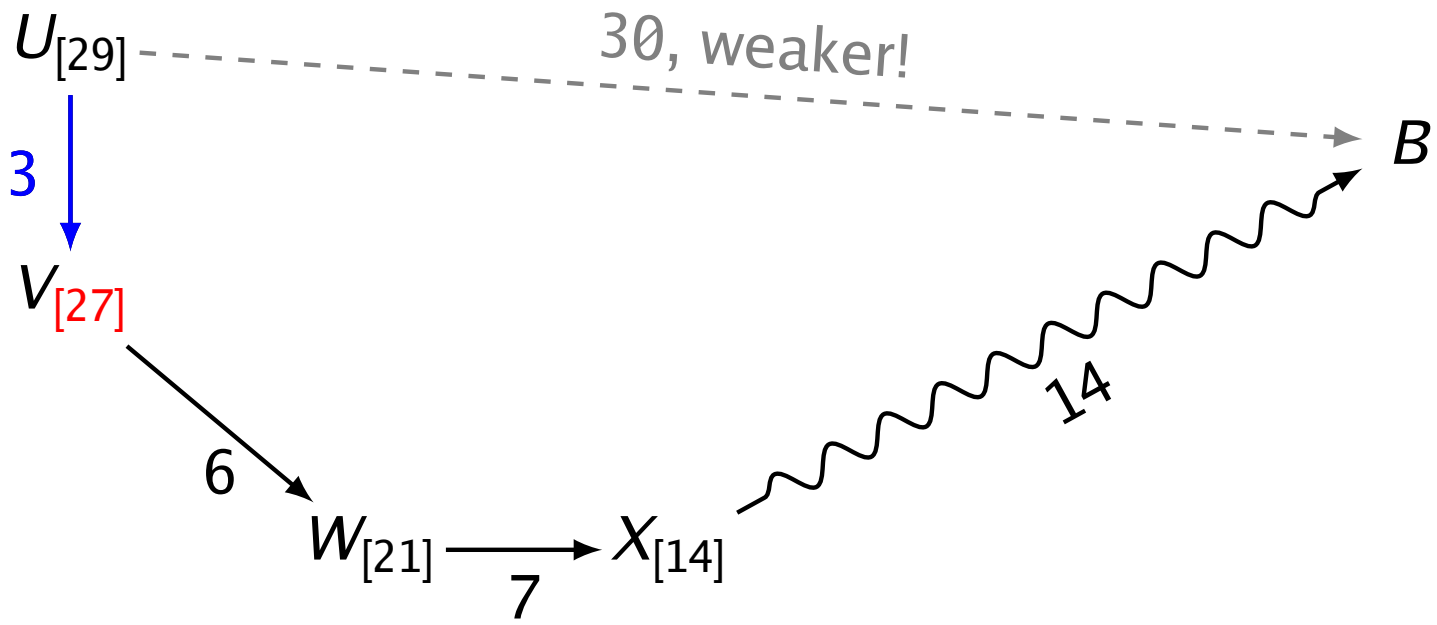
For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



Numbers in brackets are $\mathcal{D}(\_, B)$ values.

Incremental Update Algorithm

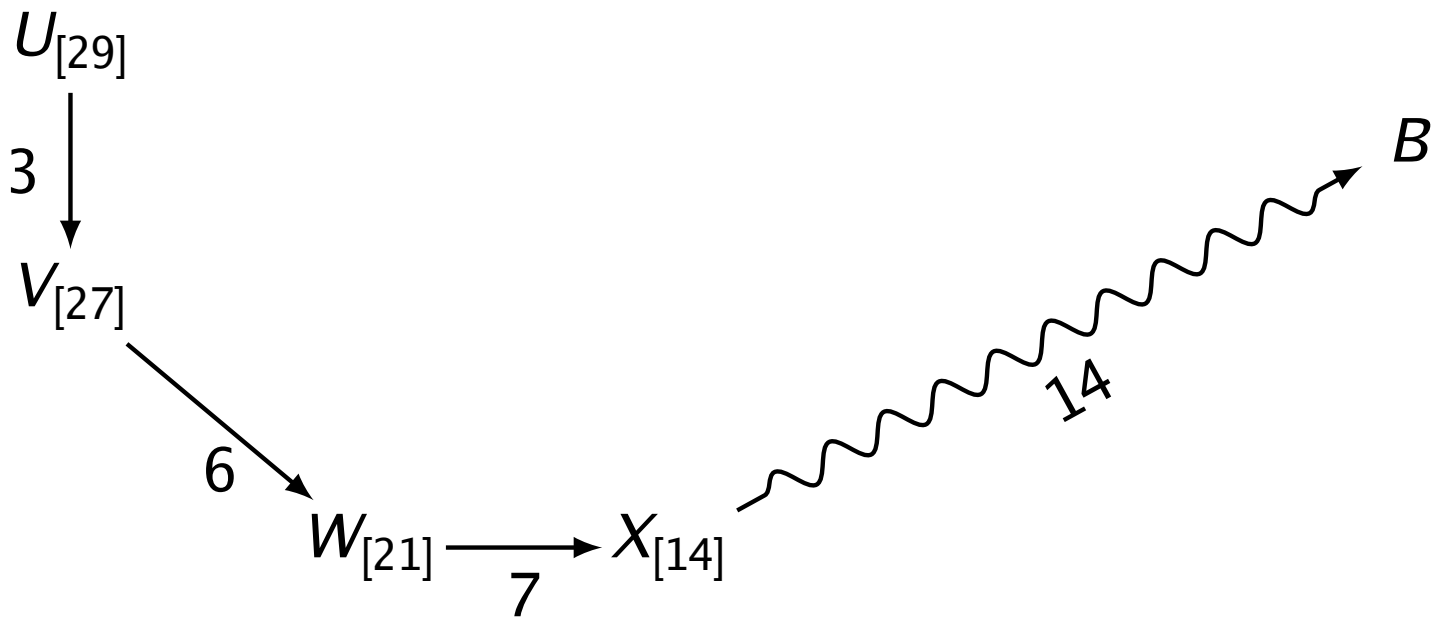
For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



No further updates along this path.

Incremental Update Algorithm

For each entry $\mathcal{D}(X, \_)$ that was updated, propagate backward from X



No further updates along this path.

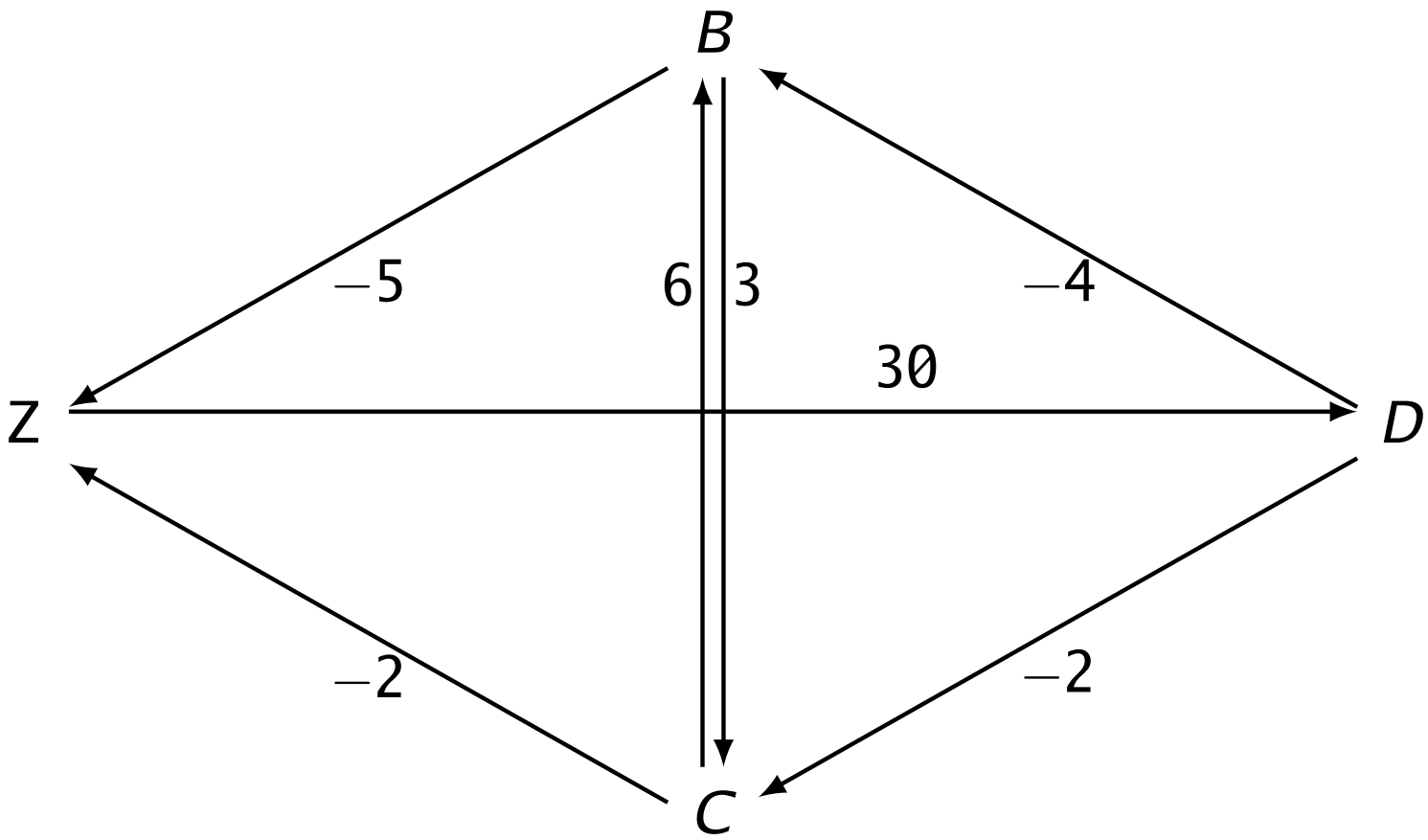
Simple Temporal Network
Incremental Consistency

Verifying consistency of an STN after inserting, weakening or deleting a constraint is less expensive than fully updating the distance matrix.*

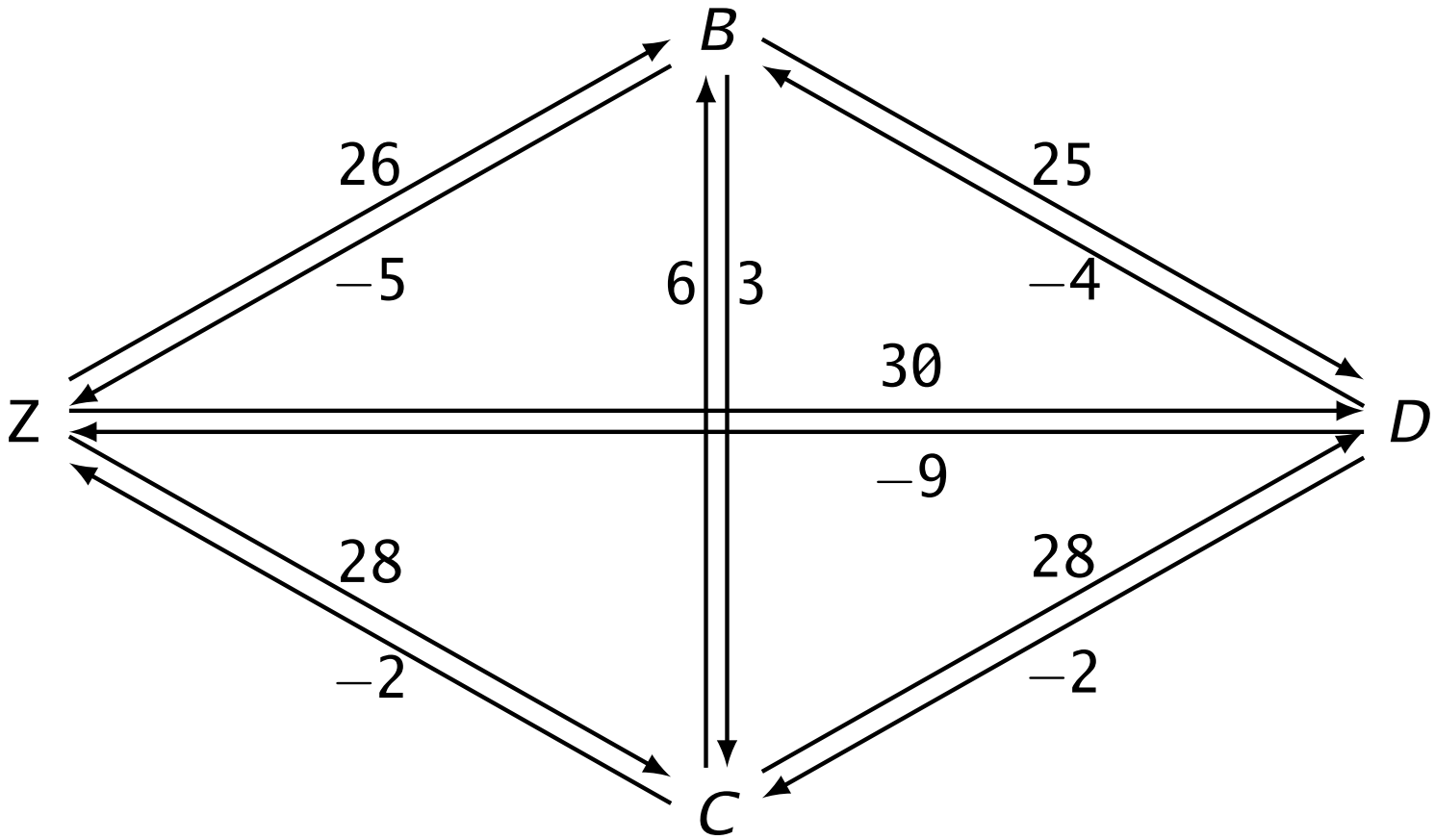
- Algorithm maintains/updates a solution to the STN.
- After inserting a new constraint (or strengthening an existing one), can verify consistency in $O(m + n \log n)$ time.
- After deleting or weakening a constraint, only need constant time, because the same solution will work for the modified STN.

* [Ramalingam et al., 1999]

Simple Temporal Network
Executing an STN in real time

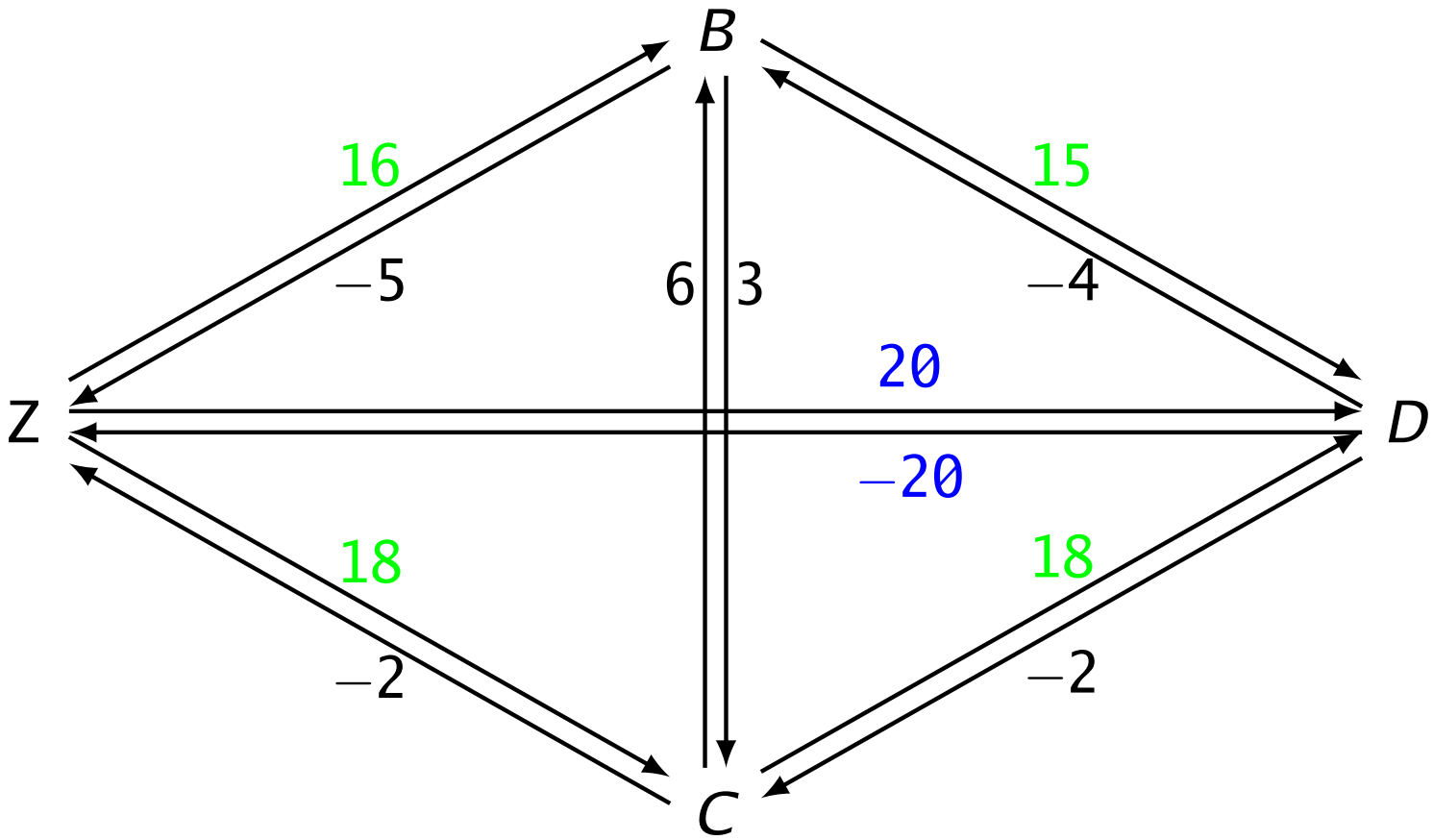


First, form APSP graph (equiv. compute $\mathcal{D}$).



Time Windows: $B \in [5, 26]$, $C \in [2, 28]$, $D \in [9, 30]$

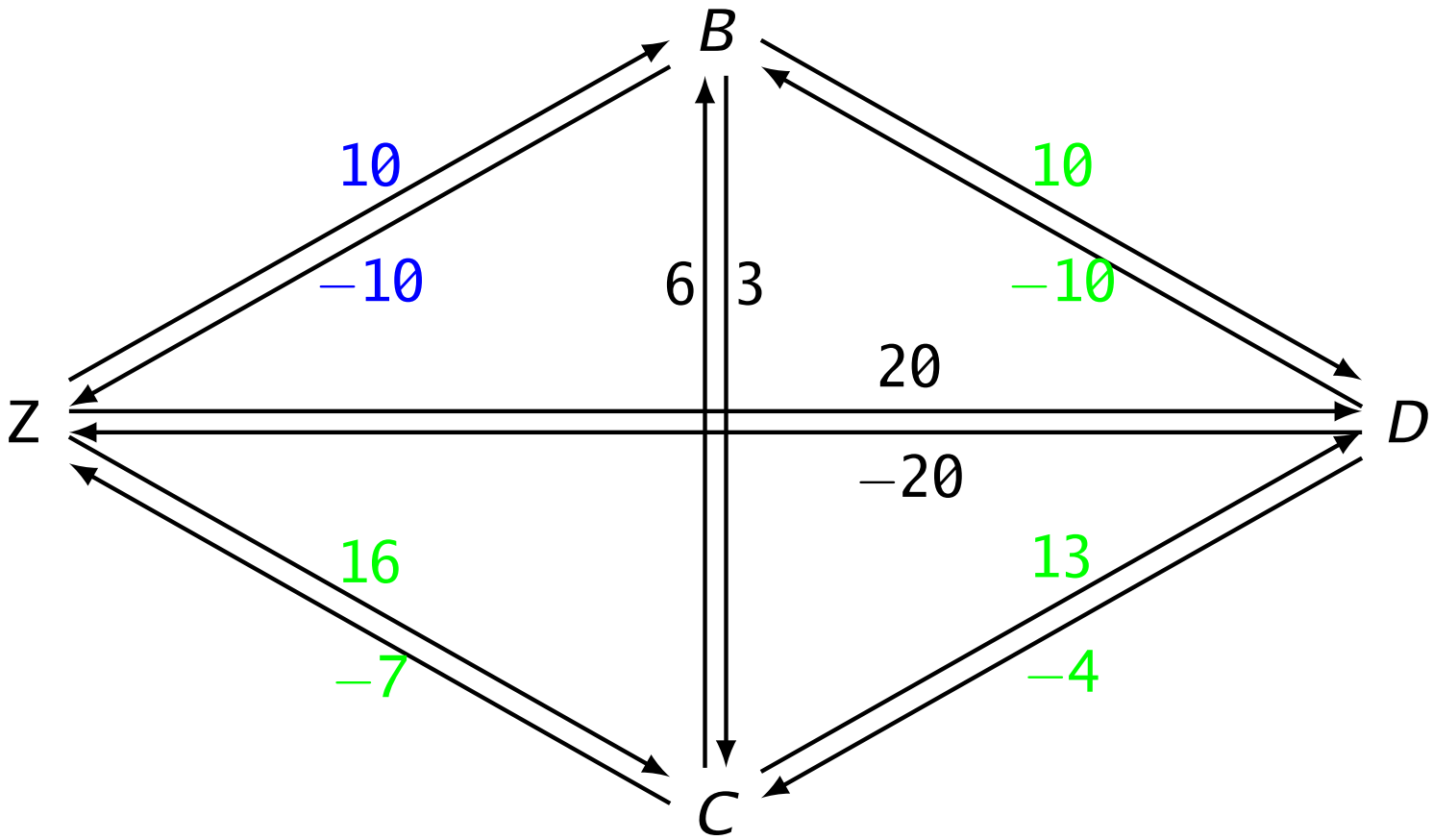
Next, select $D = 20$; and update APSP graph:



Remaining Time Windows: $B \in [5, 16]$, $C \in [2, 18]$

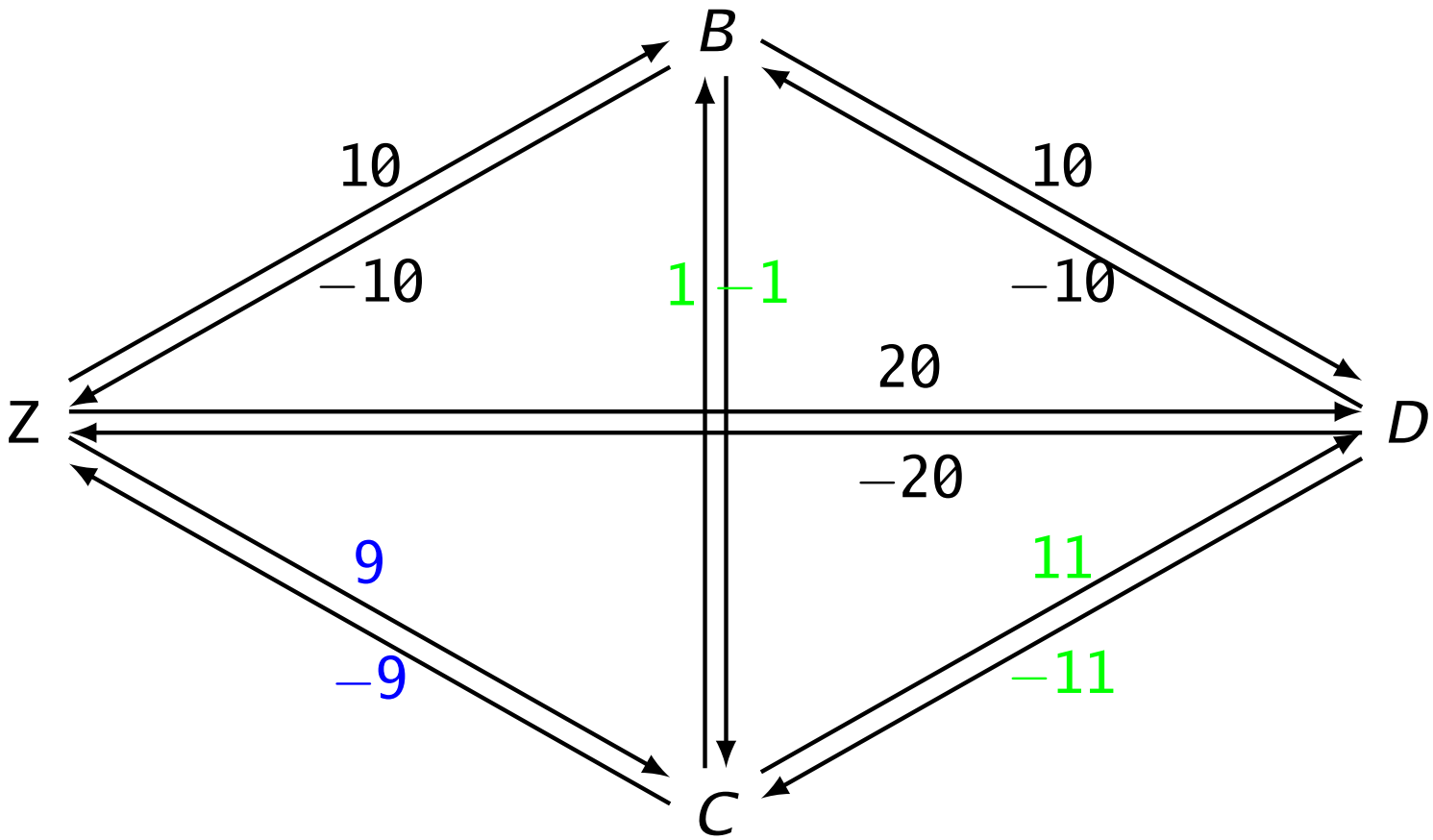
Simple Temporal Network
Executing an STN in real time

Next, select $B = 10$; and update APSP graph:



Remaining Time Windows: $C \in [7, 16]$

Finally, select $C = 9$; and update APSP graph:



Easy to verify that this is a solution.

Simple Temporal Network
Problems that can occur when executing an STN in real time

- May need to go back in time:
  Pick $D = 20$, then after updating, go back in time to pick $B = 10$
  (i.e., no relationship to real–time execution)
- Expensive to update $\mathcal{D}$

Simple Temporal Network
Executing an STN in real time

- Only executed *enabled* time–points (i.e., those having no outgoing negative edges to unexecuted time–points).
- Focus updating on entries involving Z:
  reduces cost to linear time per update, $O(n^2)$ overall.*
- Alternatively, prior to execution, transform STN into *dispatachable* form in $O(n^2 \log n + nm)$ time; then during execution, only need to propagate bounds to *neighboring* time–points.†

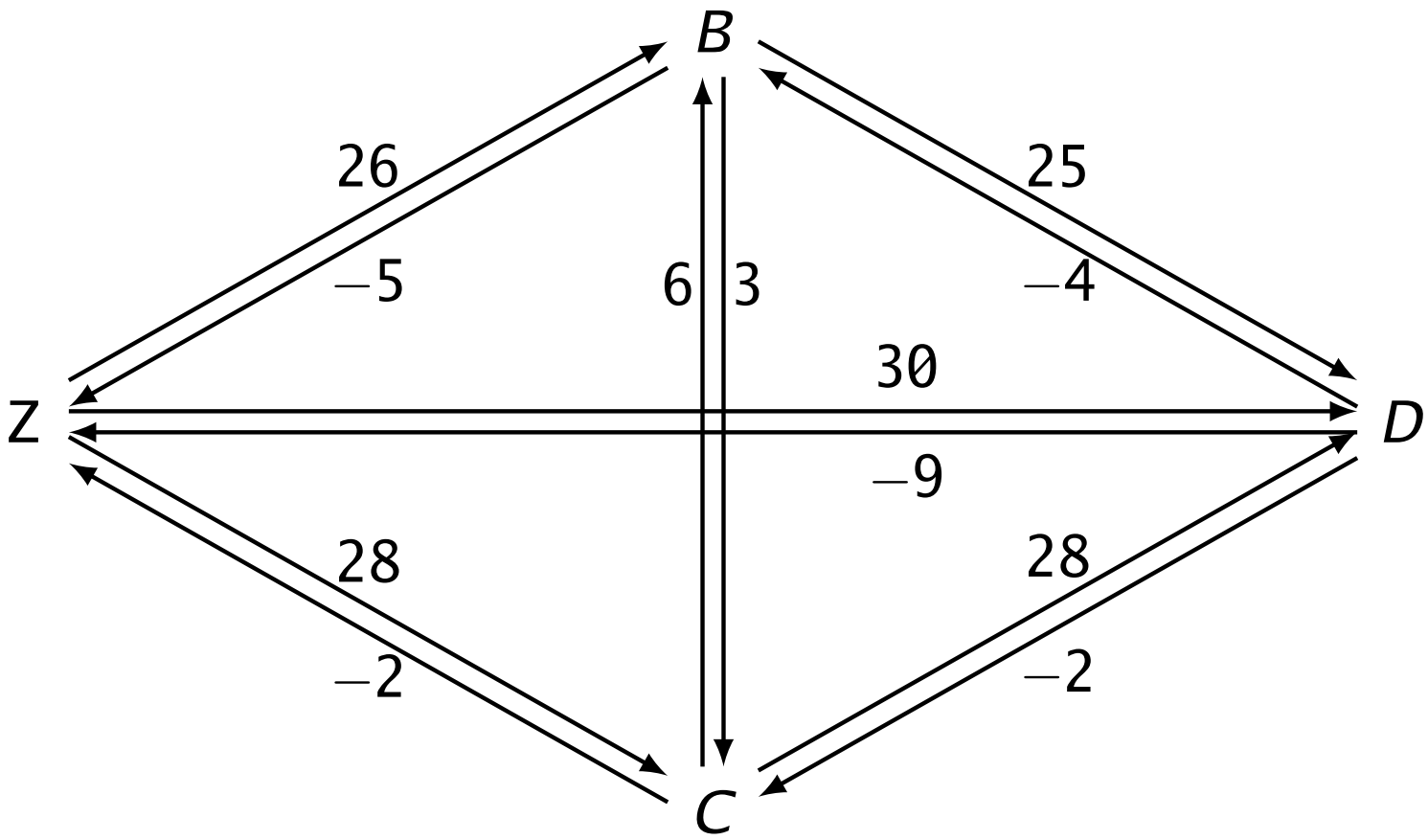
†[Muscettola et al., 1998], †[Tsamardinos et al., 1998]

Simple Temporal Network
Dispatchable STN

An STN $\mathcal{S}$ is *dispatchable* if the following algorithm necessarily successfully executes $\mathcal{S}$:

1. $t := 0$ (curr. time); $\mathcal{X} := \{\}$ (executed); $\mathbf{E} := \{Z\}$ (enabled);
2. Remove any $X \in \mathbf{E}$ such that t is in X's time window;
3. Execute: set $X := t$, and add X to $\mathcal{X}$;
4. Propagate: propagate $t \leq X \leq t$ to X's *immediate neighbors*;
5. Update: update $\mathbf{E}$ to include all $Y \in \mathcal{T} \backslash \mathcal{X}$ for which all negative edges emanating from Y have a destination in $\mathcal{X}$;
6. Wait: wait until t has advanced to some time between $\min\{lb(W) \mid W \in \mathbf{E}\}$ and $\min\{ub(W) \mid W \in \mathbf{E}\}$;
7. If $\mathcal{X} \neq \mathcal{T}$, go back to (2); else done.

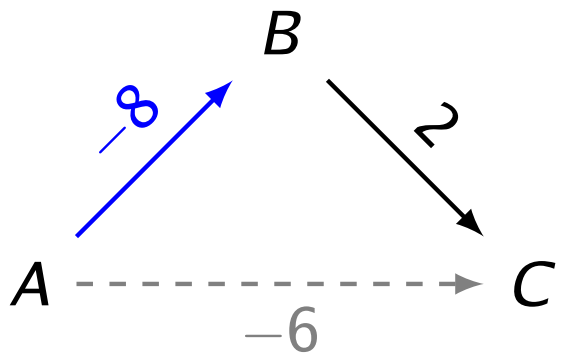
Start with APSP Graph (it is necessarily dispatchable):



Then remove *dominated* edges . . .

Simple Temporal Network
Dominated Edges

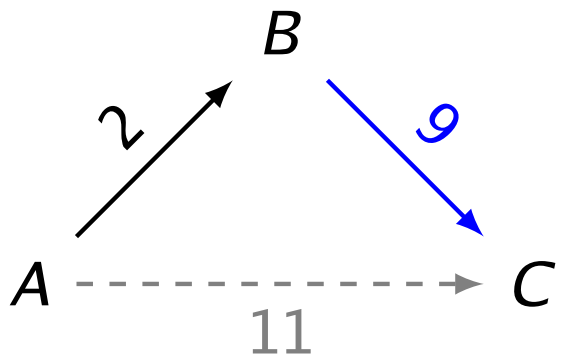
A negative edge AC is dominated by a negative edge AB if $\mathcal{D}(A, B) + \mathcal{D}(B, C) = \mathcal{D}(A, B)$:



- AB and AC have the *same source* node: A.
- During execution, it is not necessary to propagate along dominated edges (e.g., AC).

Simple Temporal Network
Remove Dominated Edges (ctd.)

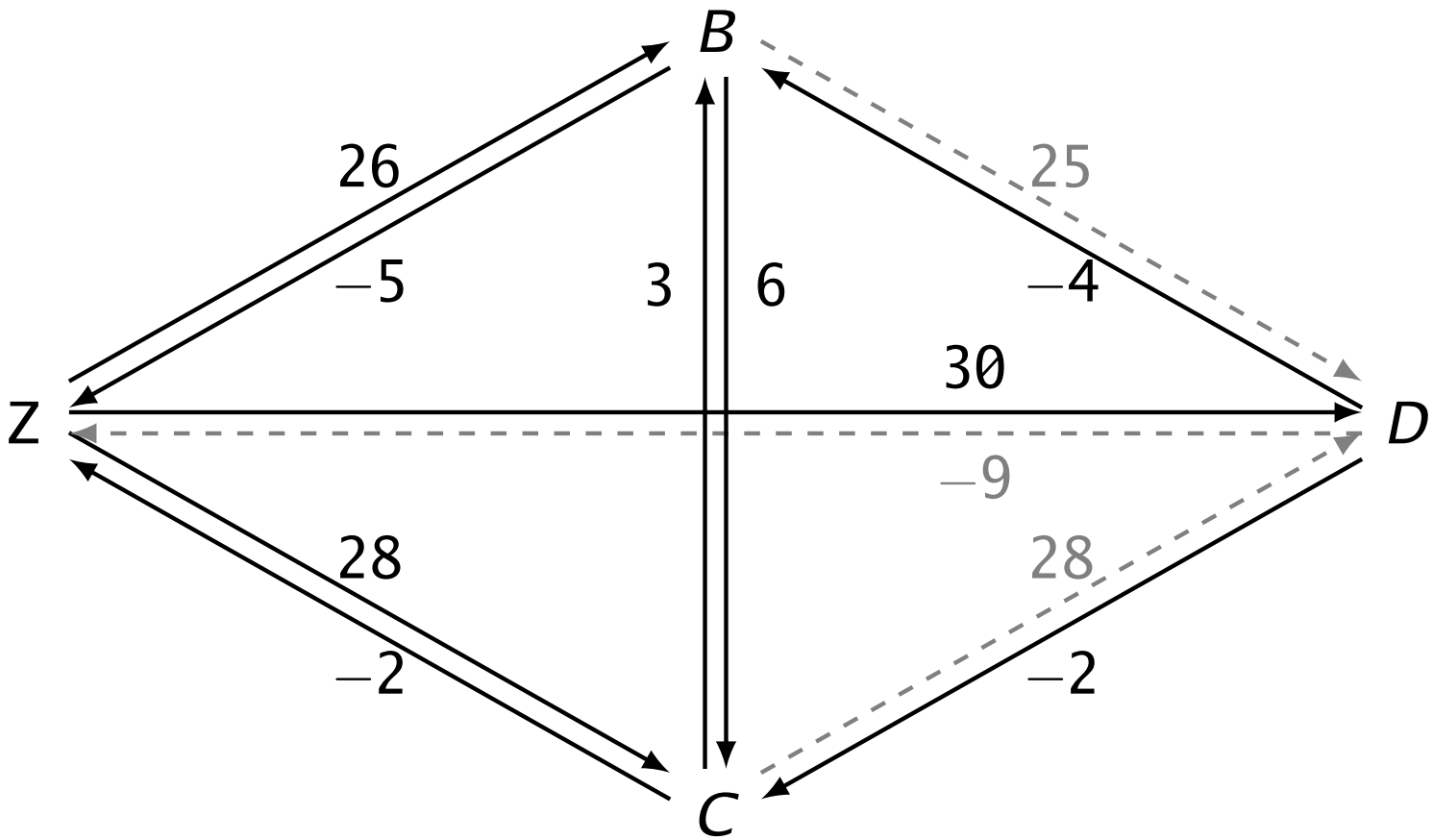
A non–negative edge AC is dominated by a non–negative edge BC if $\mathcal{D}(A,B) + \mathcal{D}(B,C) = \mathcal{D}(A,B)$:



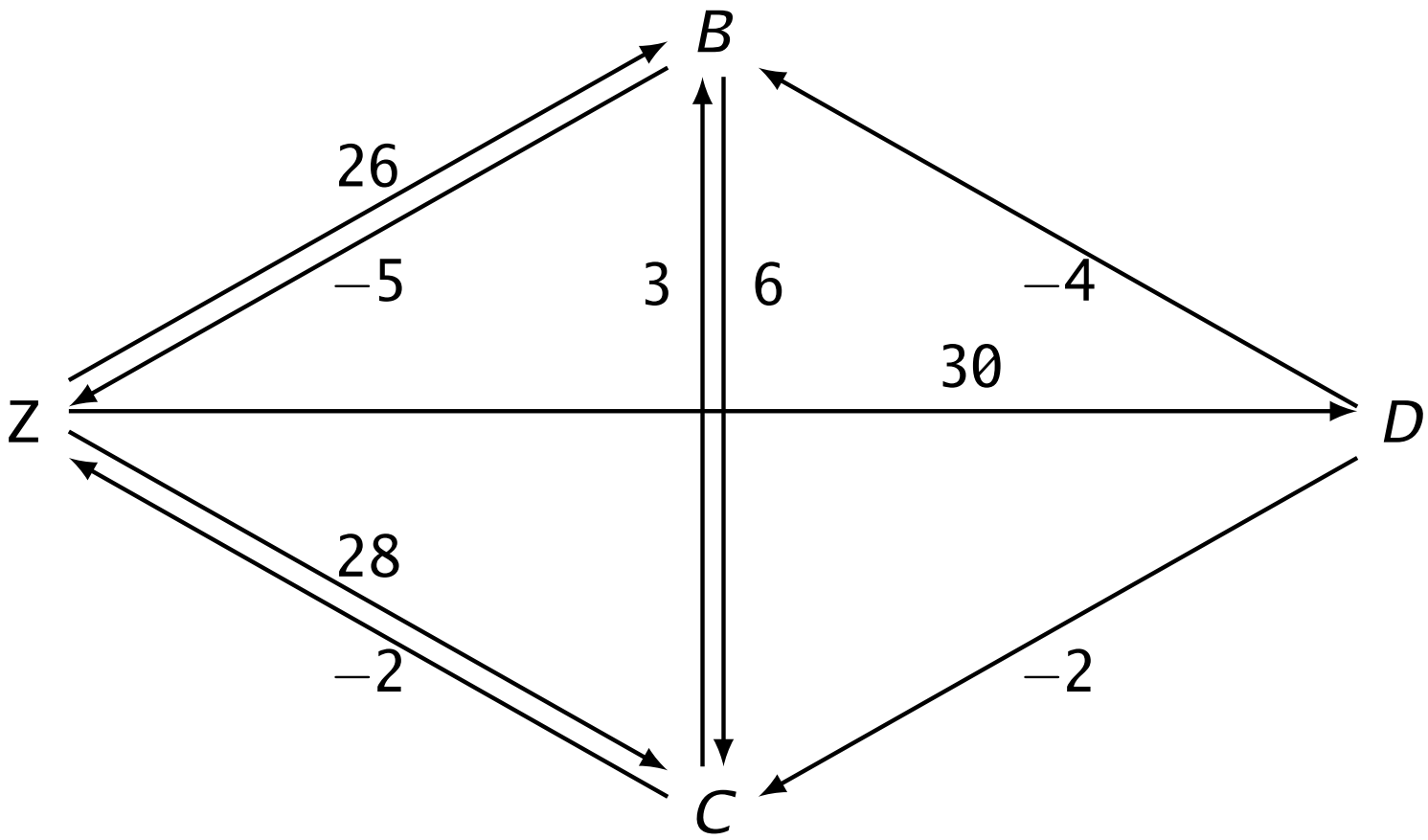
- BC and AC have the *same destination* node: C.
- During execution, it is not necessary to propagate along dominated edges (e.g., AC).

Simple Temporal Network
Making STN Dispatchable (ctd.)

Remove "dominated" edges:*



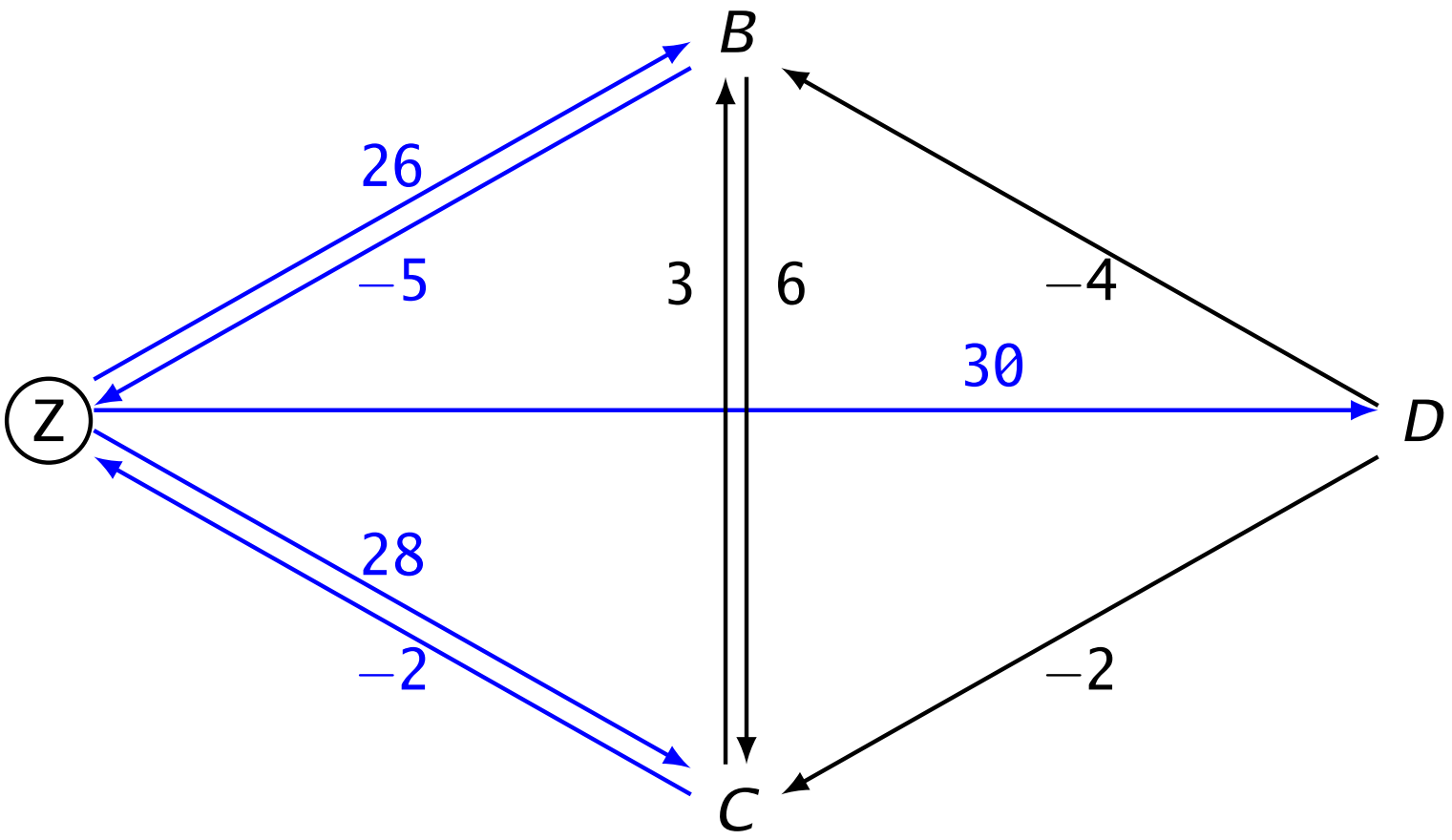
Initially: $t = 0$, $\mathcal{X} = \{\}$, $\mathbf{E} = \{Z\}$.



Remove Z from $\mathbf{E}$. Set $Z = 0$. Add Z to $\mathcal{X}$.

Simple Temporal Network
Dispatching the STN (ctd.)

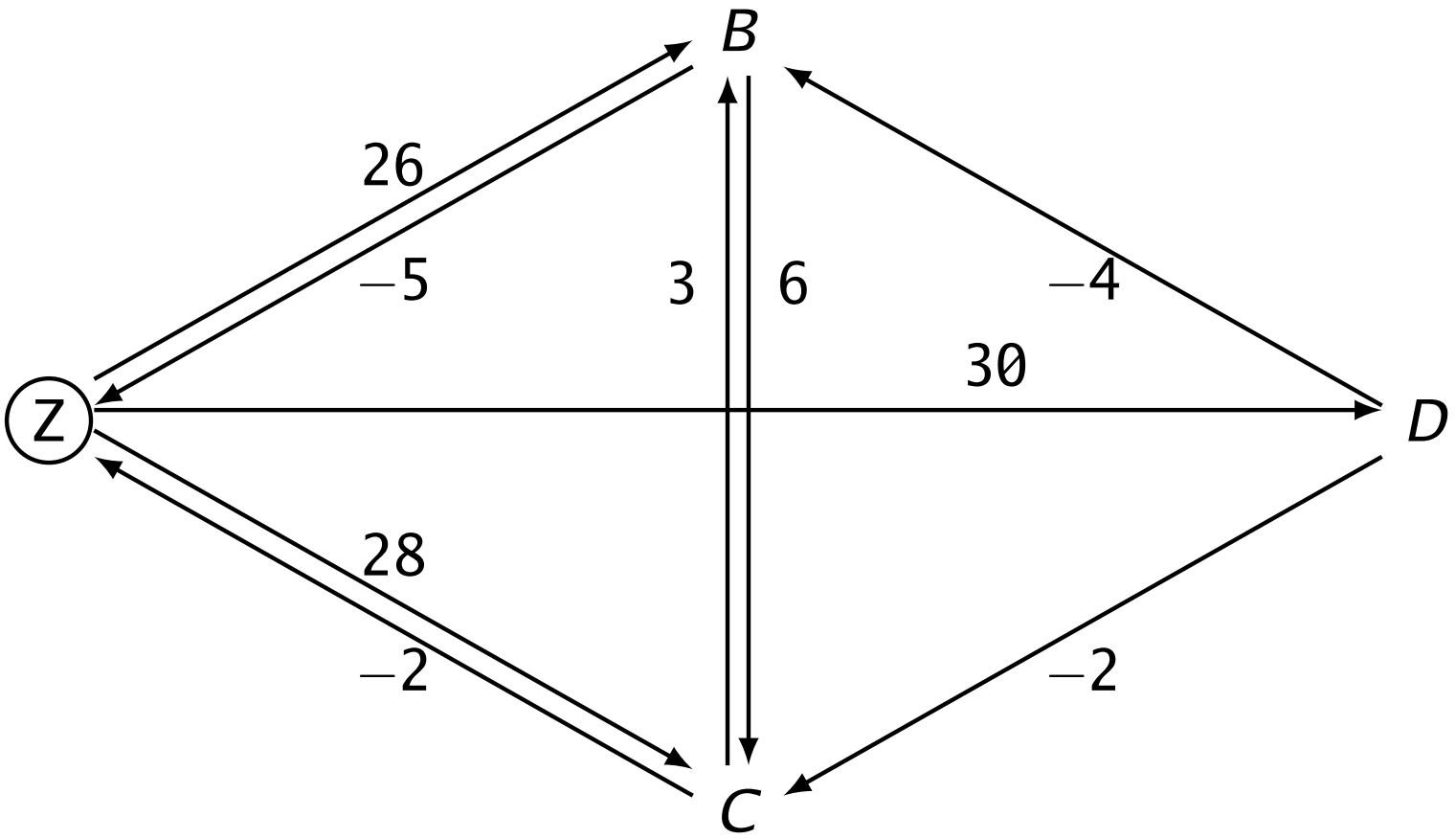
Propagate $Z = 0$ to neighbors;



$\mathcal{X} = \{Z\}$, $\mathbf{E} = \{B, C\}$; $B \in [5, 26], C \in [2, 28], D \in [0, 30]$.

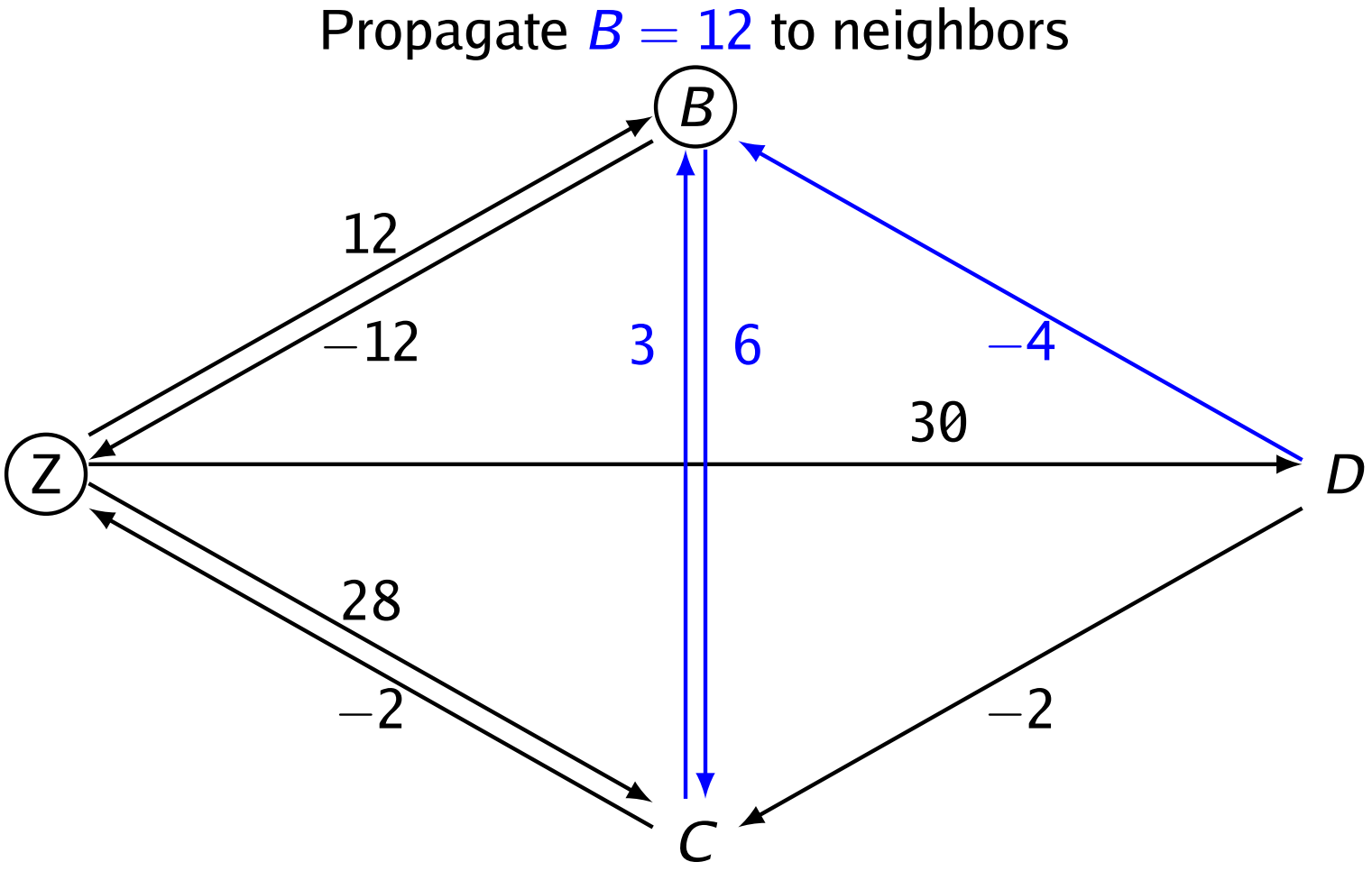
Simple Temporal Network
Dispatching the STN (ctd.)

$\mathcal{X} = \{Z\}$, $\mathbf{E} = \{B, C\}$; Bounds: $B \in [5, 26]$, $C \in [2, 28]$.



Let t advance to 12; Pick B from $\mathbf{E}$; Set $B = 12$.

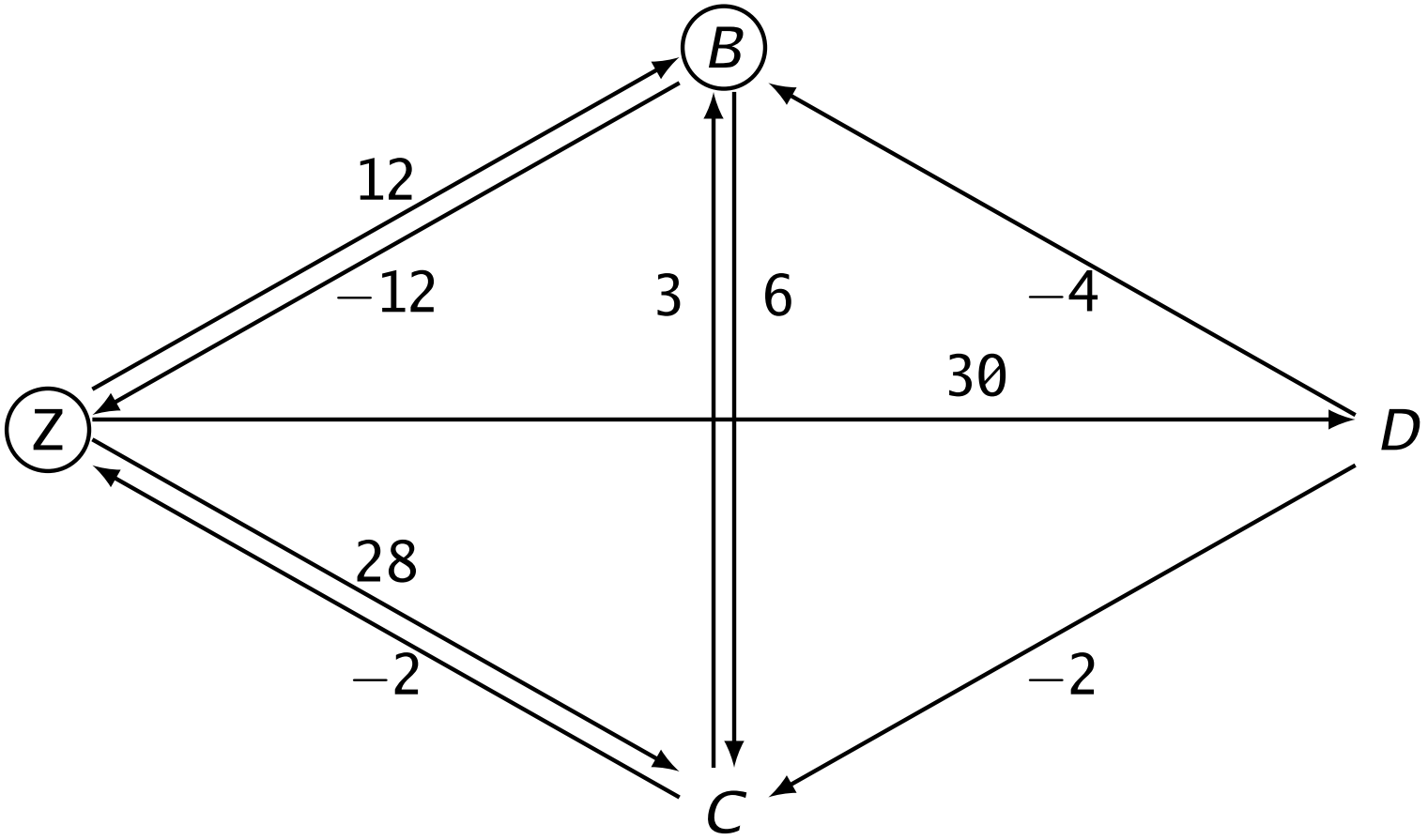
Simple Temporal Network
Dispatching the STN (ctd.)



$\mathcal{X} = \{Z, B\}$, $t = 12$, $\mathbf{E} = \{C\}$, $C \in [12, 18]$, $D \in [16, 30]$

Simple Temporal Network
Dispatching the STN (ctd.)

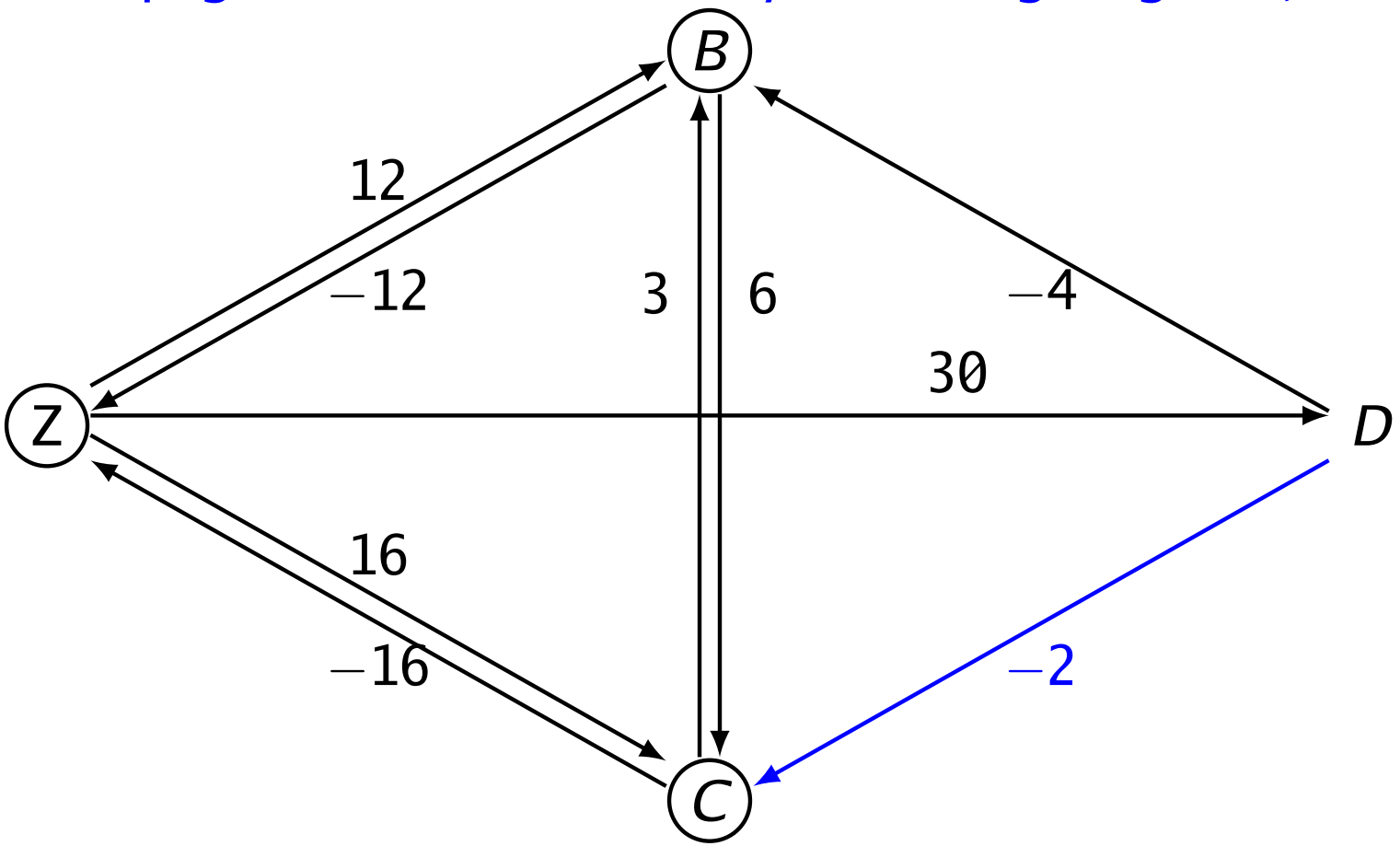
$\mathcal{X} = \{Z, B\}$, $t = 12$, $\mathbf{E} = \{C\}$, $C \in [12, 18]$, $D \in [16, 30]$



Let t advance to 16, pick C from $\mathbf{E}$, set $C = 16$.

Simple Temporal Network
Dispatching the STN (ctd.)

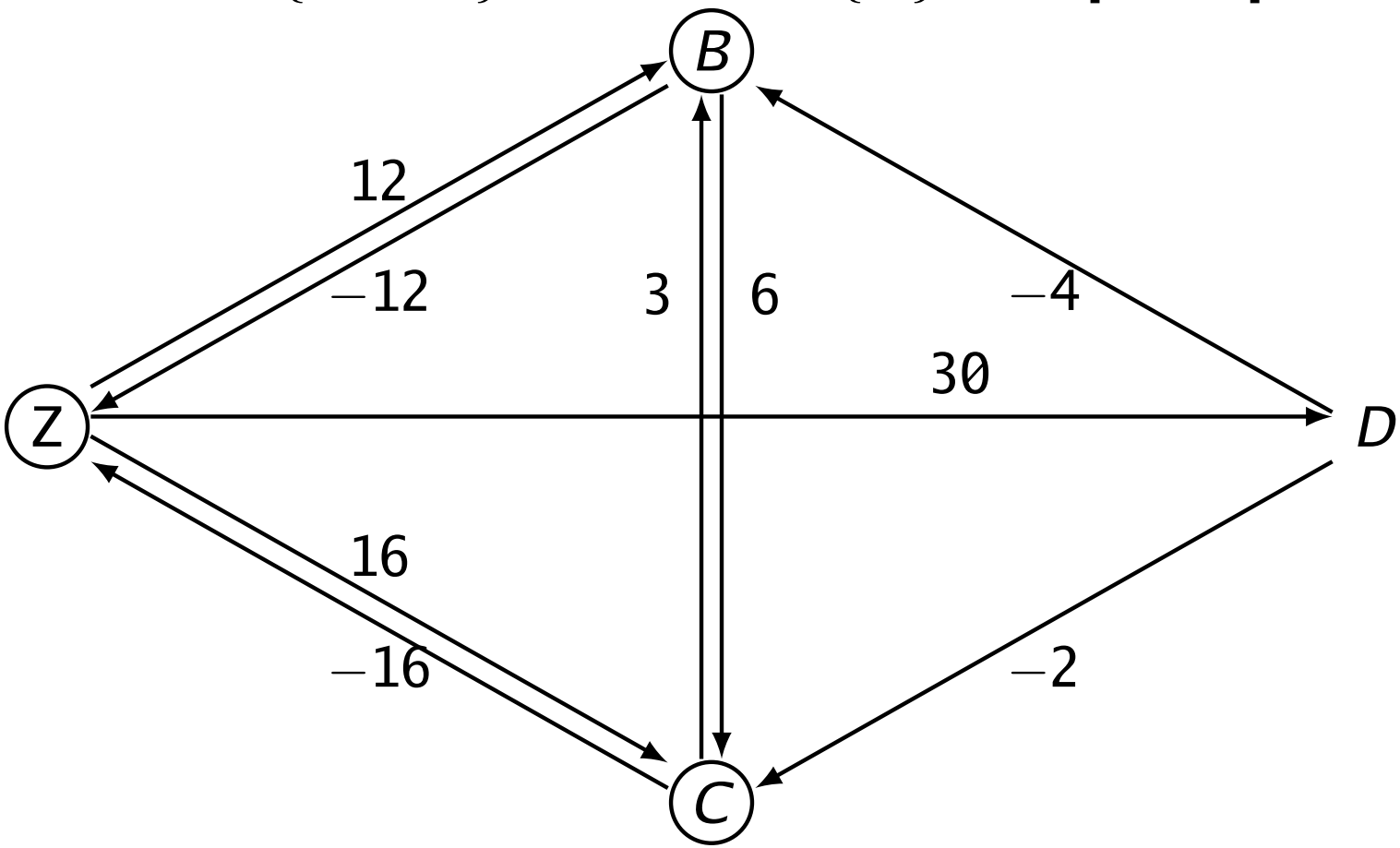
Propagate $C = 16$ to C's only remaining neighbor, D.



$\mathcal{X} = \{Z, B, C\}$, $t = 16$, $\mathbf{E} = \{D\}$, $D \in [18, 30]$

Simple Temporal Network
Dispatching the STN (ctd.)

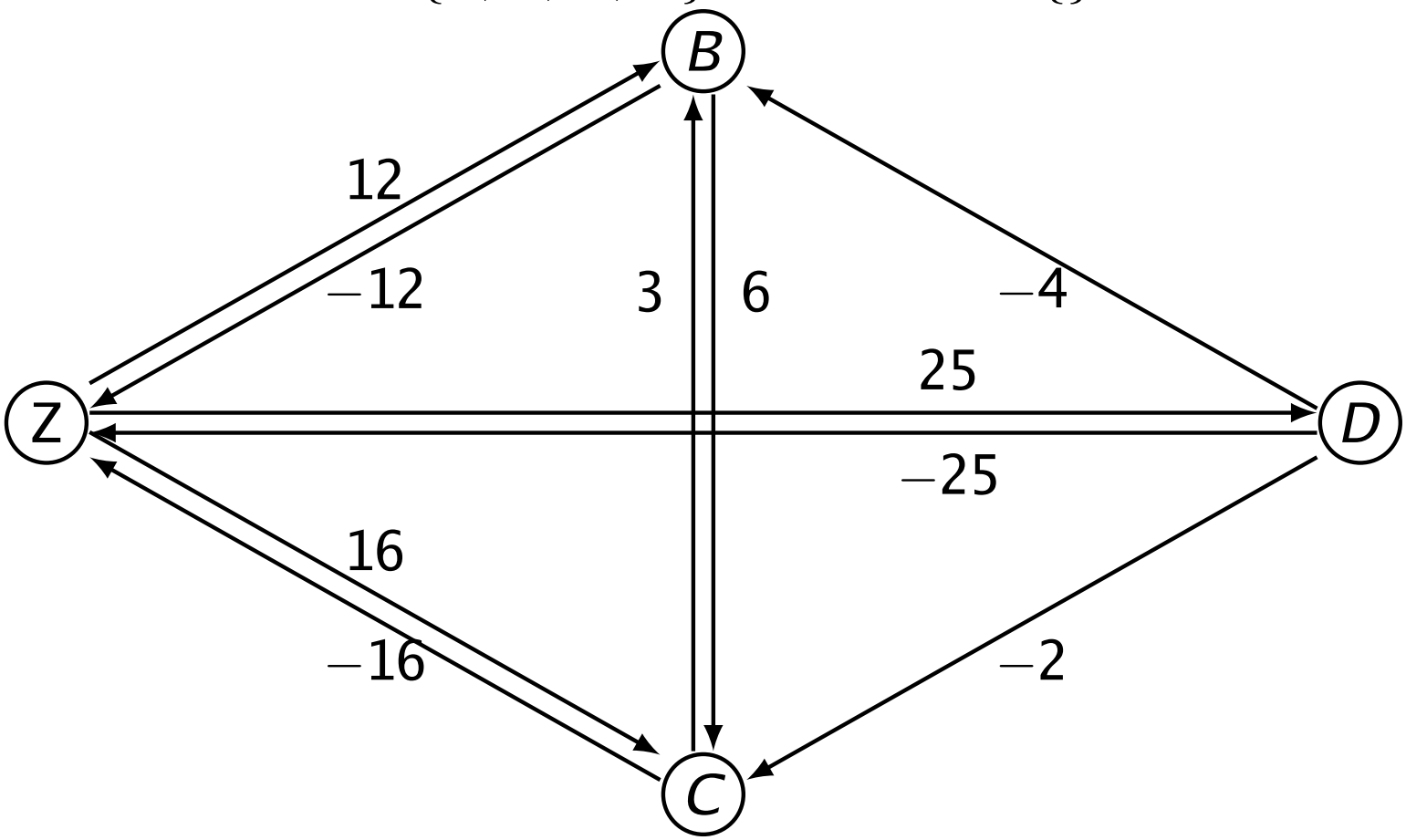
$\mathcal{X} = \{Z, B, C\}$, $t = 16$, $\mathbf{E} = \{D\}$, $D \in [18, 30]$



Let t advance to 25, pick D from $\mathbf{E}$, set $D = 25$.

Simple Temporal Network
Dispatching the STN (ctd.)

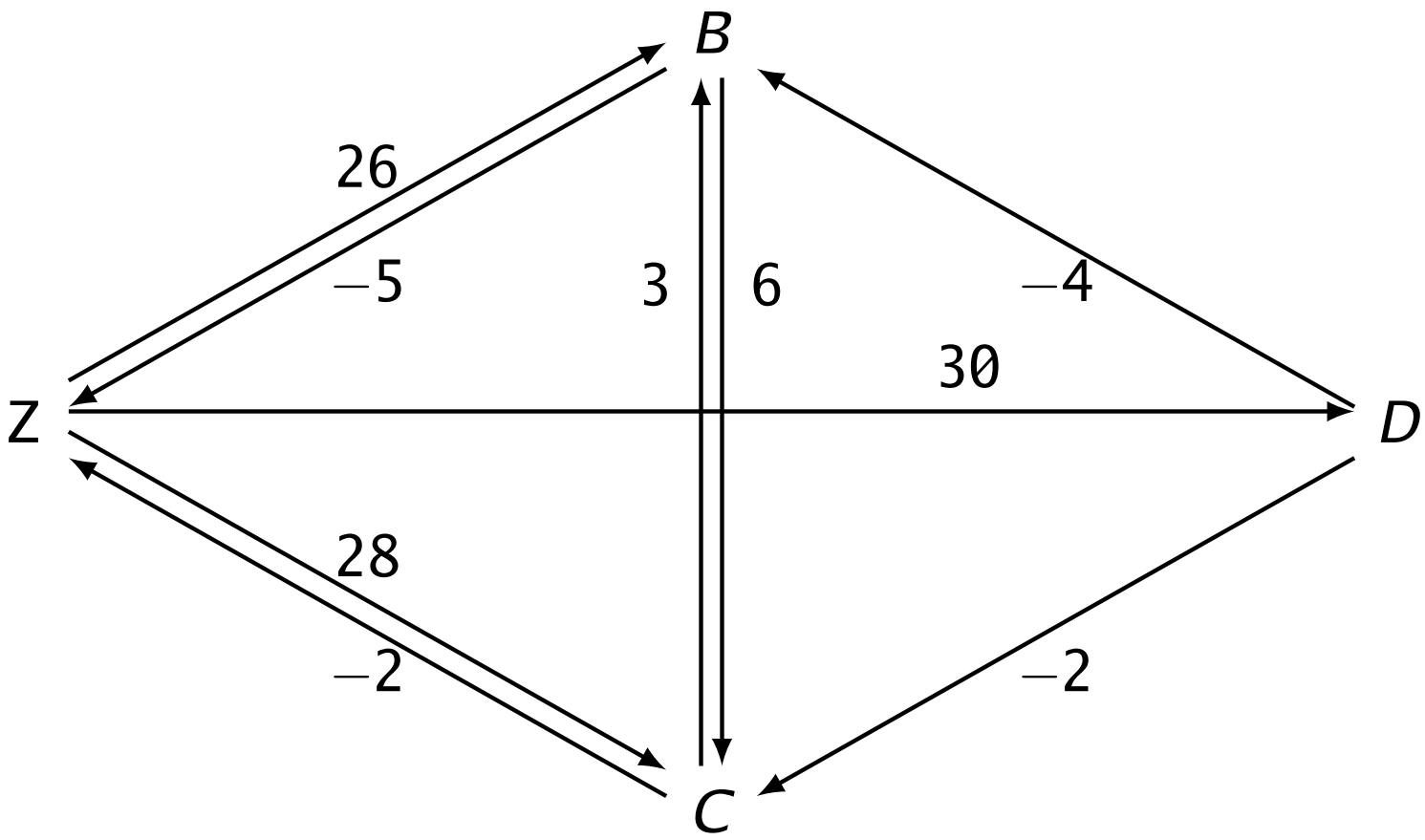
$\mathcal{X} = \{Z, B, C, D\}$, $t = 25$, $\mathbf{E} = \{\}$



Solution: $Z = 0, B = 12, C = 16, D = 25$.

Simple Temporal Network
Dispatching the STN (ctd.)

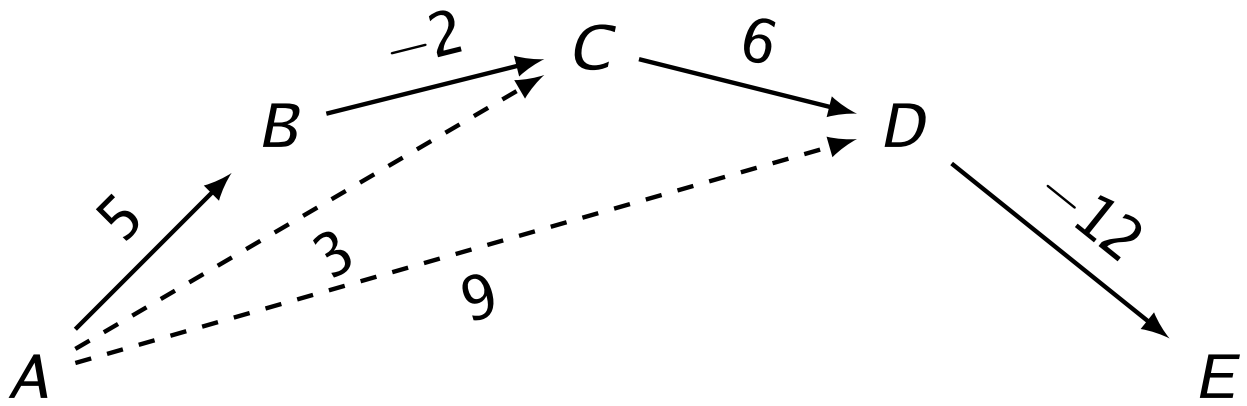
Easy to check that $Z = 0, C = 20, B = 23, D = 28$ can also be generated by the dispatcher.



Simple Temporal Network
New View of Dispatchability*

(1) A path $\mathcal{P}$ has the prefix/postfix (PP) property if:

- Every proper prefix of $\mathcal{P}$ has non–negative length, and
- Every proper postfix of $\mathcal{P}$ has negative length.

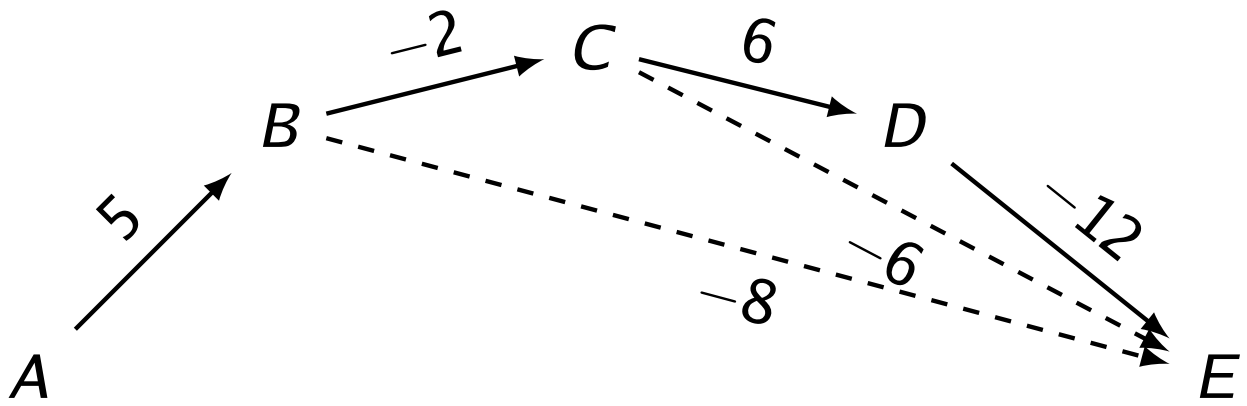


* [Morris, 2014]

Simple Temporal Network
New View of Dispatchability*

(1) A path $\mathcal{P}$ has the prefix/postfix (PP) property if:

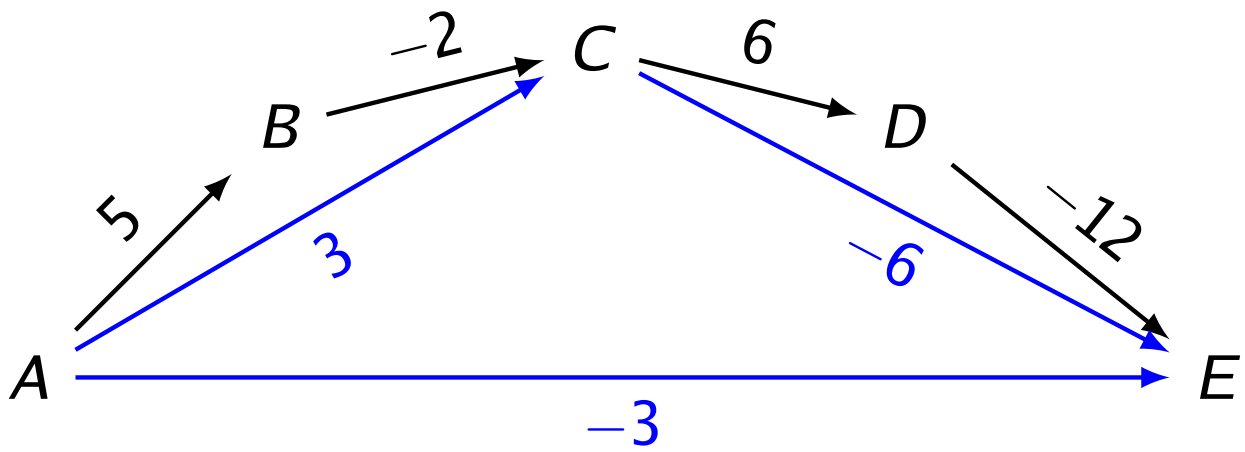
- Every proper prefix of $\mathcal{P}$ has non–negative length, and
- Every proper postfix of $\mathcal{P}$ has negative length.



* [Morris, 2014]

Simple Temporal Network
New View of Dispatchability (ctd)

(2) An STN is PP–complete if for each shortest path from any X to any Y that has the prefix/postfix property, there is an edge from X to Y with the same length.



(3) **A consistent and PP–complete STN is dispatchable.**

* [Morris, 2014]

Further graphical analyses of the dispatchability of STNs has been presented recently [Morris, 2016].

Additional Research on STNs

- Temporal Decoupling Problem (TDP) [Hunsberger, 2002; Jr. and Durfee, 2013; Mountakis et al., 2017]
- APSP algorithms on *chordal graphs* [Xu and Choueiry, 2003]
- Enforcing partial path consistency [Planken, 2008]
- Incorporating STNs in multi–agent auctions [Hunsberger and Grosz, 2000]
- For further info:
  `http://www.cs.vassar.edu/~hunsberg/__papers__/`
  (See pages 22–24, 27–28, 32, 53–70, 76–79 of my 2005 AAMAS tutorial.)

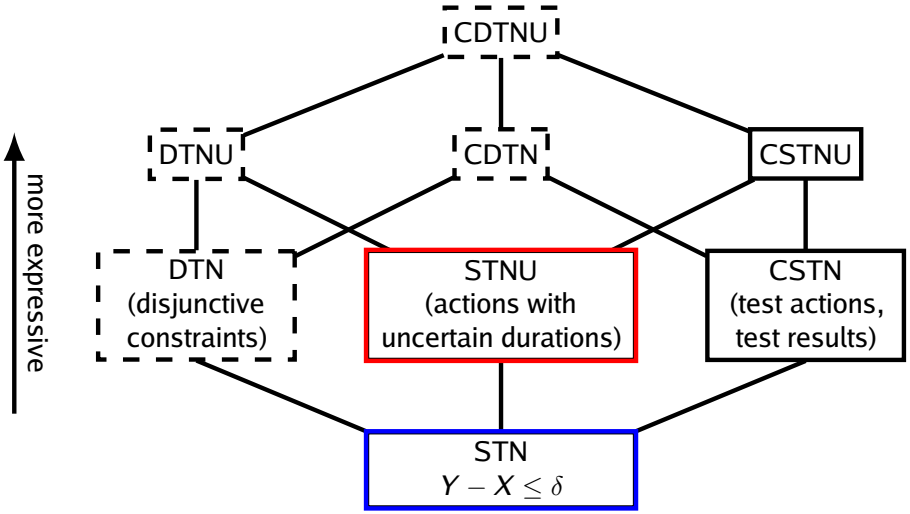
STN Summary

- STNs have been used to provide flexible planning and scheduling systems for more than a decade.
- Efficient algorithms for checking consistency, incrementally updating the APSP matrix, and managing execution in real time for maximum flexibility.
- However, STNs cannot represent uncertainty (e.g., actions with uncertain durations) or conditional constraints (e.g., only do X if test result is negative).

Simple Temporal Networks with Uncertainty
Motivation

- You may control when an action starts, but not how long it lasts:
  — taxi ride, bus ride, baseball game, medical procedure.
- Although their durations may be uncertain, they are often within known bounds.
- Such actions can be represented by *contingent links* in a temporal network . . .

STN with Uncertainty*



An STNU is a triple,

$\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, where:

- $(\mathcal{T}, \mathcal{C})$ is an STN
- $\mathcal{L}$ is a set of contingent links, each of the form (A, x, y, C):
  - A is the activation time–point.
  - C is the contingent time–point.
  - Duration bounded: $C - A \in [x, y]$ but *uncontrollable*

* [Morris et al., 2001]

STN with Uncertainty
STNU Graph

- Nodes and edges as in an STN graph

$$Y - X \in [3, 7] \iff X \underset{-3}{\overset{7}{\rightleftarrows}} Y \iff X \xrightarrow{[3,7]} Y$$

- Contingent Links $\iff$ Labeled Edges*

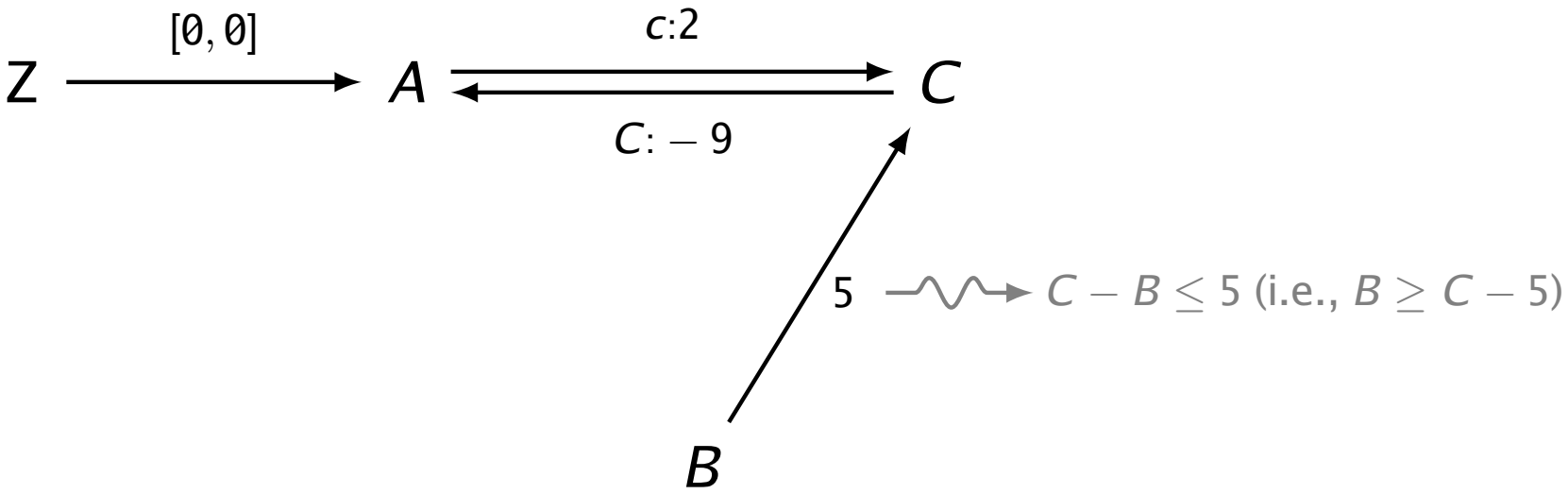
$$(A, 3, 7, C) \iff A \underset{C:-7}{\overset{c:3}{\rightleftarrows}} C \iff X \xRightarrow{[3,7]} Y$$

  Labeled edges represent uncontrollable possibilities.

- The lower–case edge, $A \xrightarrow{c:3} C$, represents the uncontrollable possibility that $C - A$ might equal 3.
- The upper–case edge, $A \xleftarrow{C:-7} C$, represents the uncontrollable possibility that $C - A$ might equal 7.

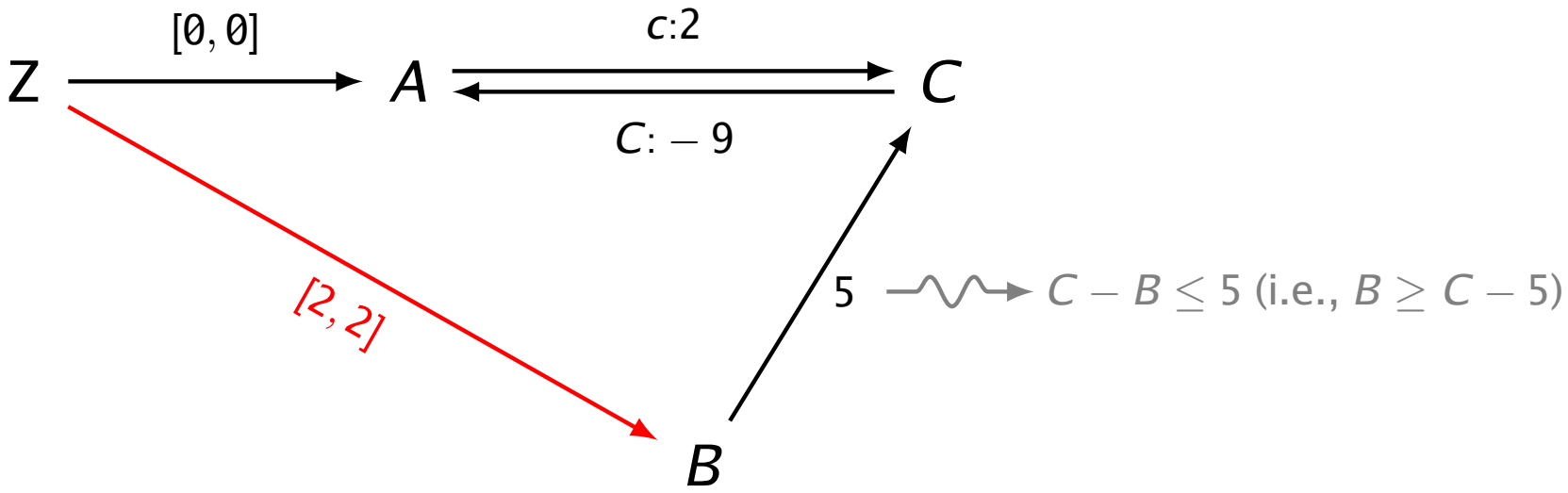
* [Morris and Muscettola, 2005]

STN with Uncertainty
STNU Example



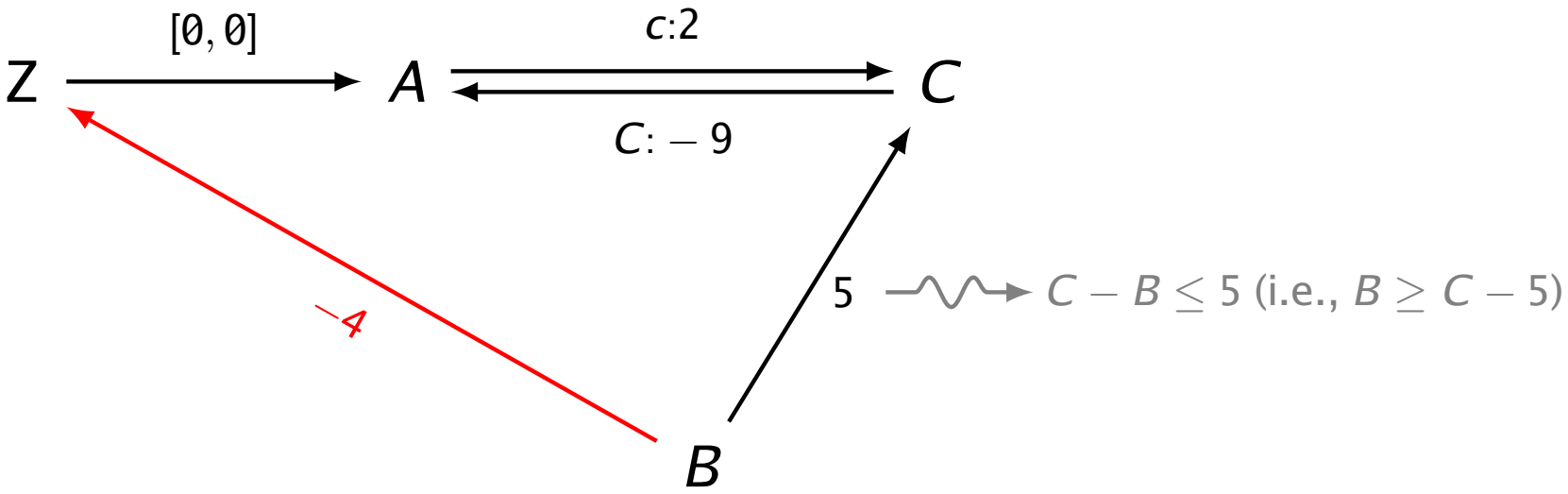
If $A = 0$, when is it safe to execute B?

STN with Uncertainty
STNU Example



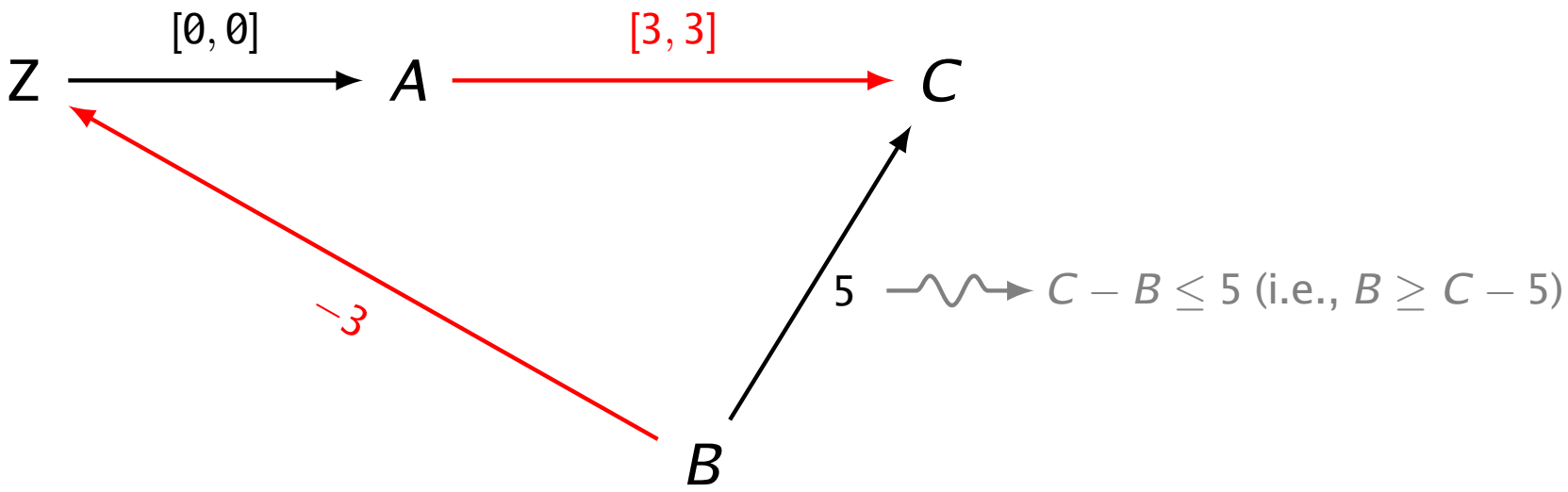
If $A = 0$ and $B = 2$, then problem arises if $C > 7$.

STN with Uncertainty
STNU Example



If $A = 0$ and $B \geq 4$, then no problems!

STN with Uncertainty
STNU Example



If $A = 0$ and $C = 3$, then $B \geq 3$ no problem!

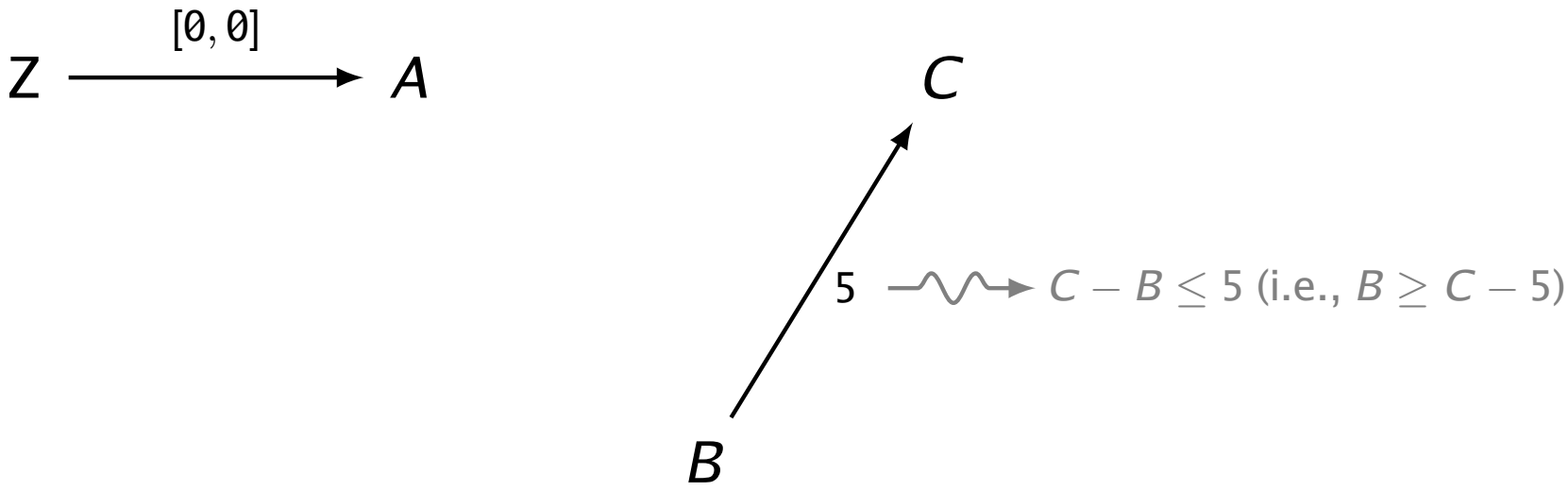
STN with Uncertainty
Dynamic Controllability (DC)

An STNU is *dynamically controllable* (DC) if:

- there exists a *dynamic strategy* ...
- for executing the *non–contingent* time–points ...
- such that *all* of the constraints will be satisfied ...
- *no matter how the contingent durations turn out.*

A dynamic strategy can *react* to contingent executions.

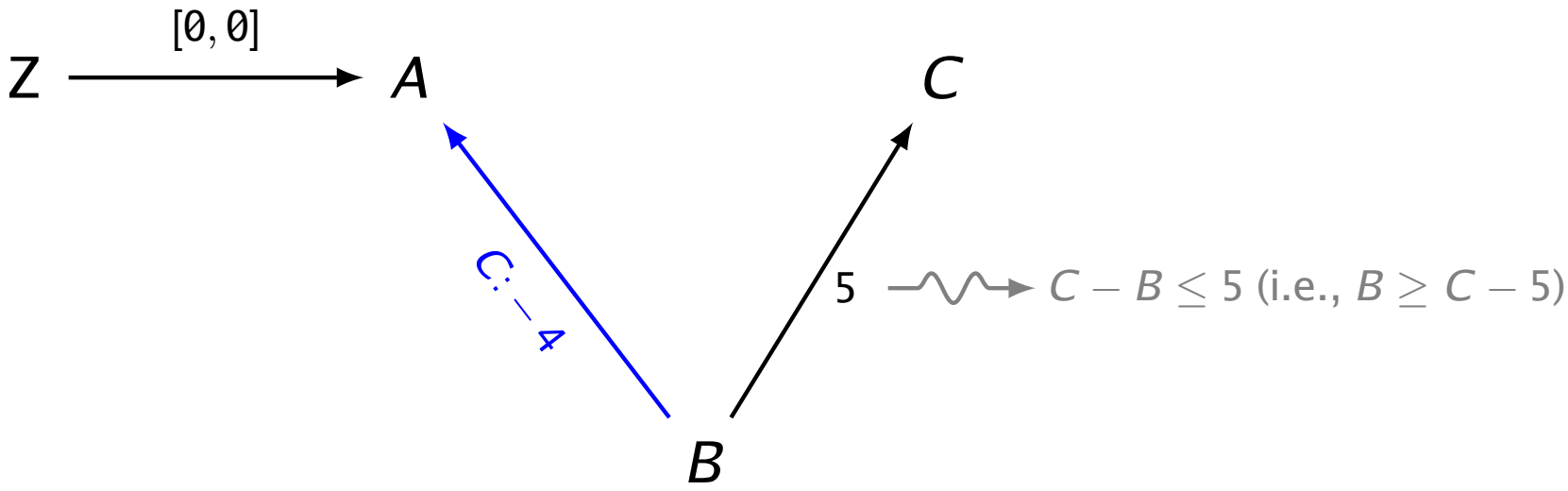
STN with Uncertainty
STNU Example



Strategy: While C unexecuted, B must wait at least 4 after A.

Wait: $A \xleftarrow{C: -x} B$ is a wait constraint when the source node (B) is different from the upper–case label (C).

STN with Uncertainty
STNU Example



Strategy: While C unexecuted, B must wait at least 4 after A.

Wait: $A \xleftarrow{C:-x} B$ is a wait constraint when the source node (B) is different from the upper–case label (C).

Dynamic Controllability—Original MMV Semantics*

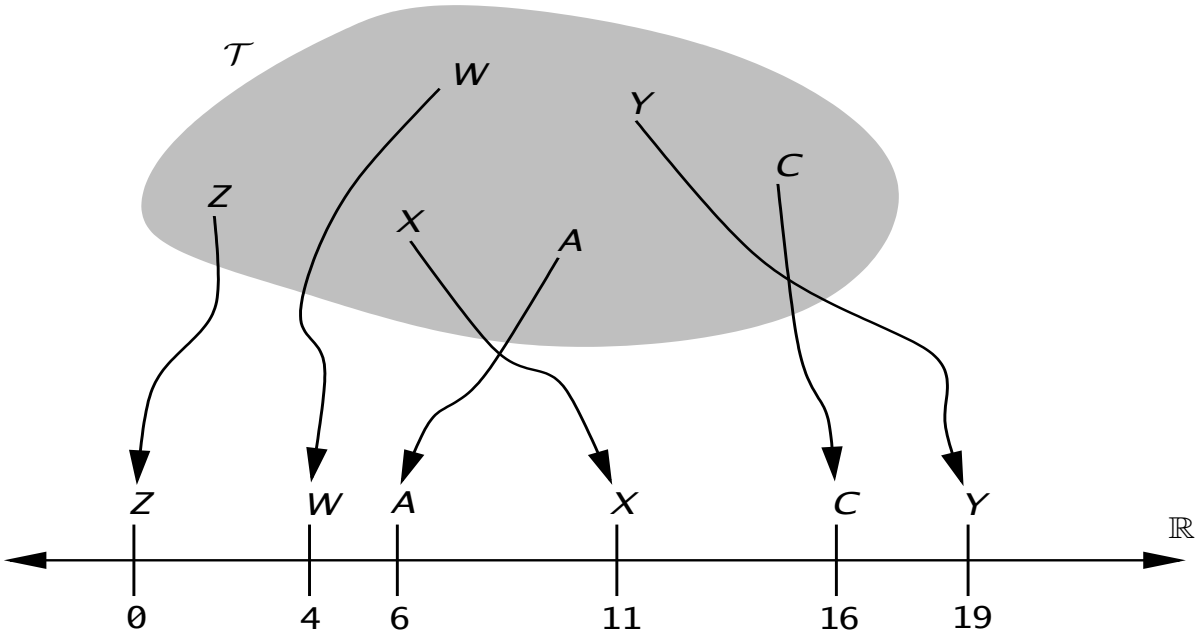
Schedules, situations, and strategies

- Schedule: Assigns times to all time–points
- Situation: Specifies durations of all contingent links
- Strategy: Mapping from (complete) situation to (complete) schedule
- But execution happens incrementally, based on current, partial view of "real" situation
- Therefore, need to carefully restrict strategies to ensure that their decisions do not depend on advance knowledge of future events.

* [Morris et al., 2001]

Dynamic Controllability
Schedules*

- $\psi : \mathcal{T} \to \mathbb{R}$ is a *schedule*; and Ψ is the set of all schedules for $\mathcal{S}$.
- $[\psi]_X$ = time of X in schedule ψ

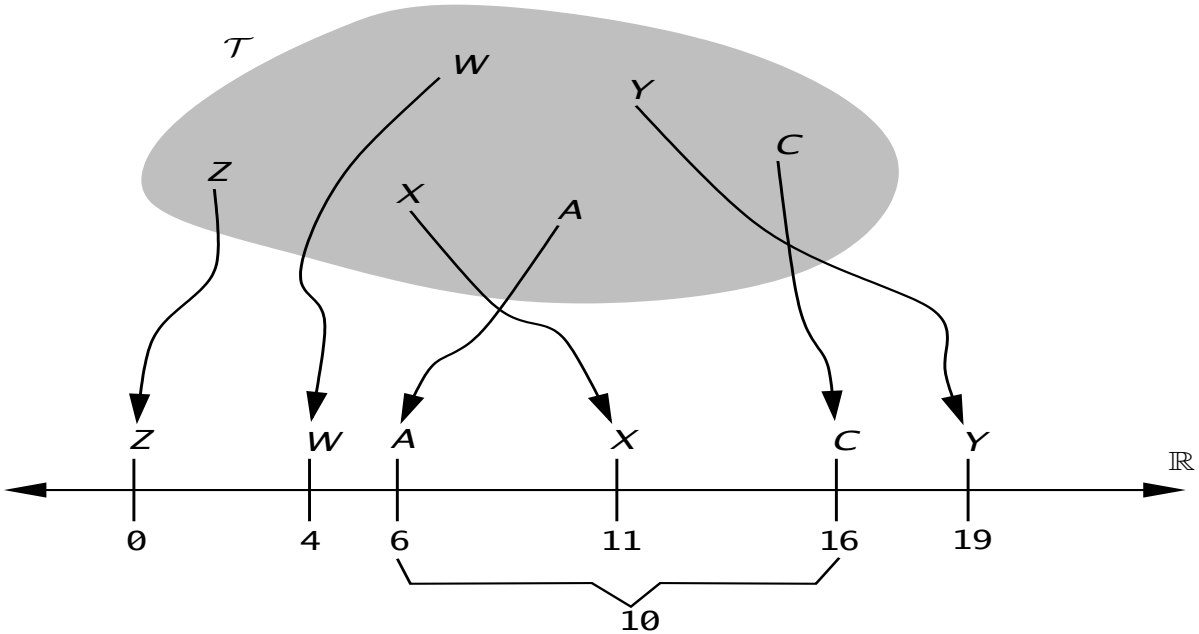


$$[\psi]_Z = 0, \quad [\psi]_W = 4, \quad [\psi]_A = 6, \quad [\psi]_X = 11, \quad \ldots$$

* [Morris et al., 2001]

Dynamic Controllability
Pre–histories*

- For any $t \in \mathbb{R}$, $[\psi]^{<t}$ = the *pre–history* of ψ relative to time t.†
  - — specifies the durations of all contingent links that finished before time t in the schedule ψ.



$$[\psi]^{<5} = \emptyset, \quad [\psi]^{<12} = \emptyset, \quad [\psi]^{<19} = \{(C - A = 10)\}$$

* [Morris et al., 2001], † [Hunsberger, 2009]

Dynamic Controllability
Situations and projections*

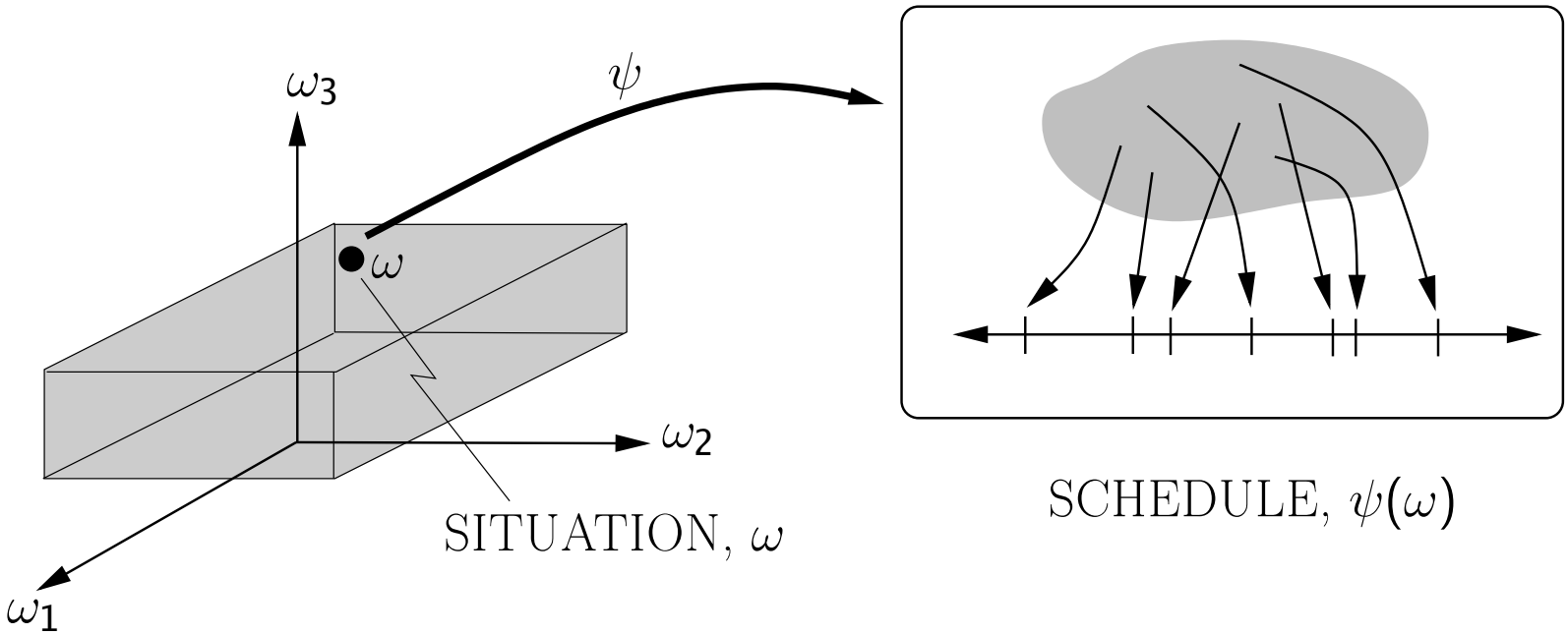
For an STNU with k contingent links, (A_i, x_i, y_i, C_i):

- $\omega = (d_1, \ldots, d_k)$ is a *situation*, where $x_i \leq d_i \leq y_i$, for each i.
- $\Omega = [x_1, y_1] \times \ldots \times [x_k, y_k]$ is the *space of situations*.
- Projection: $\mathcal{S}_\omega$ is the *STN* that results from forcing the contingent links to take on the values in ω.

* [Morris et al., 2001]

Dynamic Controllability
Execution strategies*

A *strategy* is a mapping, $\sigma : \Omega \rightarrow \Psi$, from (complete) situations to (complete) schedules.



* [Morris et al., 2001]

Dynamic Controllability
Valid execution strategies*

- Given a (complete) situation ω, $\sigma(\omega)$ is a (complete) schedule.
- A strategy σ is *valid* if for each situation ω, the schedule $\sigma(\omega)$ is consistent with the projection $\mathcal{S}_\omega$.
- In other words, a valid strategy does not control the durations of contingent links.

* [Morris et al., 2001]

Dynamic Controllability

Dynamic execution strategies*

An execution strategy σ is *dynamic* if:

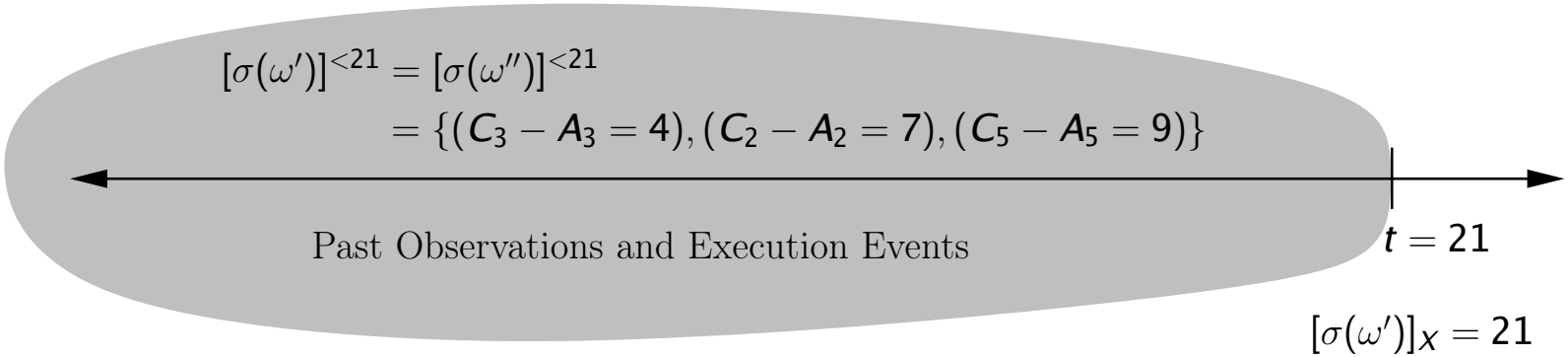
- for any situations ω' and ω'',
- and any non–contingent time–point $X \in \mathcal{T}$,
- let $t = [\sigma(\omega')]_X$

  if $[\sigma(\omega')]^{<t} = [\sigma(\omega'')]^{<t}$, then $[\sigma(\omega'')]_X = t$.

In other words, if σ executes X at time t in situation ω', and the pre–histories of $\sigma(\omega')$ and $\sigma(\omega'')$ are the same (i.e., at time t the agent cannot distinguish between the situations ω' and ω''), then σ must also execute X at time t in ω''.

* [Morris et al., 2001]

Dynamic Controllability
Dynamic execution strategies



σ executes X at time 21 in situation ω'; and $[\sigma(\omega')]^{<21} = [\sigma(\omega'')]^{<21}$.

Therefore, σ must also execute X at time 21 in situation ω''.

Dynamic Controllability—at last!*

An STNU, $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, is *dynamically controllable* (DC) if there exists a strategy for $\mathcal{S}$ that is both *valid* and *dynamic*.

* [Morris et al., 2001]

Pause!

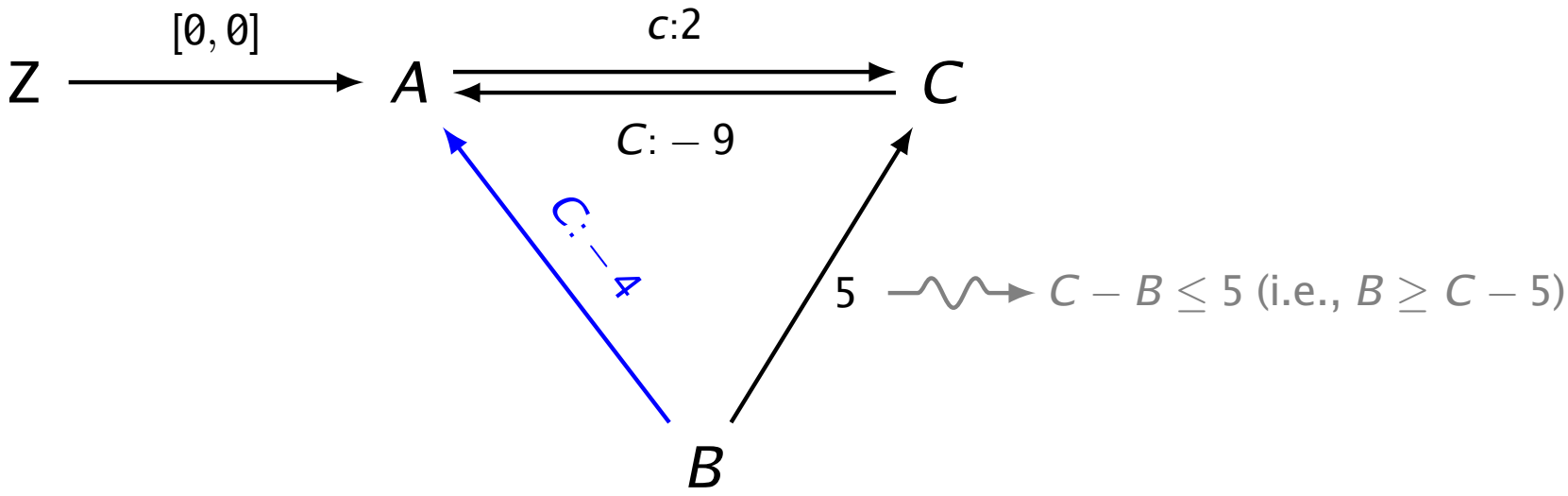
Dynamic Controllability — Take 2

Alternative Semantics: Real–Time Execution Decisions*

- The semantics for dynamic controllability can be stated equivalently in terms of *Real–Time Execution Decisions* (RTEDs):
  - WAIT:
    Wait for some activated contingent link to complete.
  - (t, χ):
    If nothing happens before time $t \in \mathbb{R}$*, then execute the (non–contingent) time–points in* χ *at time* t*.*
- An RTED–based strategy maps each *respectful partial schedule* to an *allowable* RTED.
- By construction, an RTED–based strategy is necessarily dynamic.

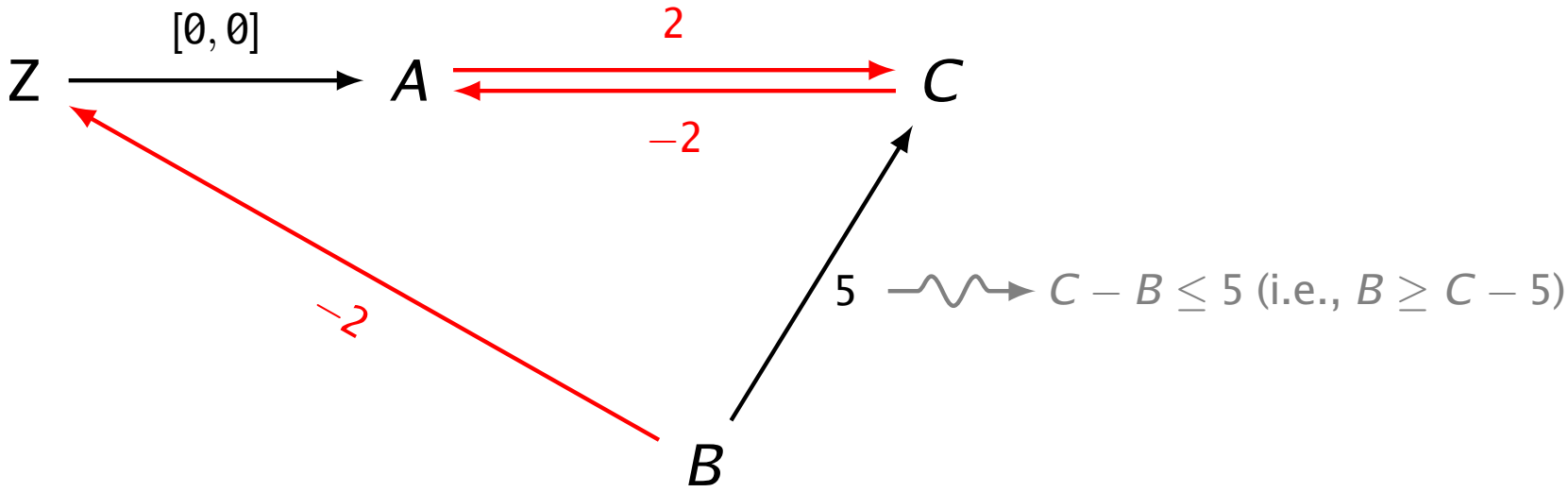
* [Hunsberger, 2009]

STN with Uncertainty
RTED First Example



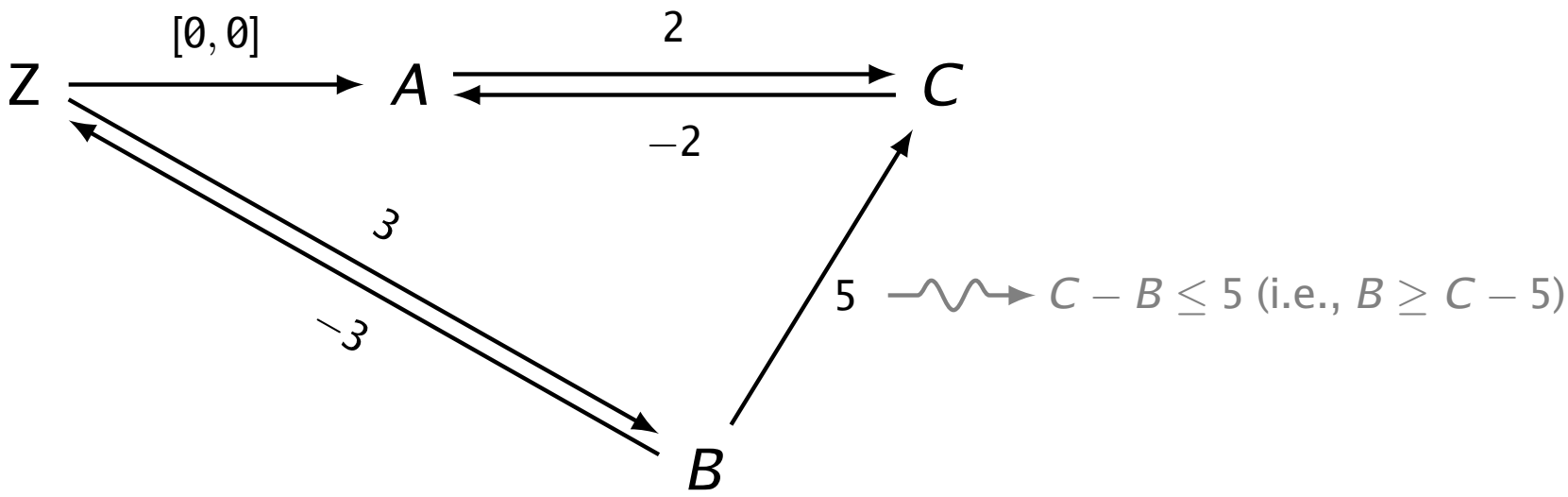
- Initial Partial Schedule: $\{(Z = 0)\}$
- Initial RTED: $(4, \{B\})$
  (If nothing happens before time 4, execute B at 4.)

STN with Uncertainty
RTED First Example



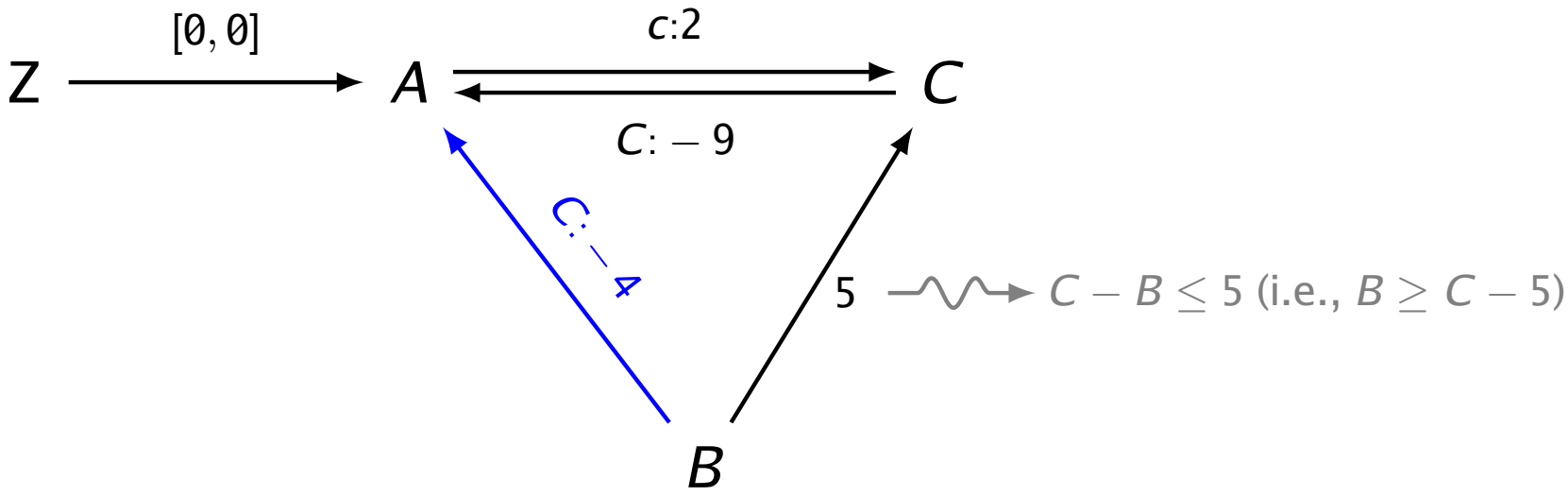
- Possible Outcome: C executes at time 2.
- Next Partial Schedule: $\{(Z = 0), (C = 2)\}$.
- Next RTED: $(3, \{B\})$
  (If nothing happens before time 3, execute B at 3.)

STN with Uncertainty
RTED First Example



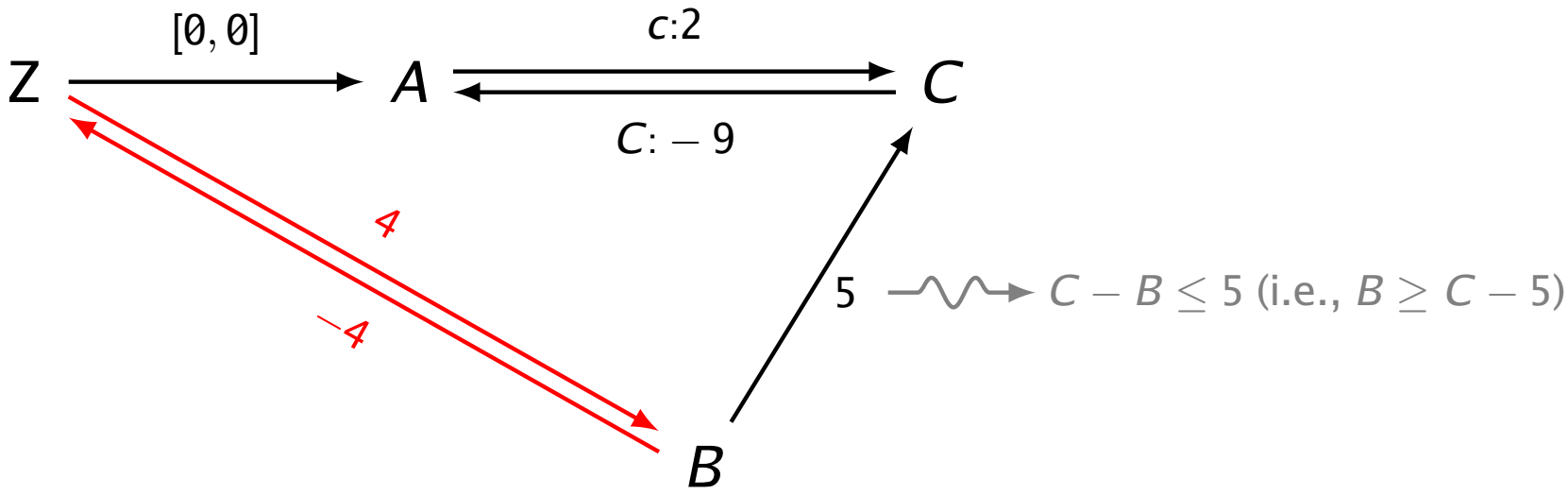
- Possible Outcome: B executes at 3.
- Final Schedule: $\{(Z = 0), (C = 2), (B = 3)\}$.

STN with Uncertainty
RTED Second Example



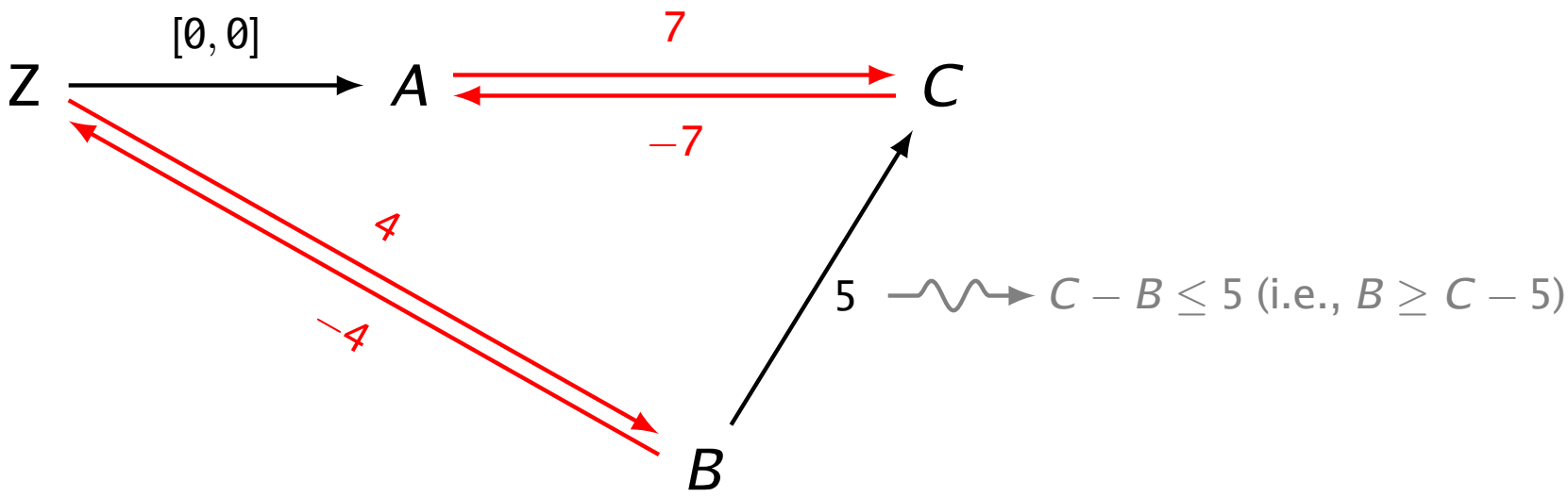
- Initial Partial Schedule: $\{(Z = 0)\}$
- Initial RTED: $(4, \{B\})$
  (If nothing happens before time 4, execute B at 4.)

STN with Uncertainty
RTED Second Example



- Possible Outcome: Time 4 arrives, but C has not yet executed. So B is executed at 4
- Next Partial Schedule: $\{(Z = 0), (B = 4)\}$.
- Next RTED: WAIT (for C to execute).

STN with Uncertainty
RTED Second Example



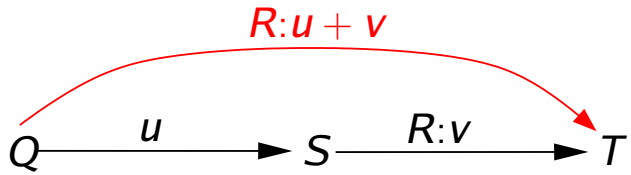
- Possible Outcome: C executes at 7.
- Final Schedule: $\{(Z = 0), (B = 4), (C = 7)\}$.

Semantics is nice, but . . .

- How can we check whether an STNU is DC?
- How can we construct a dynamic and valid strategy?

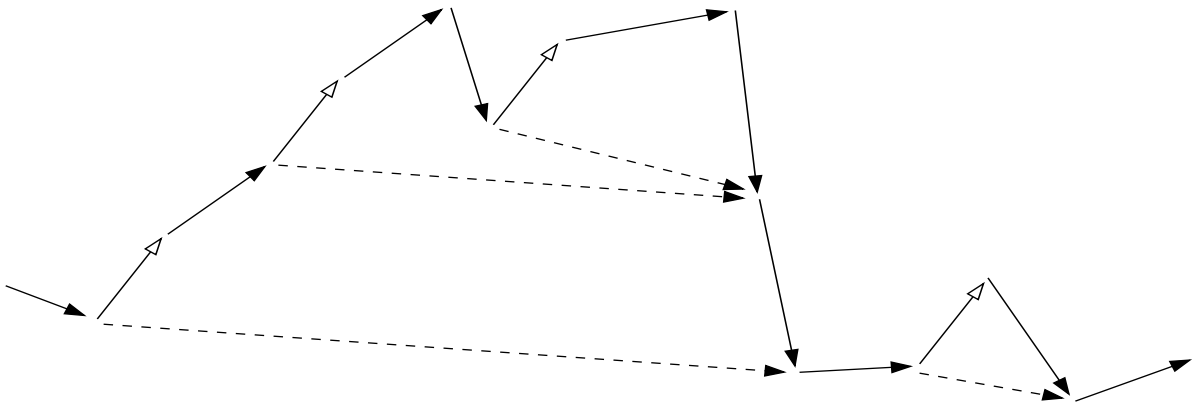
DC Checking for STNUs

- Propagate labeled edges to generate new edges



- Not all paths correspond to constraints that must be satisfied:

- Semi–reducible paths: *reduce away* lower–case edges



- DC if and only if no semi–reducible negative (SRN) loops

STN with Uncertainty
Semi–Reducible Paths

- In an STN: *shortest paths* represent the strongest constraints that every solution must satisfy.
- In an STNU: *shortest semi–reducible paths* represent the strongest constraints that a valid execution strategy.
- The *All–Pairs, Shortest Semi–Reducible Paths* (APSSRP) matrix $\mathcal{D}^*$ for an STNU is analogous to the APSP matrix for an STN.

STN with Uncertainty
Fundamental Theorem of STNUs

For an STNU $\mathcal{S}$, with graph $\mathcal{G}$, and APSSRP matrix $\mathcal{D}^*$, the following are equivalent:

- $\mathcal{S}$ is *dynamically controllable*
- $\mathcal{G}$ has no *semi–reducible* negative loops
- $\mathcal{D}^*$ has non–negative values on its main diagonal

[Hunsberger, 2010, 2013; Morris, 2006; Morris and Muscettola, 2005]

STN with Uncertainty
Edge–Generation Rules

The All–Pairs, Shortest Semi–Reducible Paths matrix for an STNU can be determined considering the following edge–generation rules:

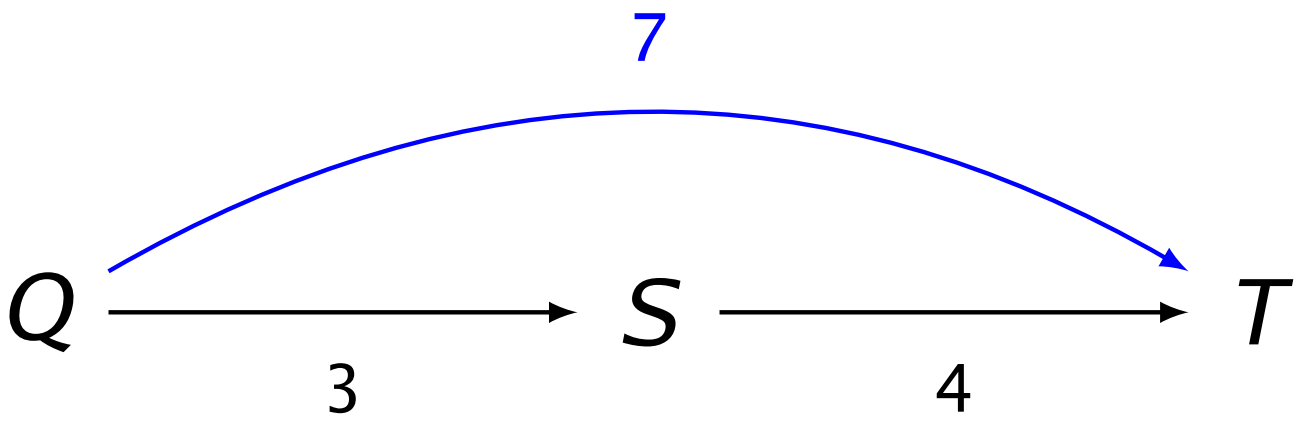
- *No Case* Rule
- *Upper–Case* Rule
- *Lower–Case* Rule
- *Cross–Case* Rule
- *Label–Removal* Rule

Key feature of the rules: they are *length–preserving*!

[Morris and Muscettola, 2005]

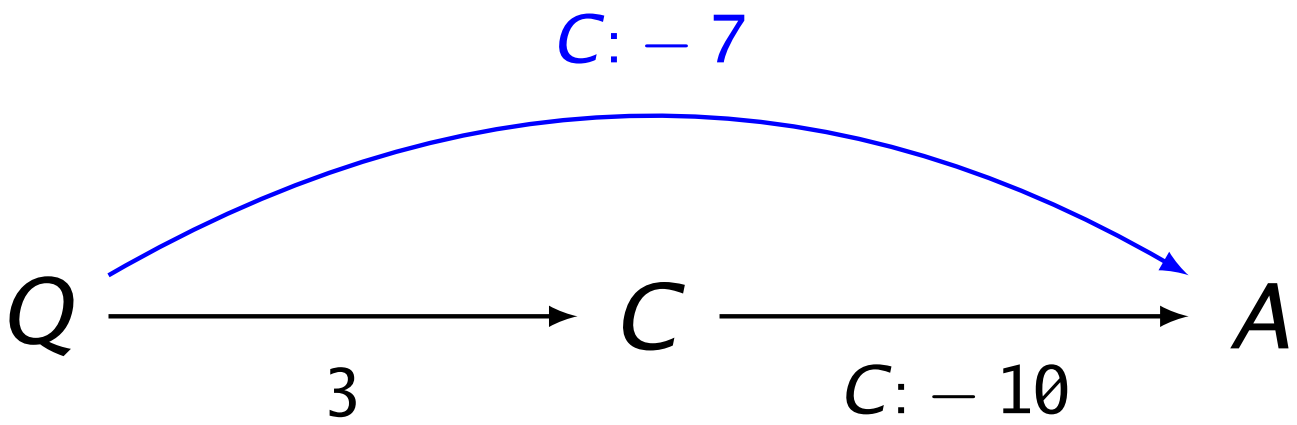
STN with Uncertainty
The No-Case Rule

STN with Uncertainty
The No-Case Rule



STN with Uncertainty
The Upper–Case Rule

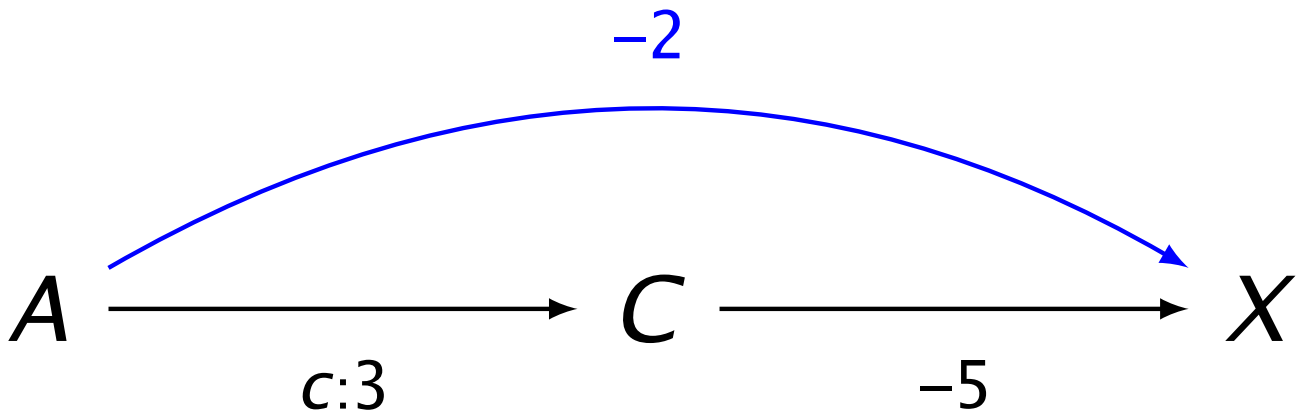
STN with Uncertainty
The Upper–Case Rule



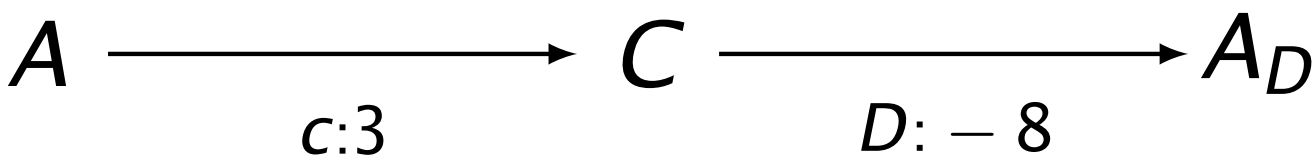
STN with Uncertainty
The Lower–Case Rule

(Applies since $-5 \leq 0$)

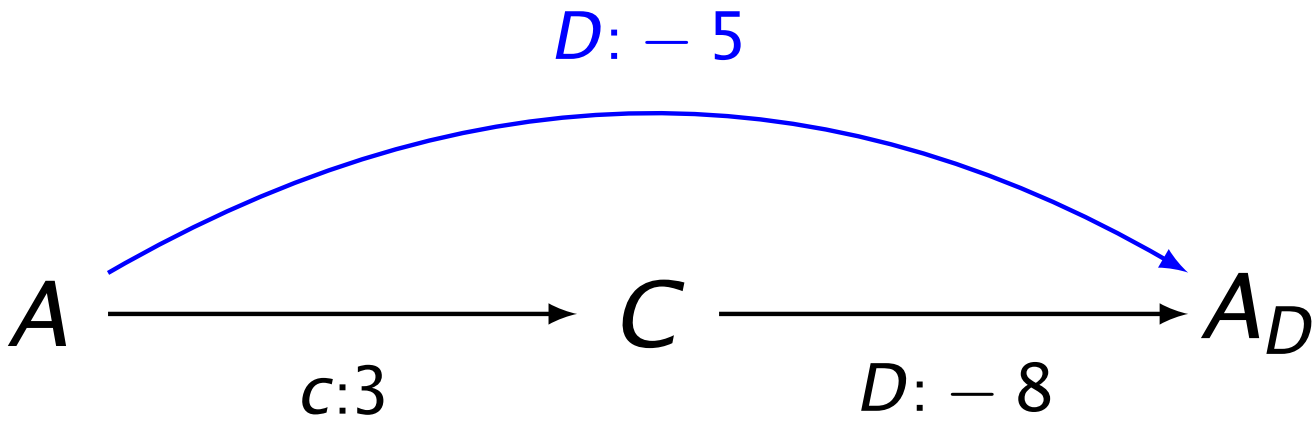
STN with Uncertainty
The Lower–Case Rule



(Applies since $-5 \leq 0$)



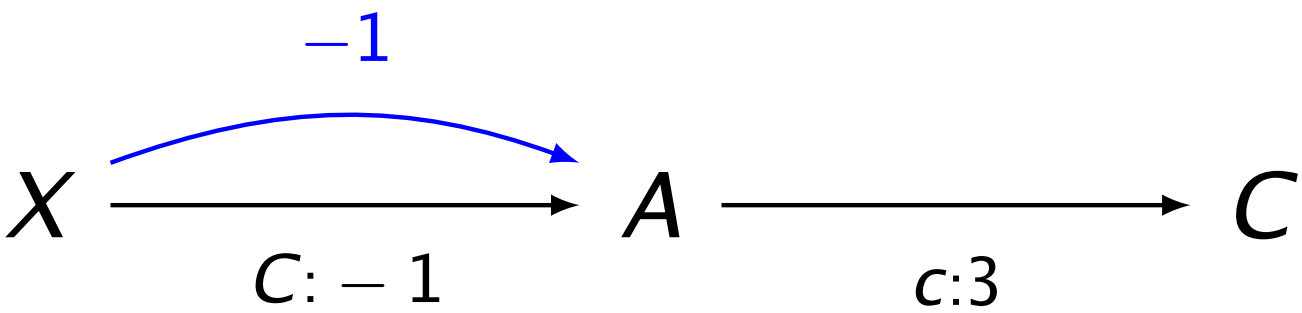
(Applies since $-8 \leq 0$ and $C \not\equiv D$)



(Applies since $-8 \leq 0$ and $C \not\equiv D$)

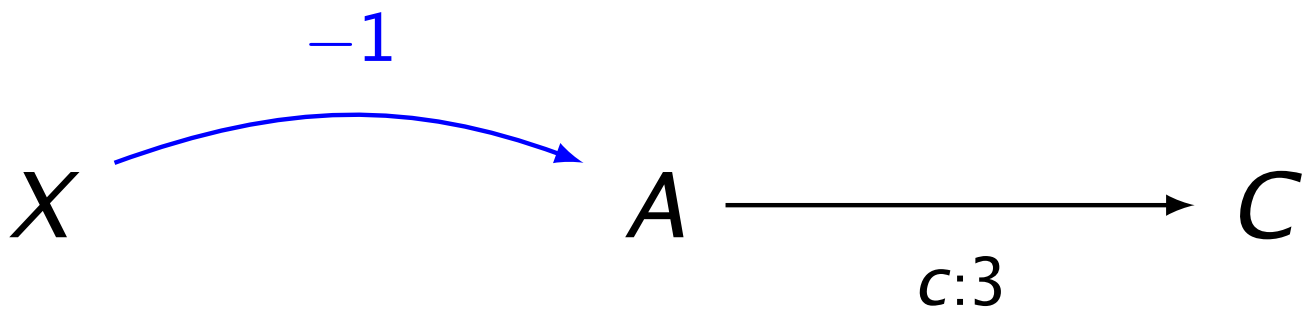
STN with Uncertainty
The Label–Removal Rule

STN with Uncertainty
The Label–Removal Rule



(Applies since $-1 \geq -3$)

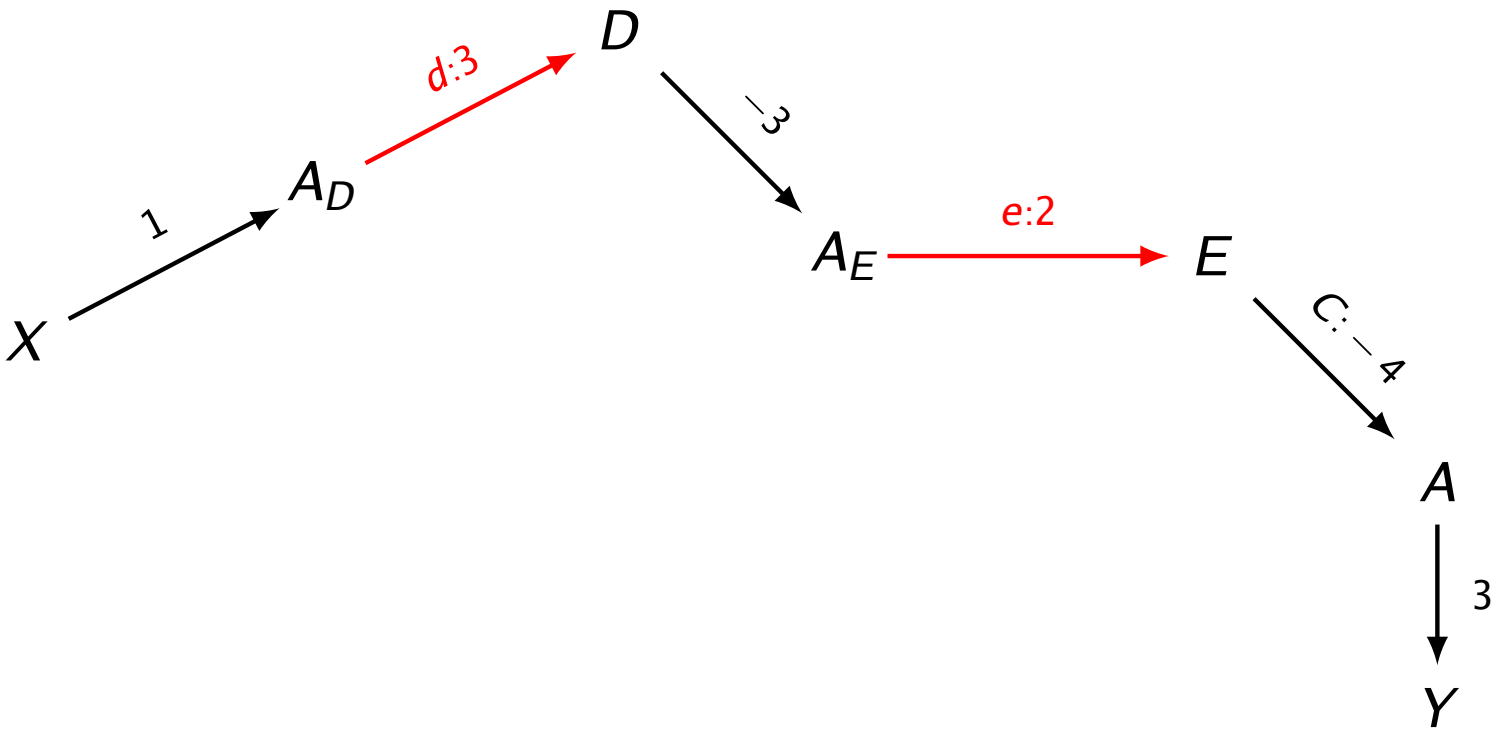
STN with Uncertainty
The Label–Removal Rule



STN with Uncertainty

Semi–Reducible Path [Morris and Muscettola, 2005]

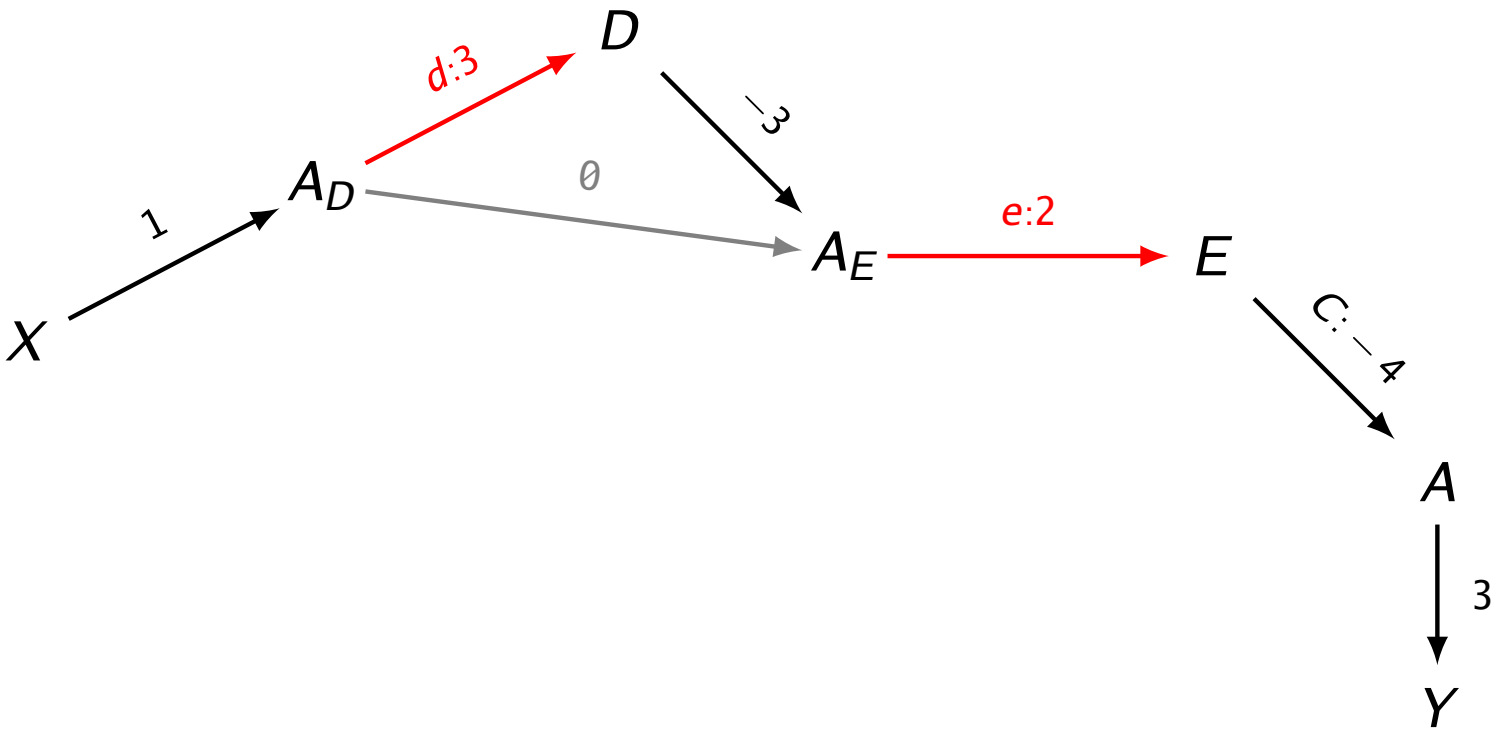
A path is *semi–reducible* if it can be transformed into a path with no *lower–case* edges.



STN with Uncertainty

Semi–Reducible Path [Morris and Muscettola, 2005]

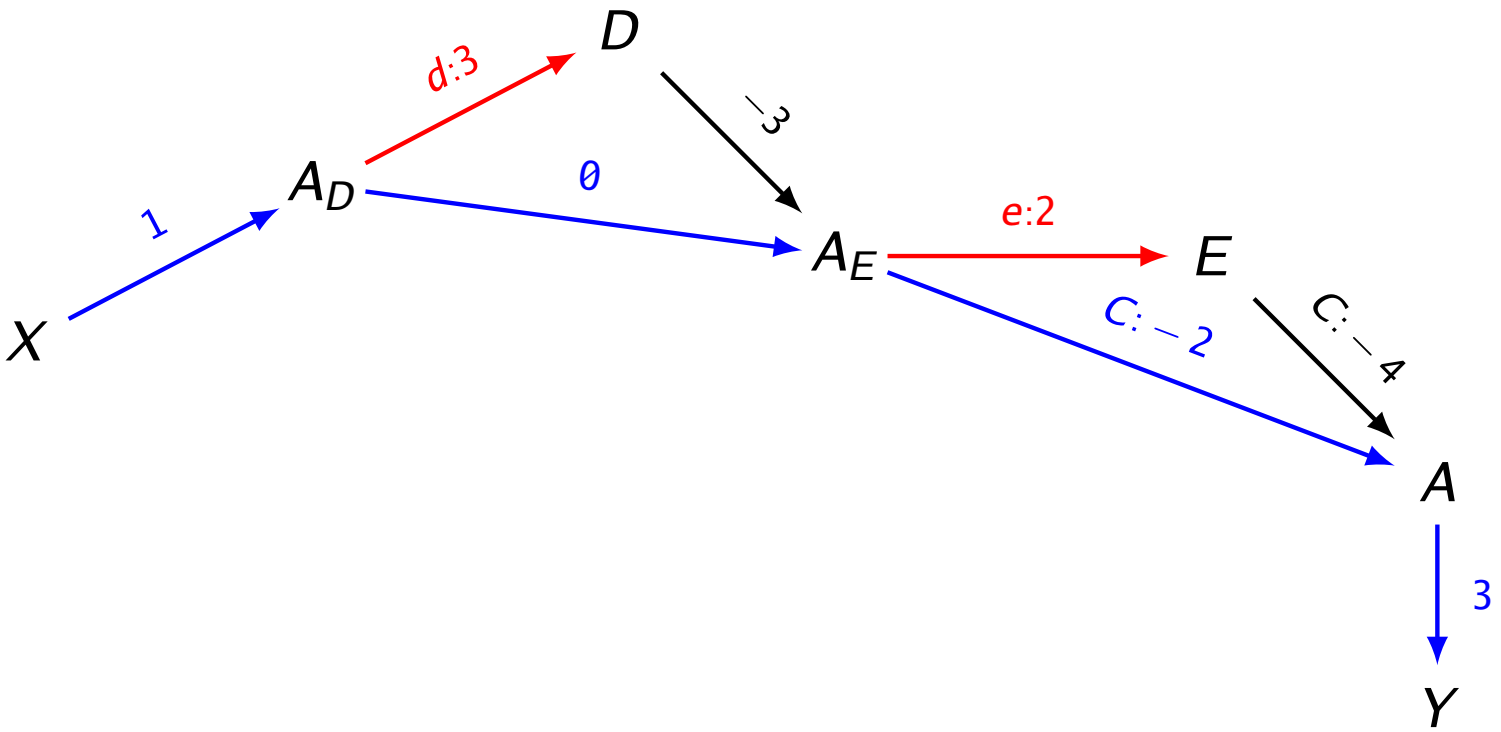
A path is *semi–reducible* if it can be transformed into a path with no *lower–case* edges.



STN with Uncertainty

Semi–Reducible Path [Morris and Muscettola, 2005]

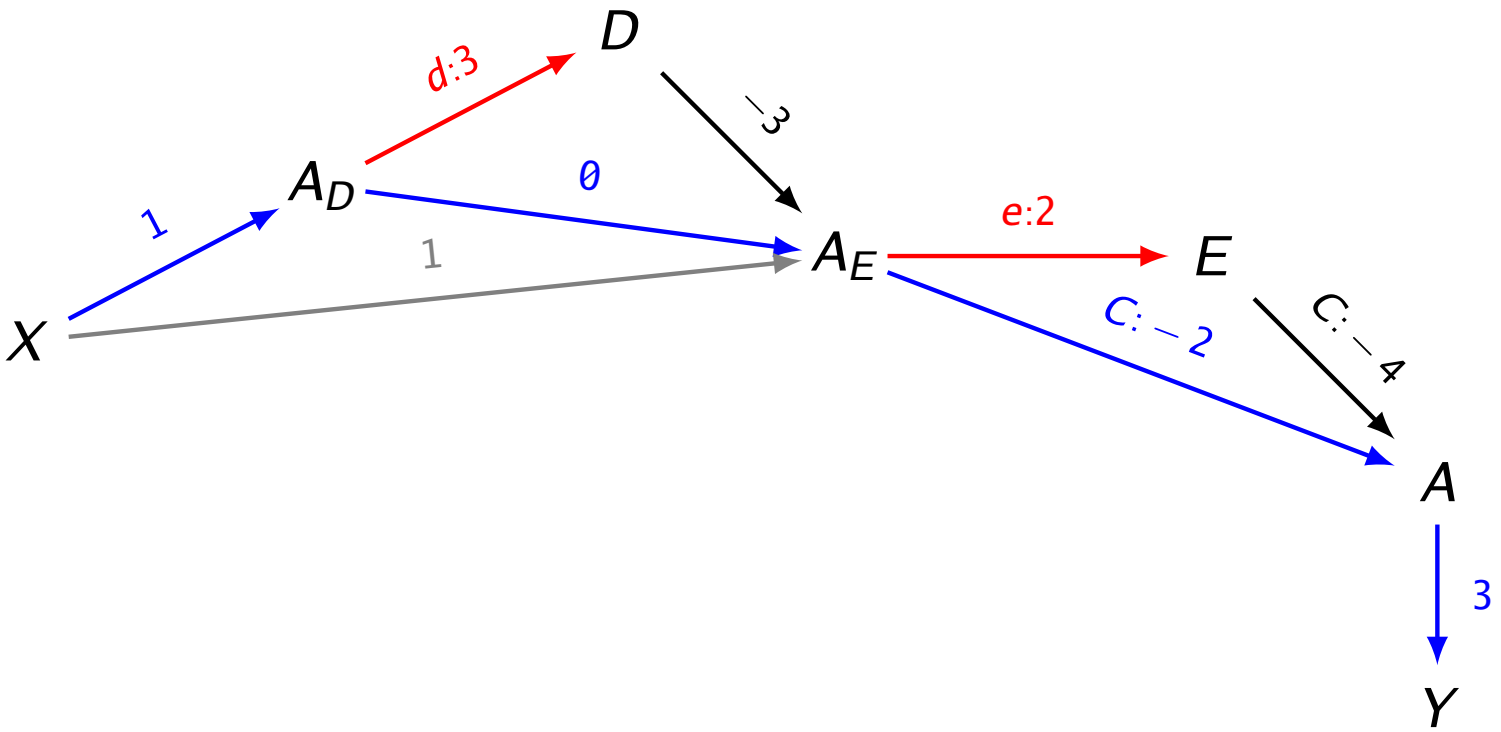
A path is *semi–reducible* if it can be transformed into a path with no *lower–case* edges.



STN with Uncertainty

Semi–Reducible Path [Morris and Muscettola, 2005]

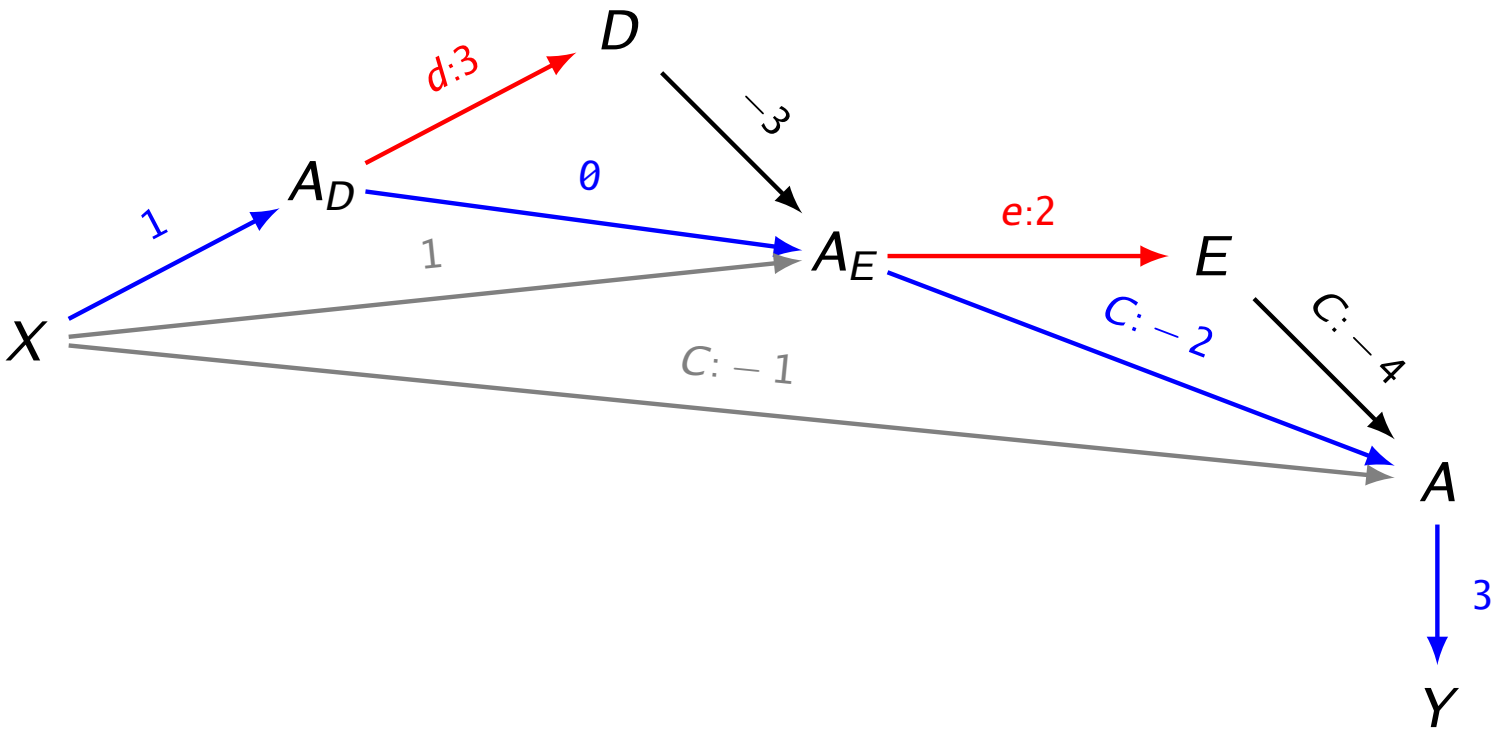
A path is *semi–reducible* if it can be transformed into a path with no *lower–case* edges.



STN with Uncertainty

Semi–Reducible Path [Morris and Muscettola, 2005]

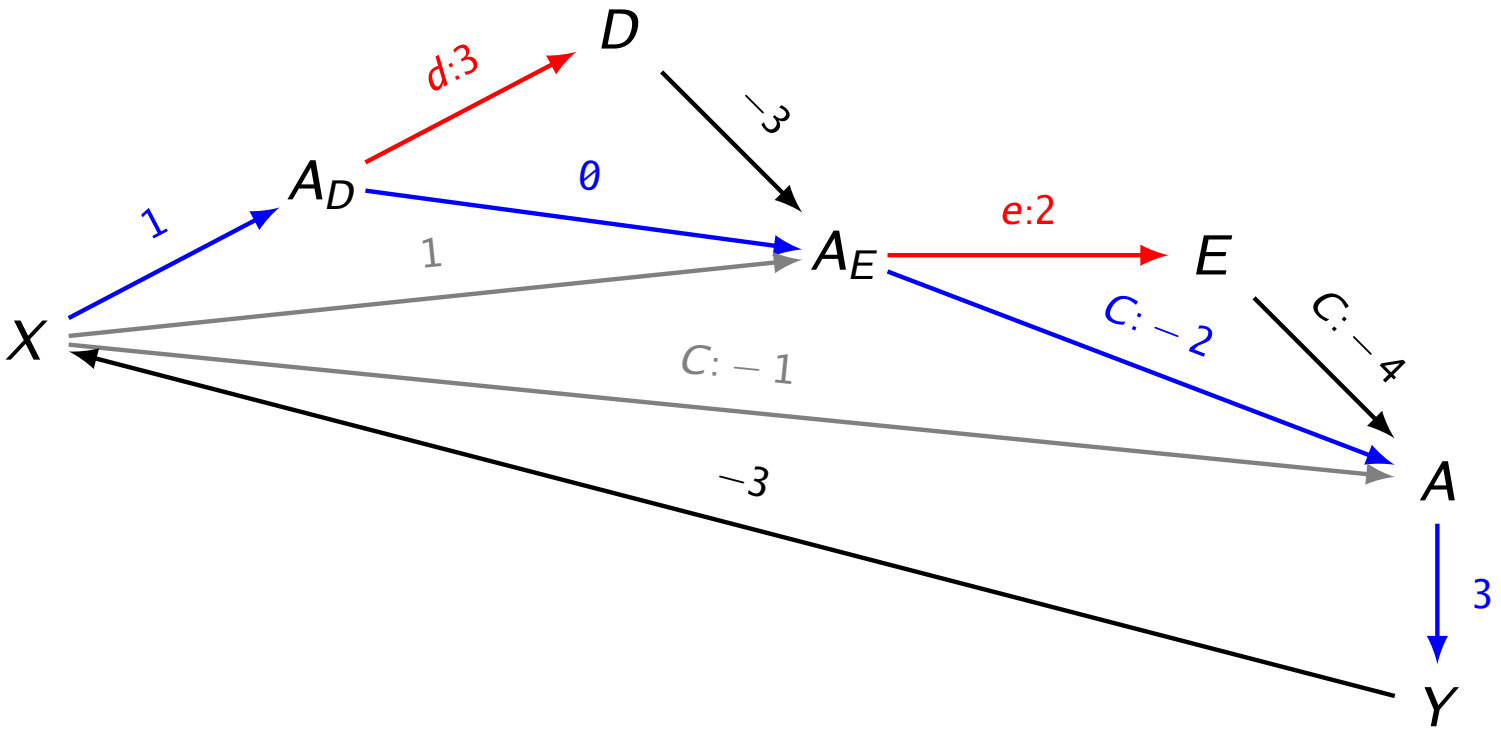
A path is *semi–reducible* if it can be transformed into a path with no *lower–case* edges.



STN with Uncertainty

Semi–Reducible Path [Morris and Muscettola, 2005]

A path is *semi–reducible* if it can be transformed into a path with no *lower–case* edges.

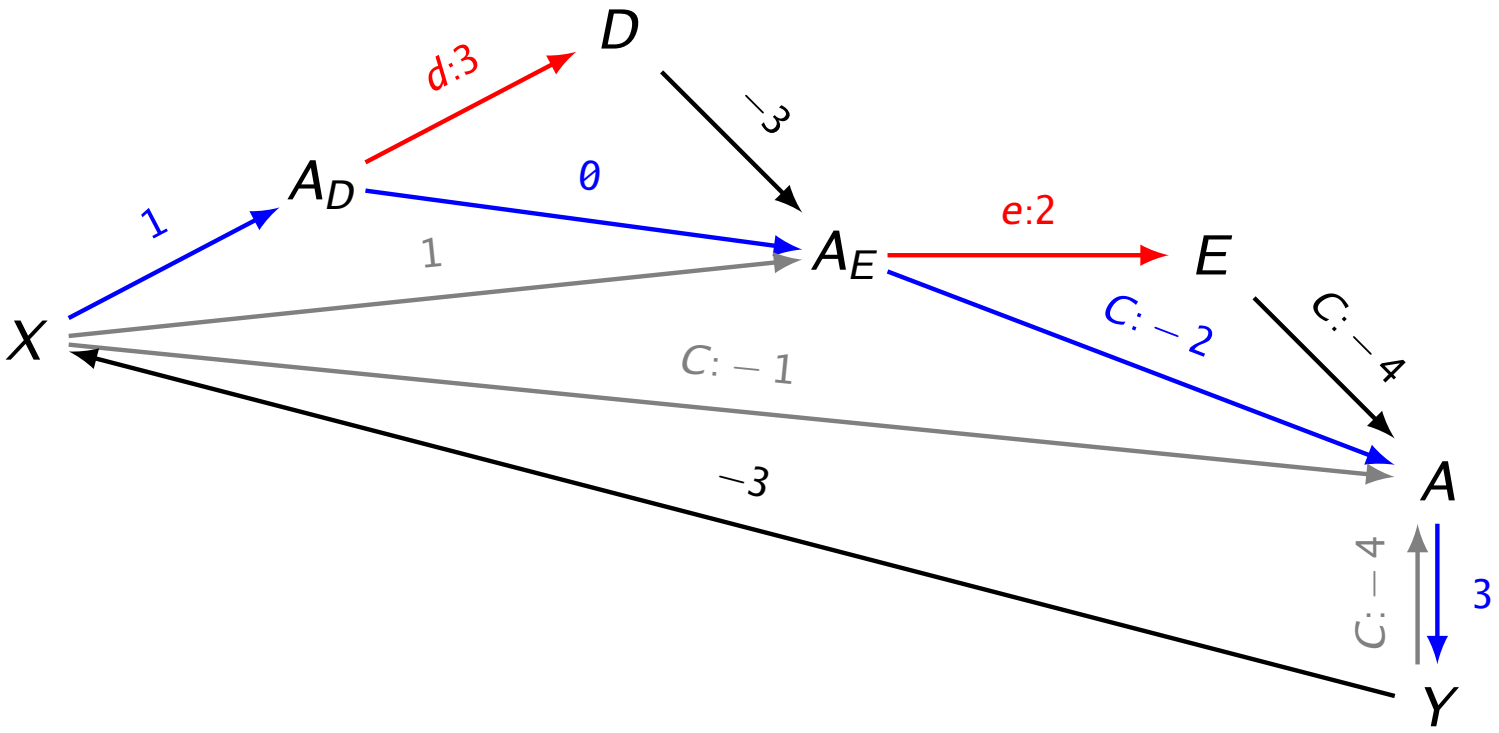


Example 2: Same as before, but with edge from Y and X

STN with Uncertainty

Semi–Reducible Path [Morris and Muscettola, 2005]

A path is *semi–reducible* if it can be transformed into a path with no *lower–case* edges.

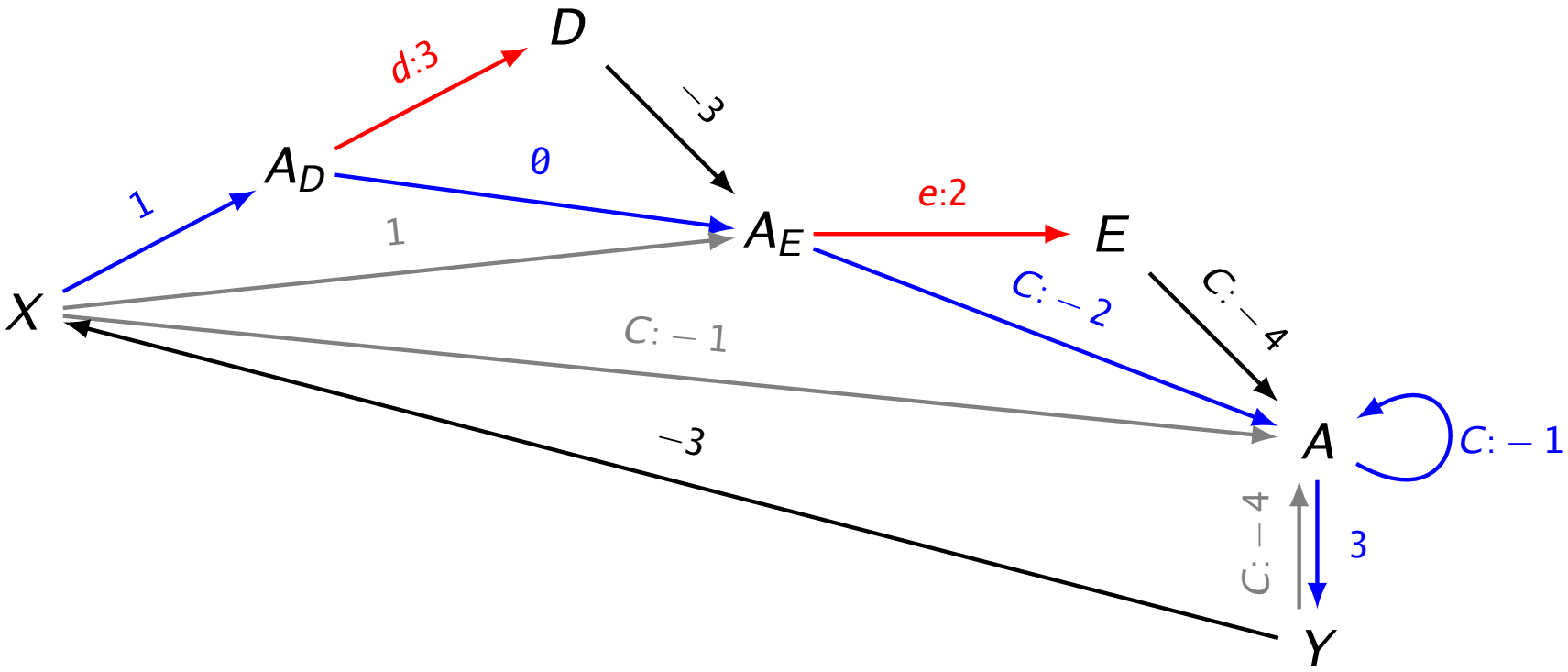


Example 2: Same as before, but with edge from Y and X

STN with Uncertainty

Semi–Reducible Path [Morris and Muscettola, 2005]

A path is *semi–reducible* if it can be transformed into a path with no *lower–case* edges.



A *semi–reducible* negative (SRN) loop at *A*!

STN with Uncertainty
Fundamental Theorem of STNUs

For an STNU $\mathcal{S}$, with graph $\mathcal{G}$, and APSSRP matrix $\mathcal{D}^*$, the following are equivalent:

- $\mathcal{S}$ is dynamically controllable
- $\mathcal{G}$ has no *semi–reducible* negative loops
- $\mathcal{D}^*$ has non–negative values on its main diagonal

[Hunsberger, 2010, 2013; Morris, 2006; Morris and Muscettola, 2005]

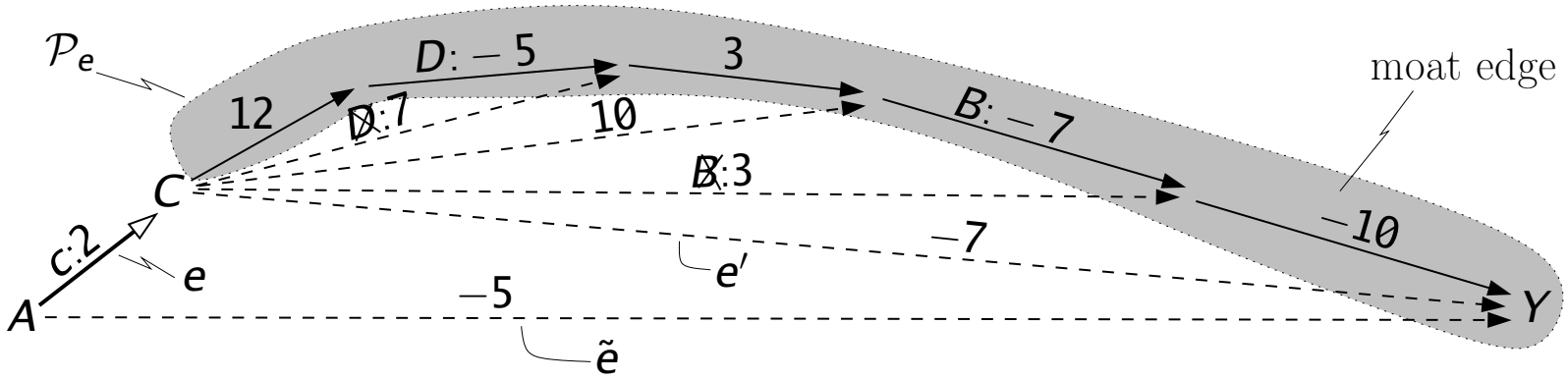
The worst–case time for DC–checking algorithms for STNUs has improved dramatically in recent years:

- Pseudo–polynomial: [Morris et al., 2001]
- $O(n^5)$: [Morris and Muscettola, 2005]
- $O(n^4)$: [Morris, 2006]
- $O(n^3)$: [Morris, 2014]
- $O(mn + k^2 n + kn \log n)$: [Cairo et al., 2018]

(k = num. cont. links; n = num. time–points; m = num. edges)

Reducing Away a Lower–Case Edge

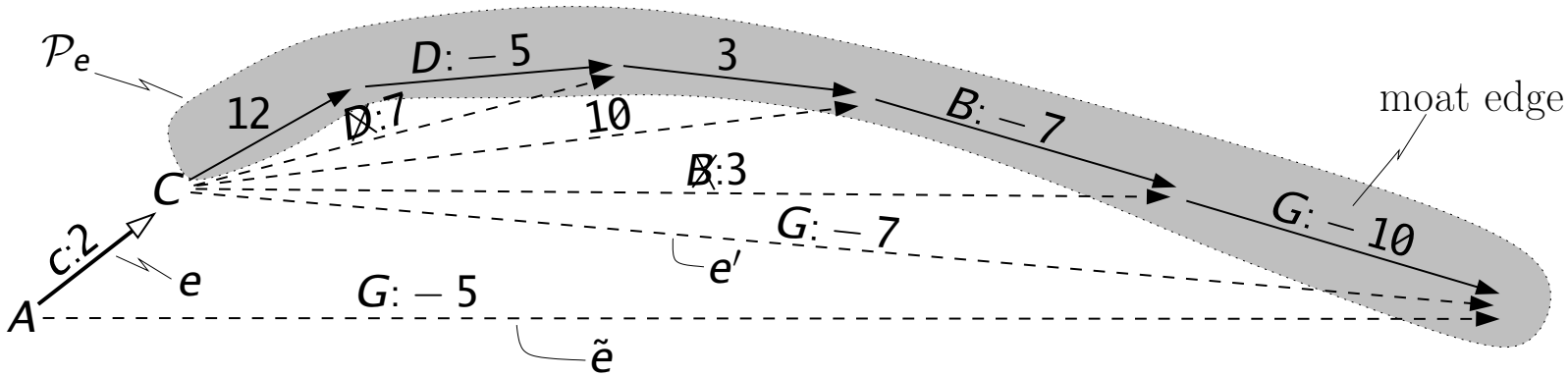
Example 1: *Moat* edge is an ordinary edge; so is generated edge.



$\mathcal{P}_e$: *extension sub–path* for the lower–case edge, e.

Reducing Away a Lower–Case Edge

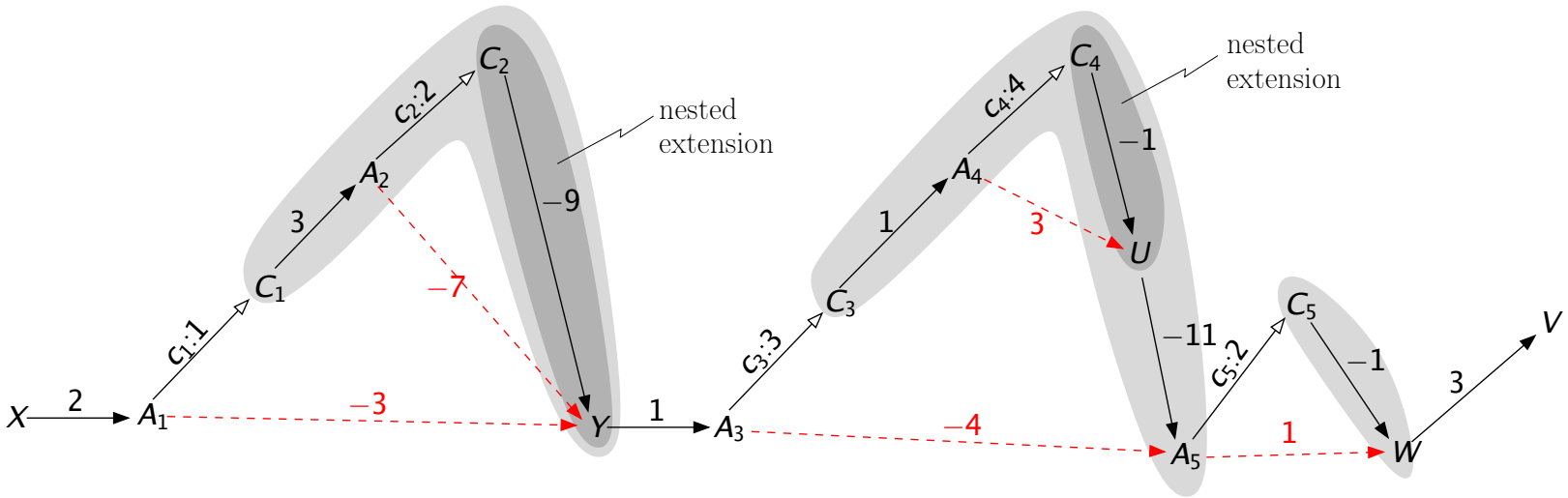
Example 2: *Moat* edge is an upper–case edge; so is generated edge.



$\mathcal{P}_e$: *extension sub–path* for the lower–case edge, e.

Reducing Away Lower–Case Edges

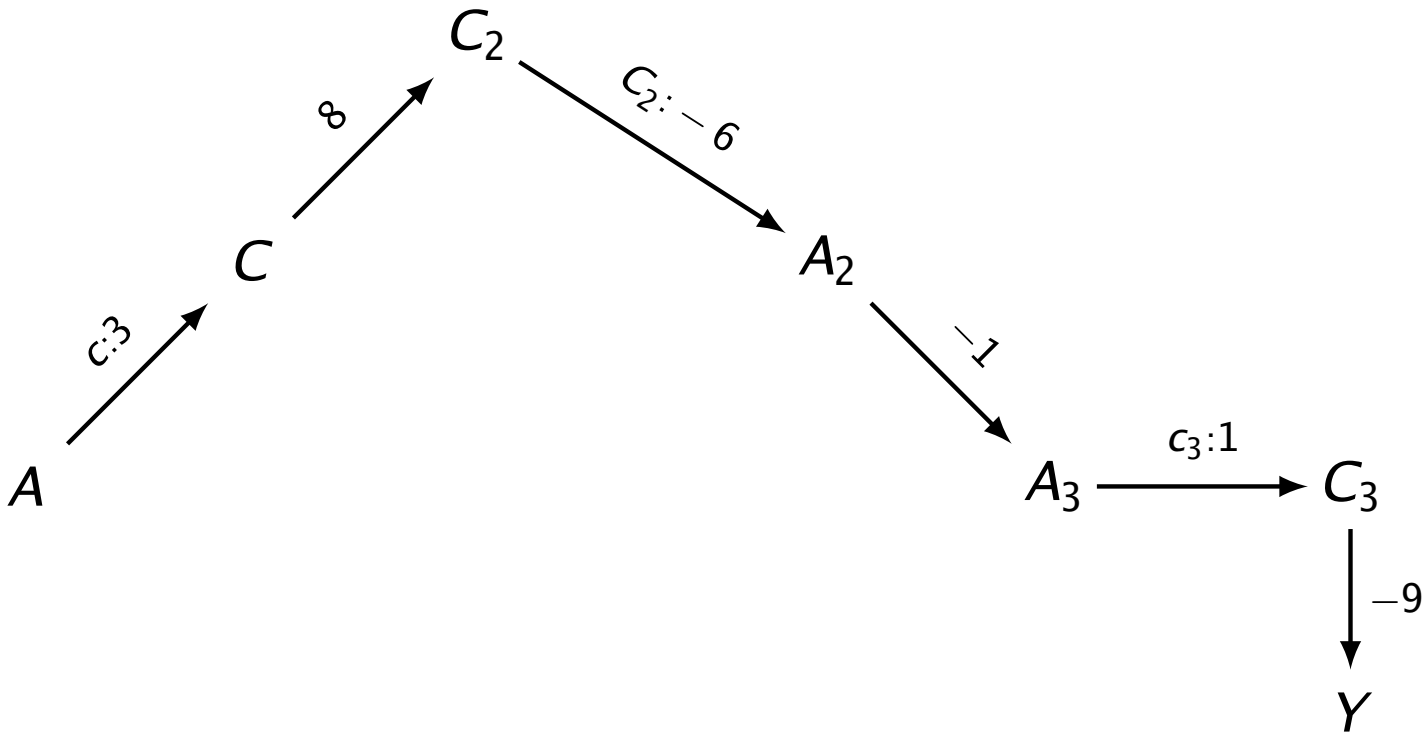
Extension sub–paths can be nested



If extension sub–paths overlap . . .
one must be nested inside the other [Morris, 2006].

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

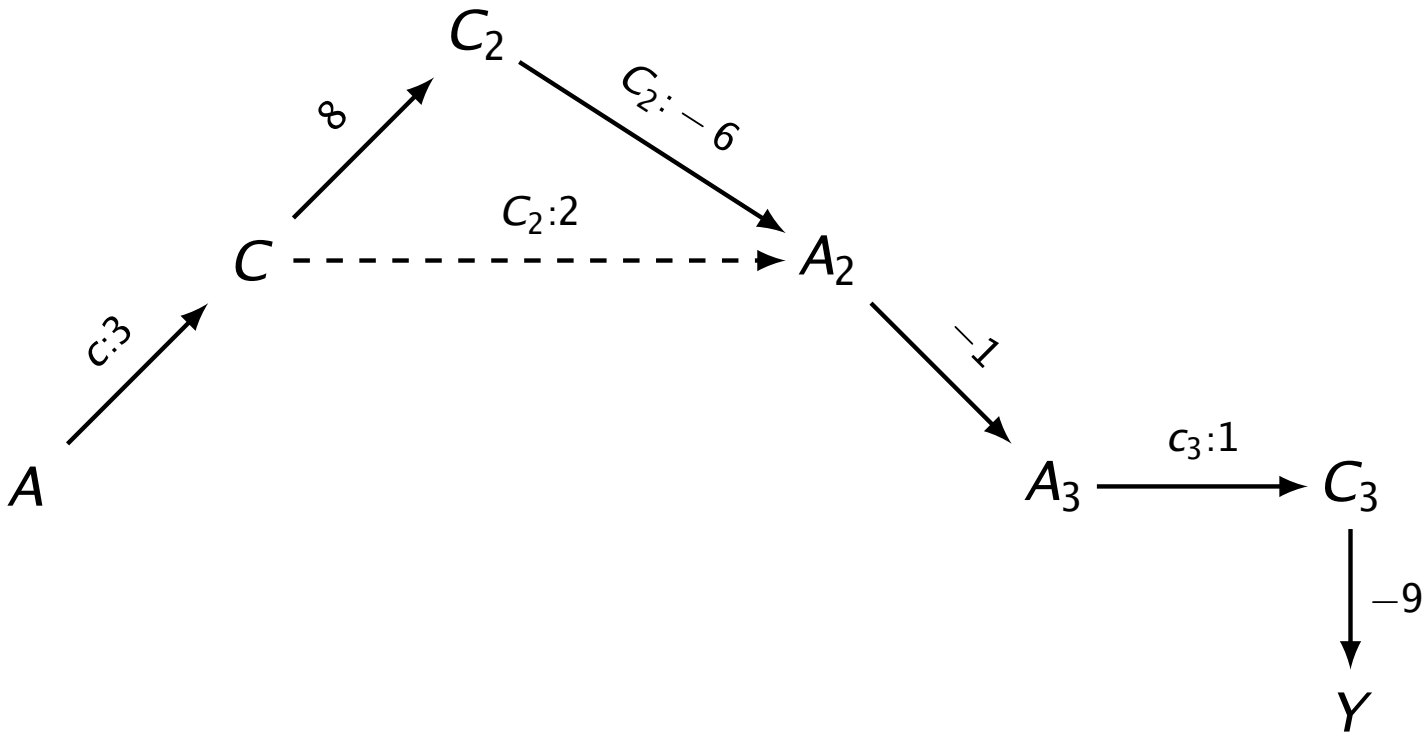
Based on "canonical reduction" of semi–reducible paths



Key idea: Propagate forward along "allowable" paths emanating from C attempting to "reduce away" the lower–case edge AC.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

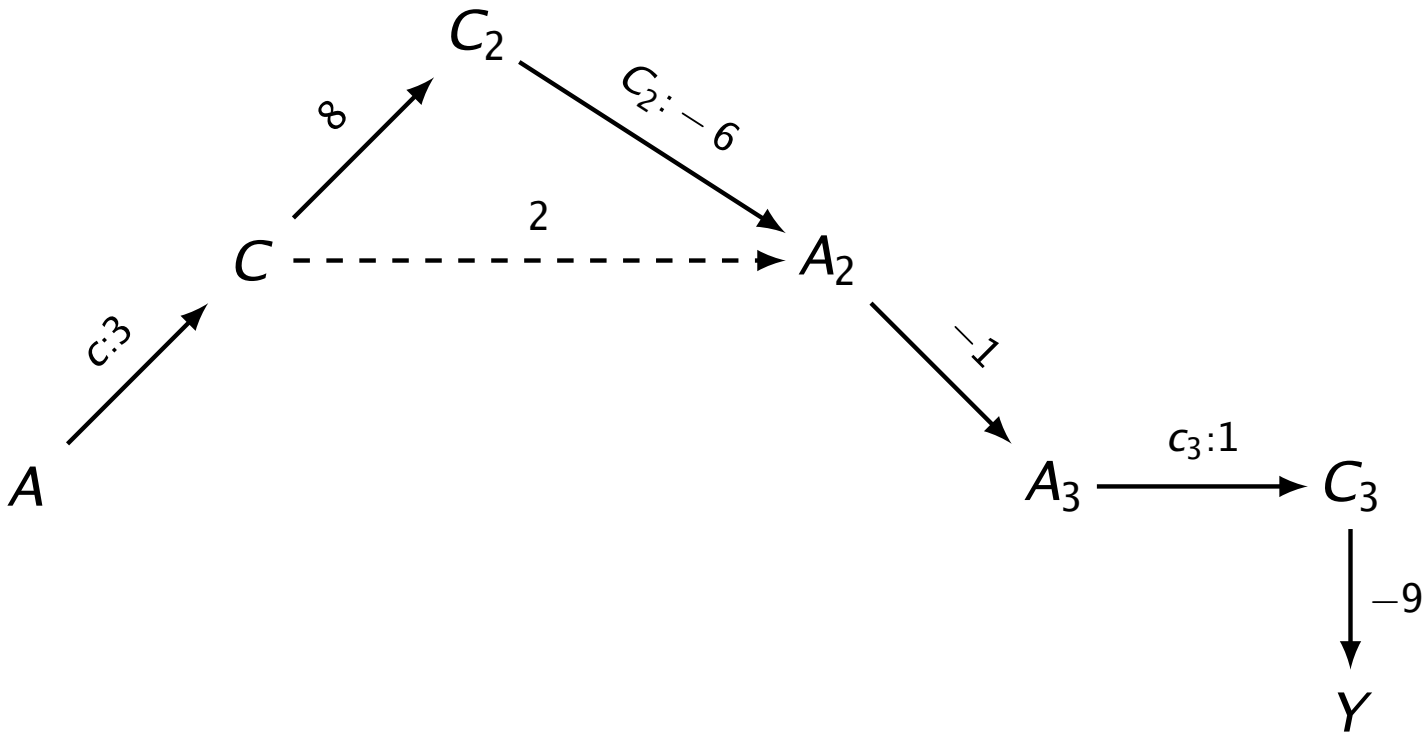
Based on "canonical reduction" of semi–reducible paths



Key idea: Propagate forward along "allowable" paths emanating from C attempting to "reduce away" the lower–case edge AC.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

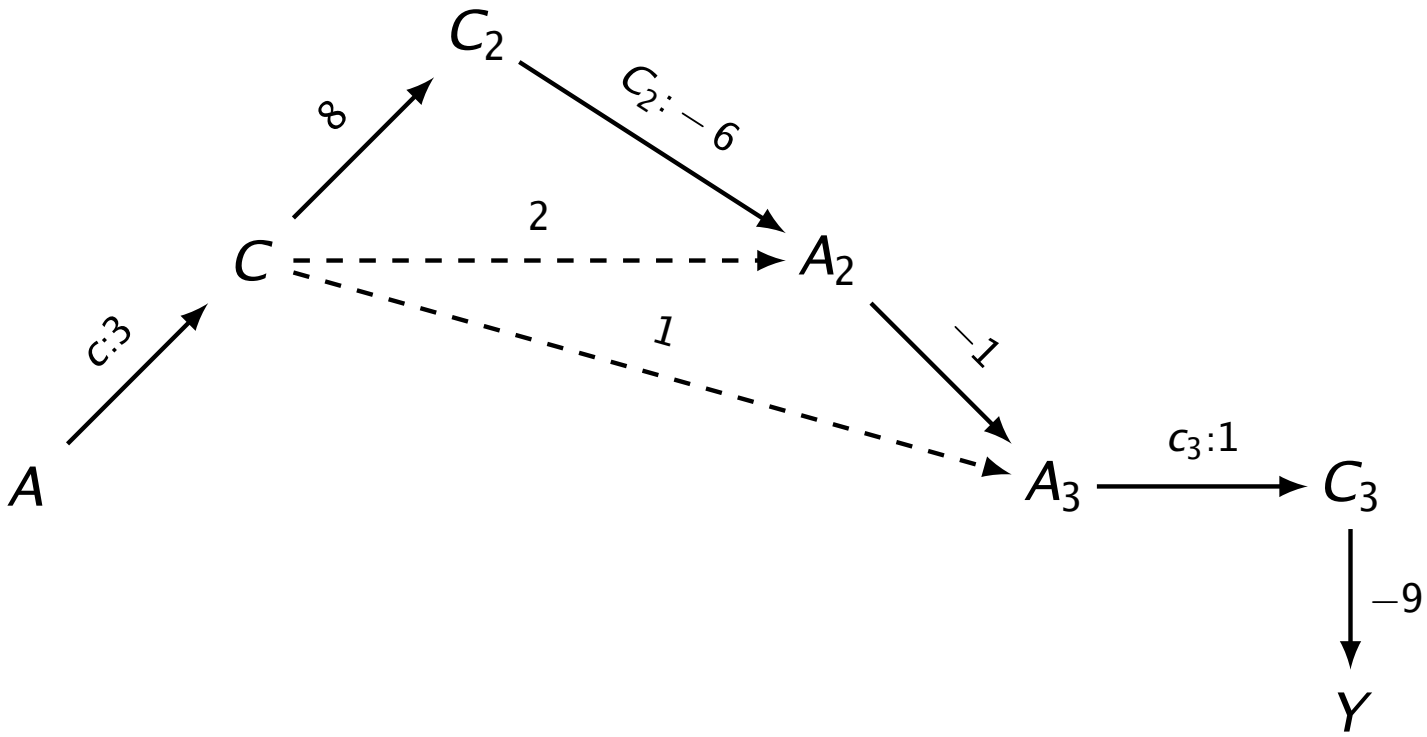
Based on "canonical reduction" of semi–reducible paths



Key idea: Propagate forward along "allowable" paths emanating from C attempting to "reduce away" the lower–case edge AC.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

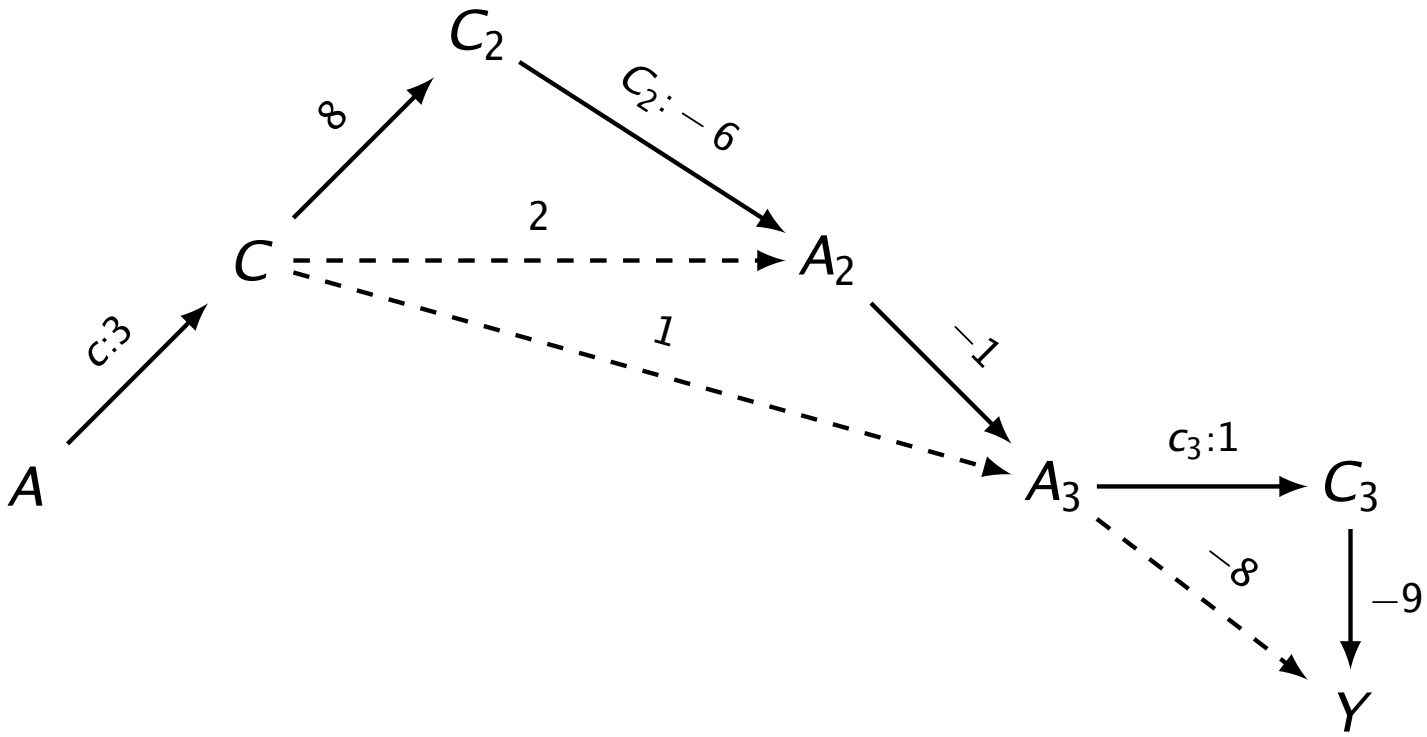
Based on "canonical reduction" of semi–reducible paths



Key idea: Propagate forward along "allowable" paths emanating from C attempting to "reduce away" the lower–case edge AC.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

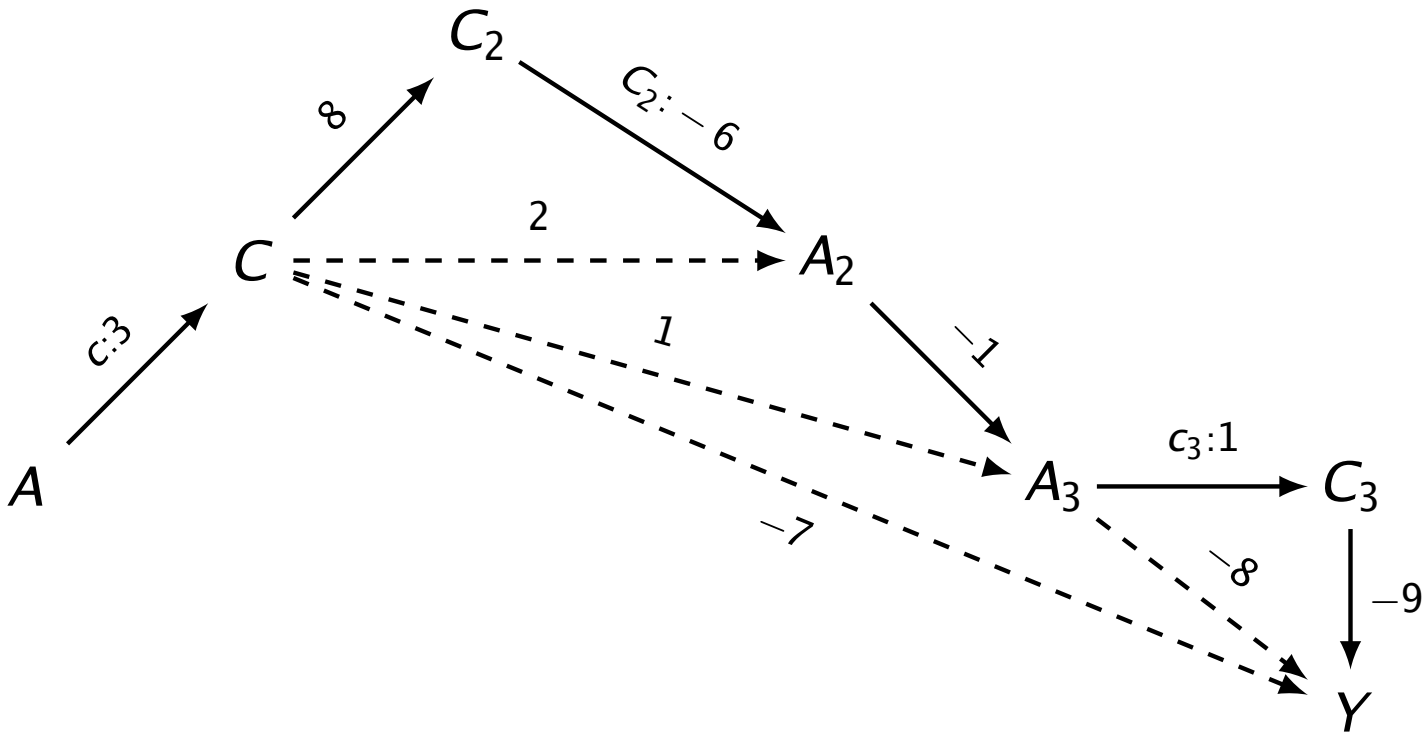
Based on "canonical reduction" of semi–reducible paths



Key idea: Propagate forward along "allowable" paths emanating from C attempting to "reduce away" the lower–case edge AC.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

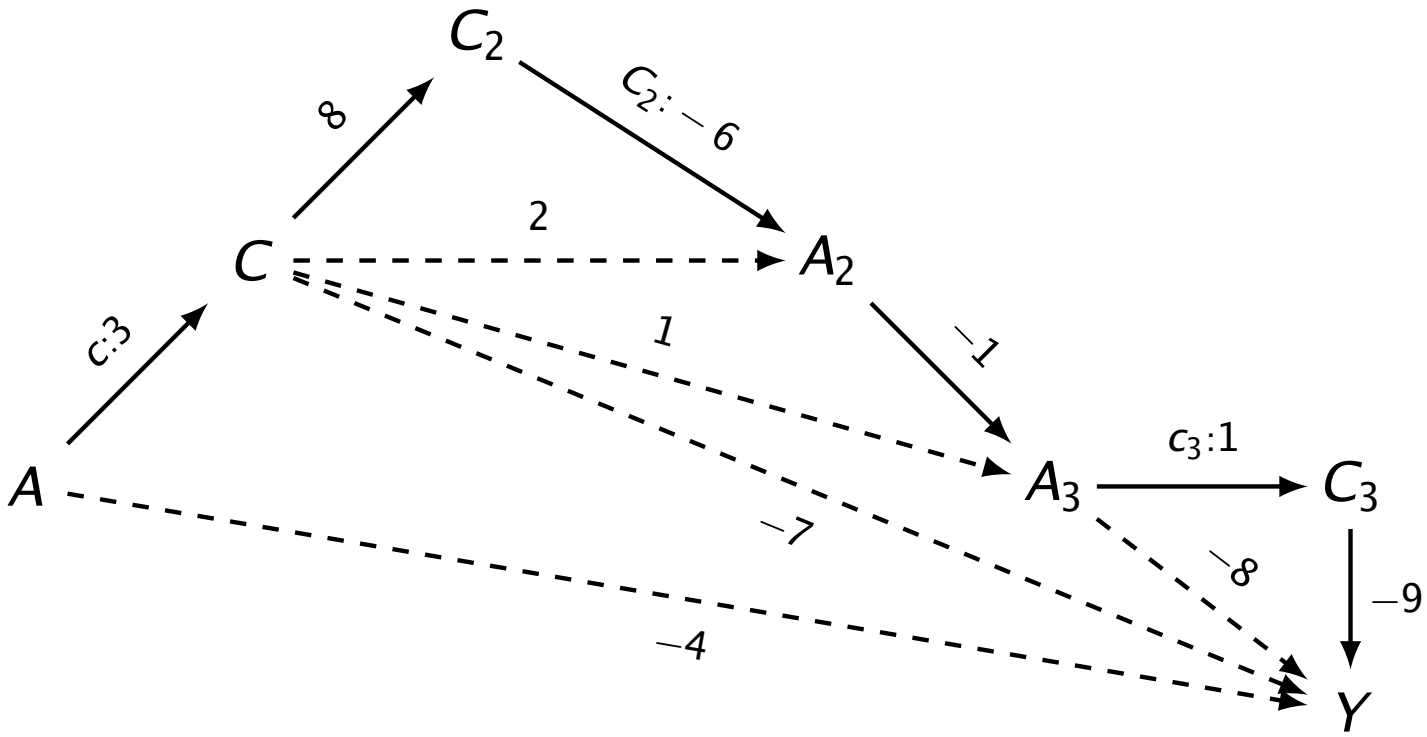
Based on "canonical reduction" of semi–reducible paths



Key idea: Propagate forward along "allowable" paths emanating from C attempting to "reduce away" the lower–case edge AC.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

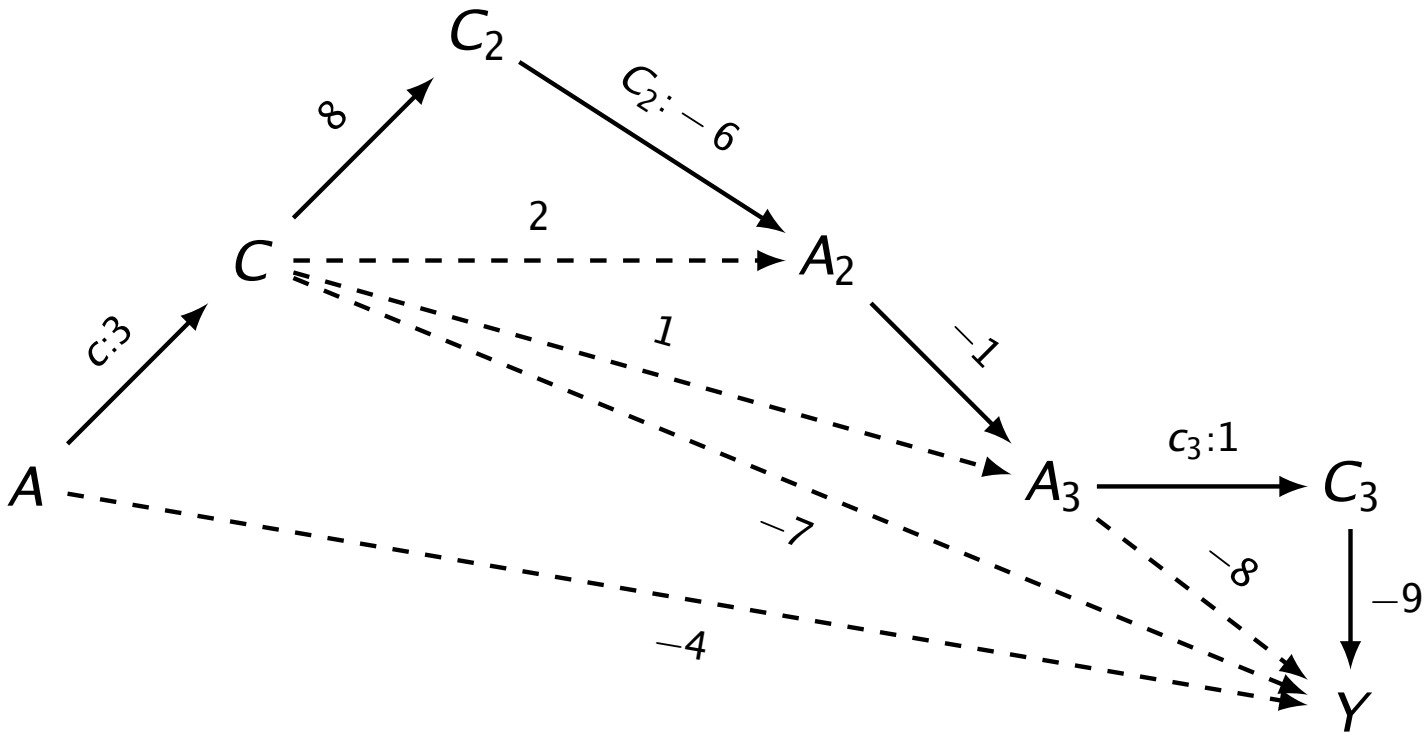
Based on "canonical reduction" of semi–reducible paths



Key idea: Propagate forward along "allowable" paths emanating from C attempting to "reduce away" the lower–case edge AC.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

Based on "canonical reduction" of semi–reducible paths



Key idea: Propagate forward along "allowable" paths emanating from C attempting to "reduce away" the lower–case edge AC.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

- Assumes strategy can react *instantaneously*!
  - —Simplifies analysis; changes applicability conditions slightly.
  - —See Hunsberger [2015] for non–instantaneous case.
- The extension sub–path used in the canonical reduction of a lower–case edge necessarily has the PP property!
  - Every proper prefix has non–negative length
  - Every proper postfix has negative length
- Restrict attention to "allowable" paths emanating from C:
  - No upper–case edges labeled by same C.
  - No lower–case edges.

The *AllMax* STN*

- The *AllMax* STN is the projection in which all contingent links take on their maximum durations.
- The *AllMax* graph can be obtained by:
  - removing all lower–case edges, and
  - deleting the upper–case labels from upper–case edges.
- The distance matrix for the *AllMax* STN is called $\mathcal{D}_{\max}$.
- If an STNU is fully propagated, then $\mathcal{D}_{\max} = \mathcal{D}^*$.†
  ($\mathcal{D}^*$: the *All–Pairs, Shortest–Semi–Reducible–Paths* matrix)
- An STNU is DC iff $\mathcal{D}^*$ has zeores down its main diagonal.

* [Morris and Muscettola, 2005], † [Hunsberger, 2015]

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

- Canonical paths used to reduce away lower–case edges can be *nested* to a depth of at most k.
- Therefore, algorithm does k rounds of propagating forward from each lower–case edge.
- Allowable paths belong to the OU–graph.
- Before each round, runs Bellman–Ford on the *AllMax* graph to generate a potential function.
- Each forward propagation uses slightly modified Dijkstra's algorithm on re–weighted edges in OU–graph.

Morris' 2006 $O(n^4)$ DC–Checking Algorithm

`for i = 1 to k,`

Run Bellman–Ford on *AllMax* graph to get potential function

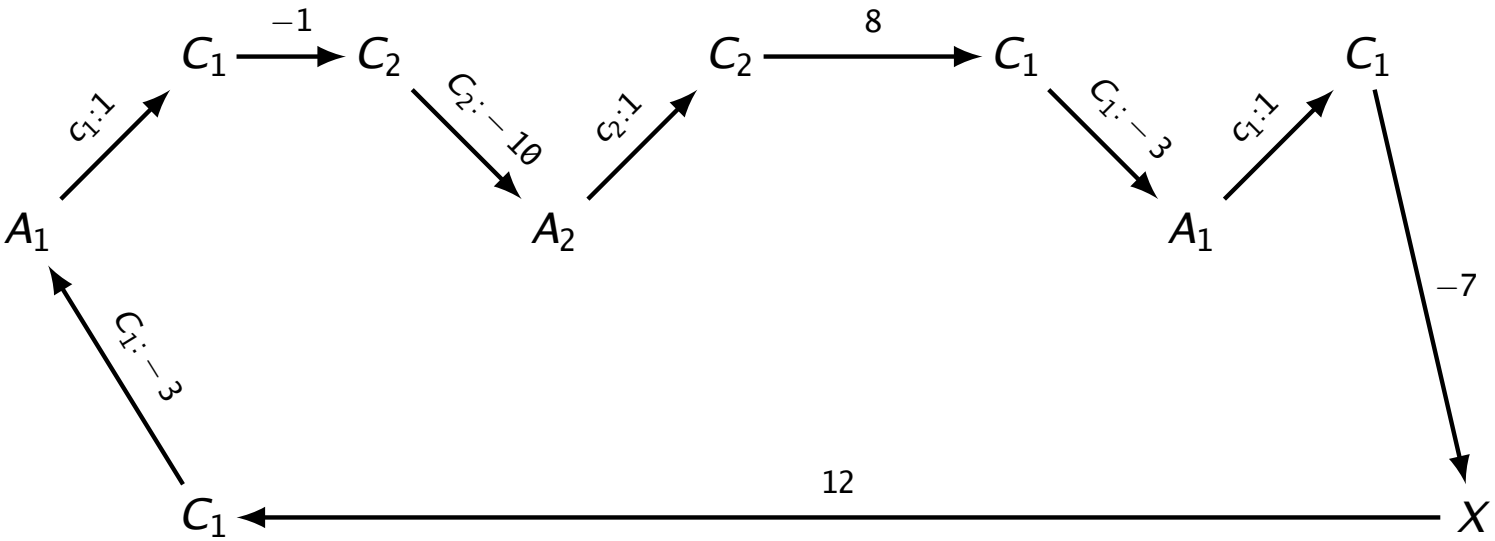
`for each contingent link` (A, x, y, C),

Do Dijkstra–like propagation using C as source,
following "allowable" paths in re–weighted OU–graph.

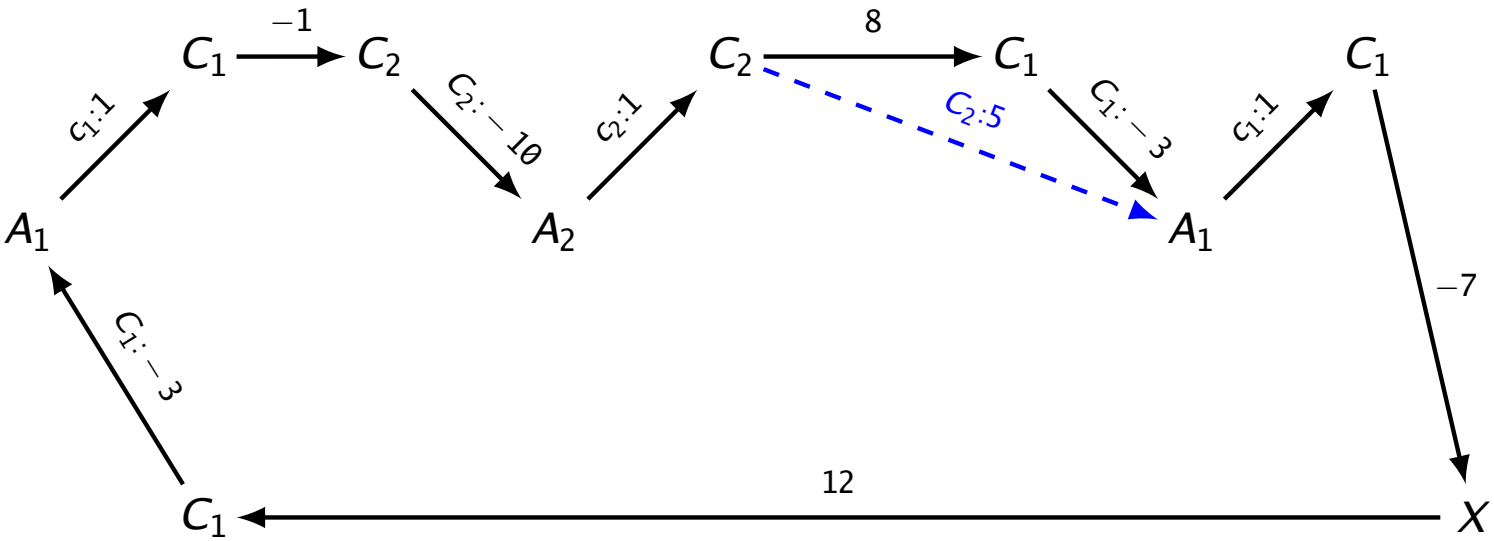
Insert all new edges from this round.

Run Bellman–Ford one last time on *AllMax* graph to verify consistency.

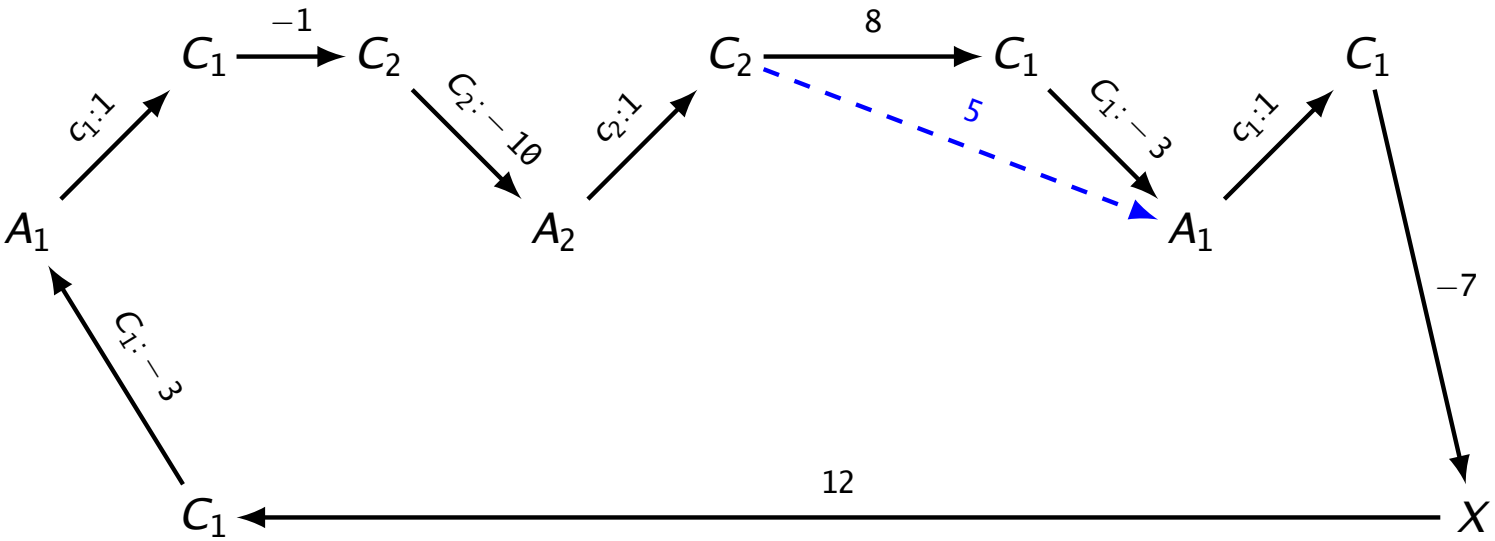
Finding a Semi–Reducible Negative Loop



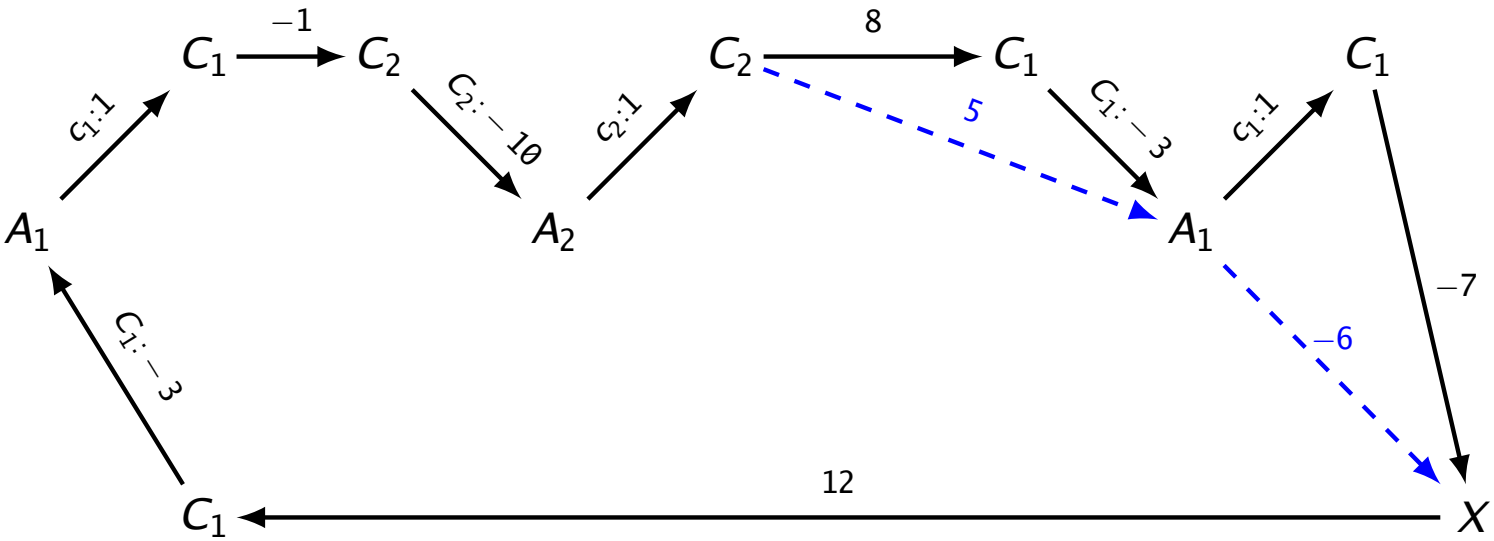
Finding a Semi–Reducible Negative Loop



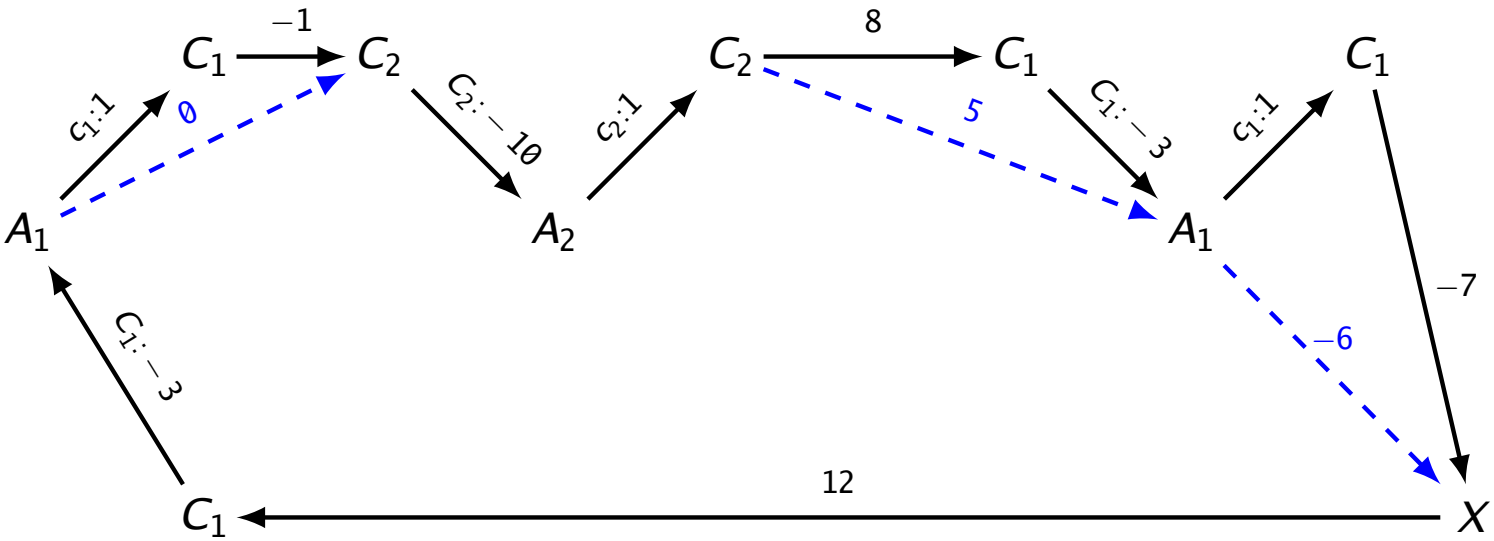
Finding a Semi–Reducible Negative Loop



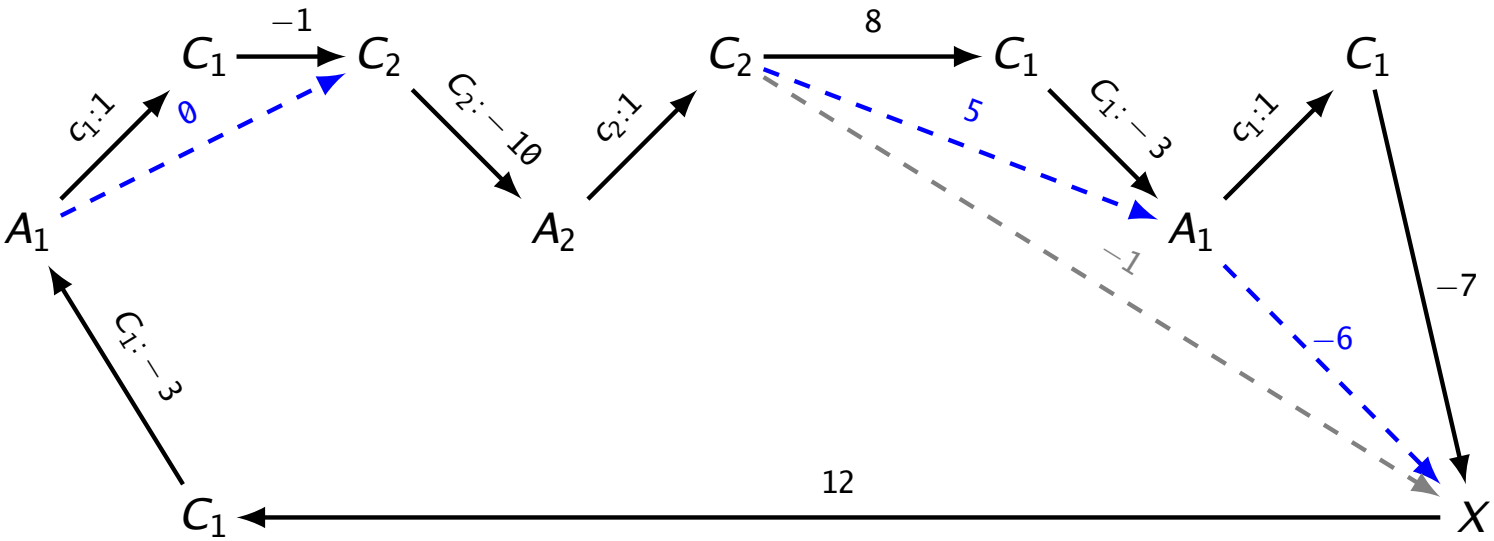
Finding a Semi–Reducible Negative Loop



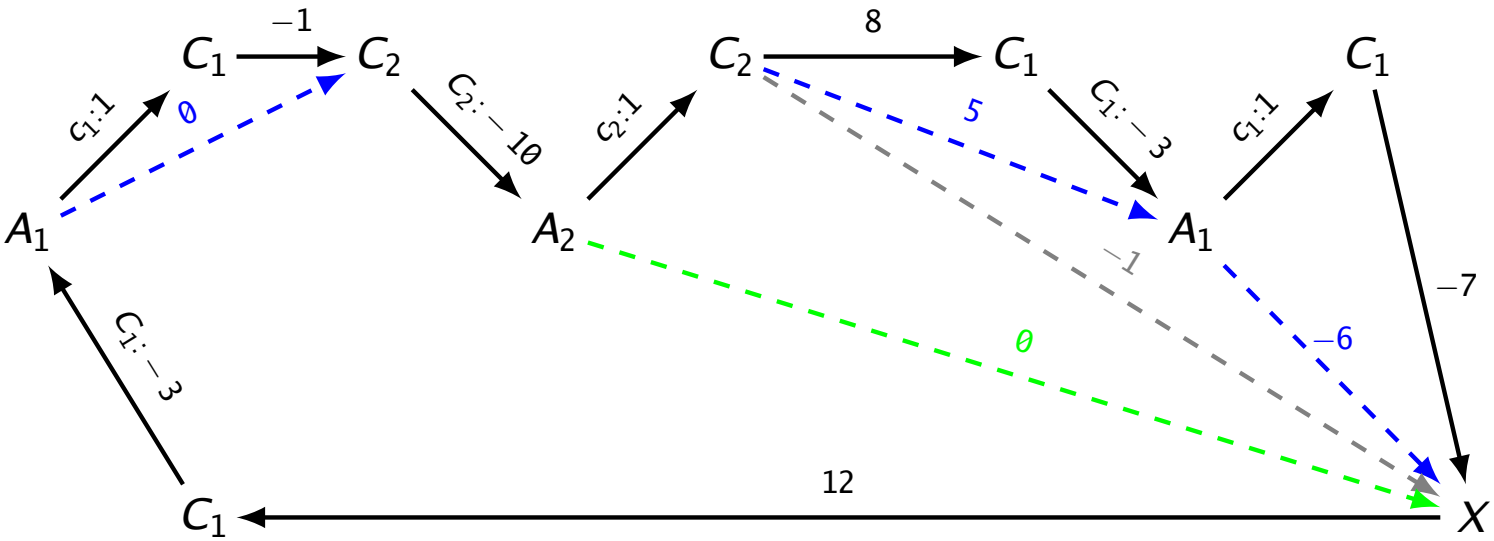
Finding a Semi–Reducible Negative Loop



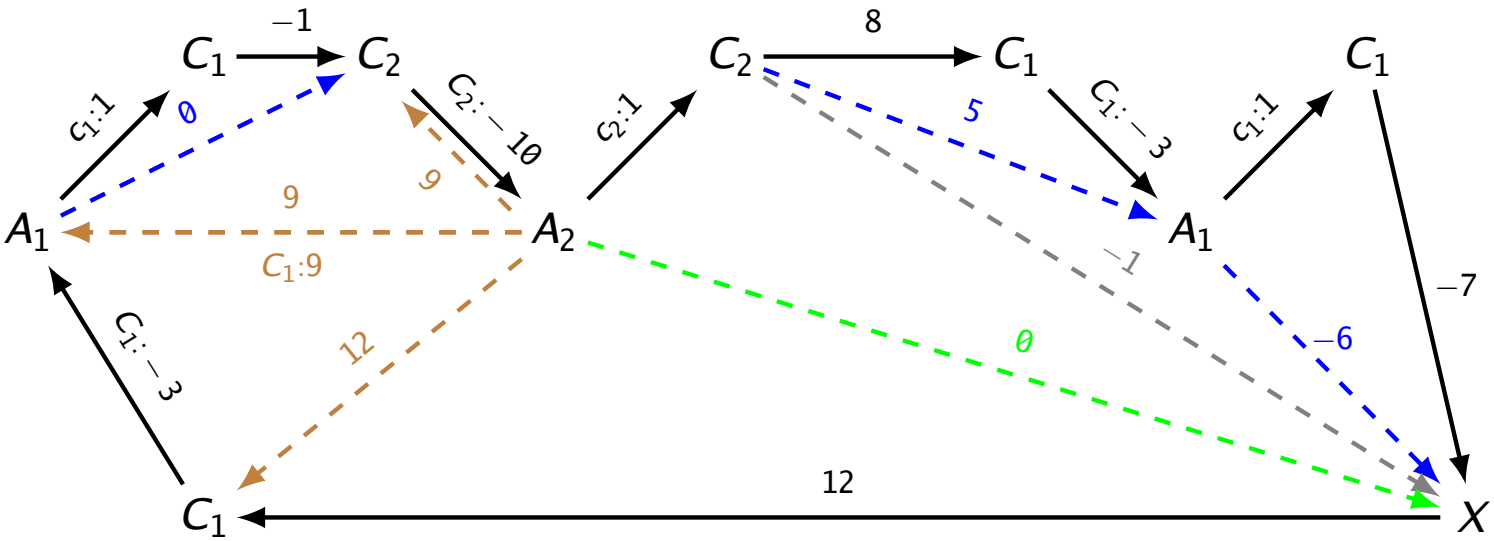
Finding a Semi–Reducible Negative Loop



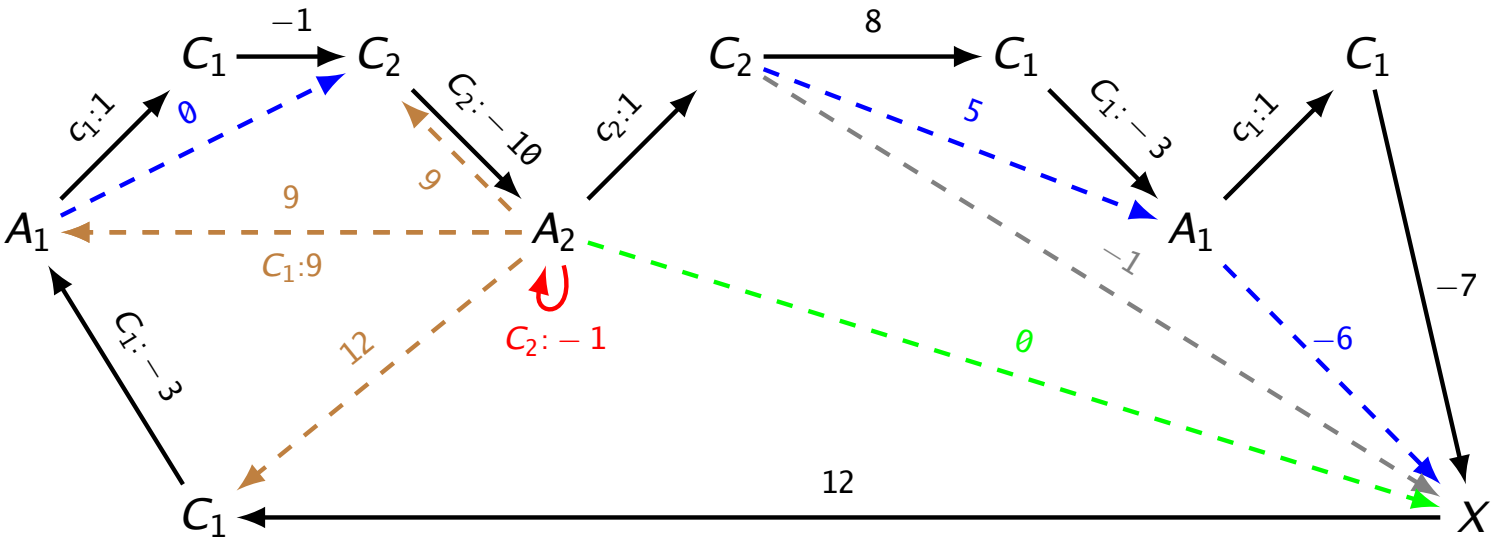
Finding a Semi–Reducible Negative Loop



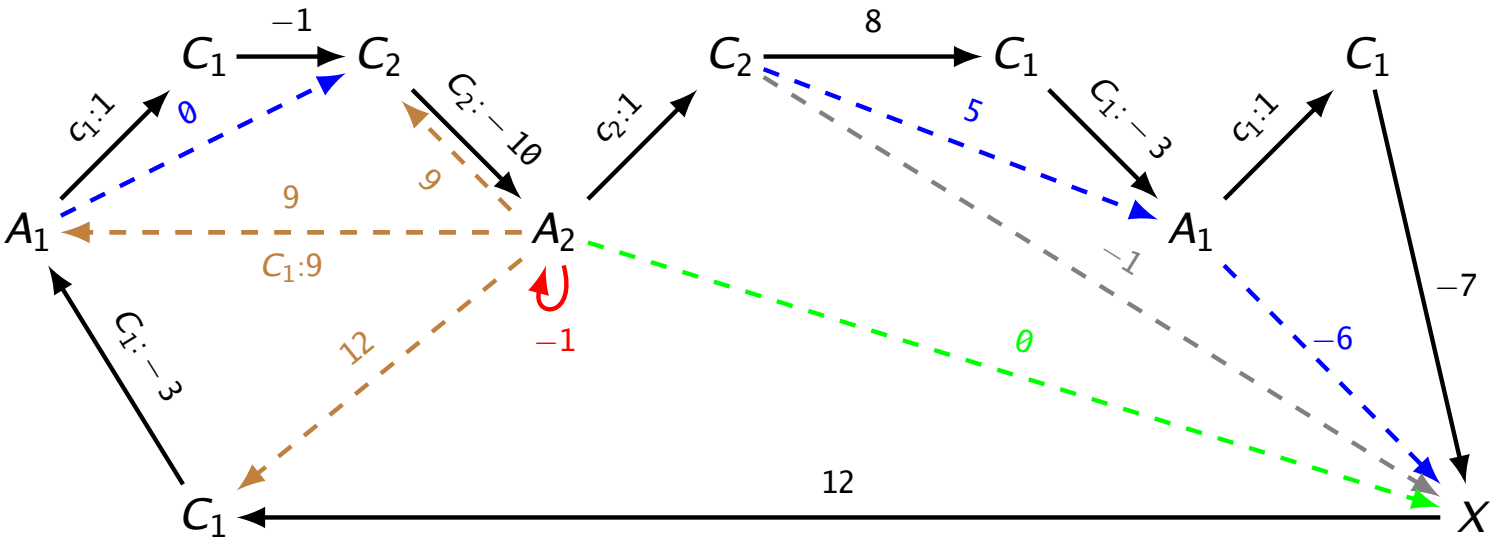
Finding a Semi–Reducible Negative Loop



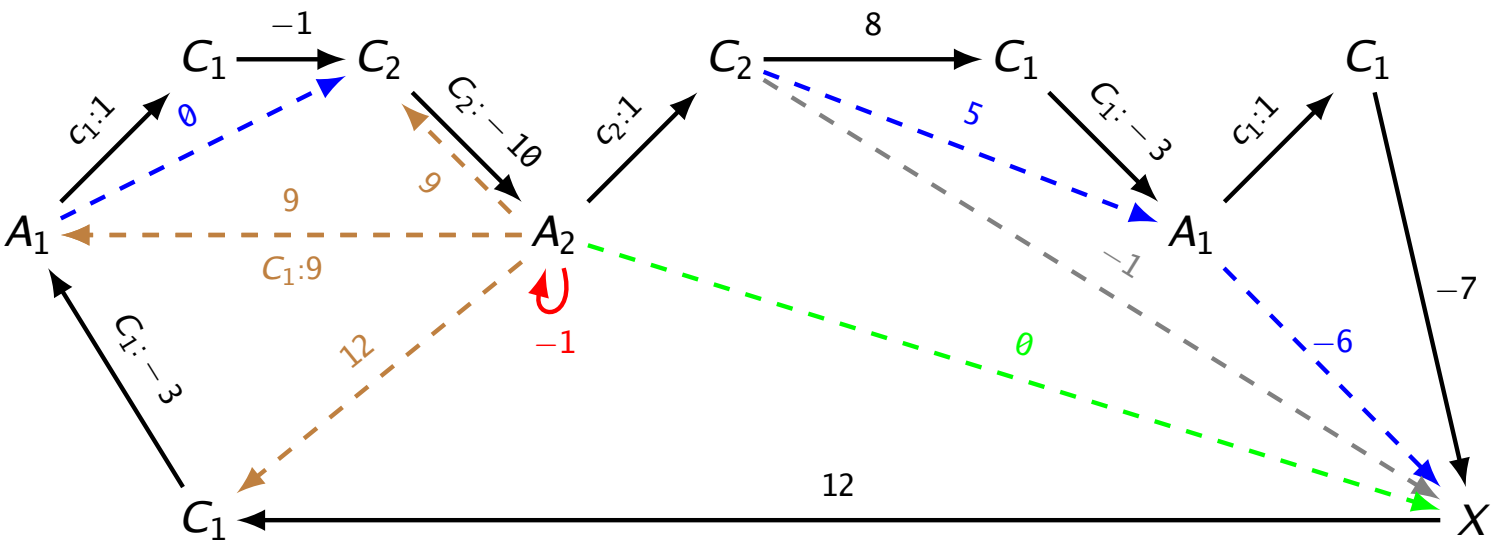
Finding a Semi–Reducible Negative Loop



Finding a Semi–Reducible Negative Loop



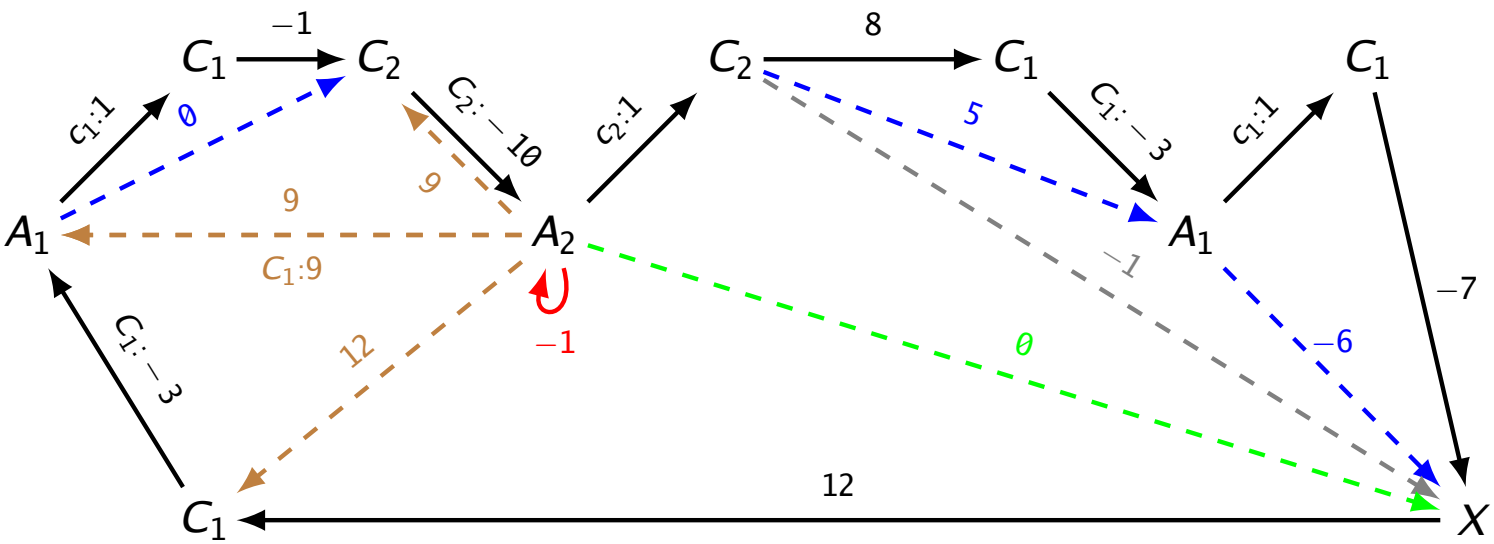
Finding a Semi–Reducible Negative Loop



- Order of processing of lower–case edges in Morris' 2006 algorithm matters!
- If process A_2C_2 first, then need two rounds;
  If process A_1C_1 first, then need only one round.
  If heuristic guesses good nesting order, $O(n^4) \rightarrow O(n^3)$.

  [Hunsberger, 2013]

Finding a Semi–Reducible Negative Loop



- Order of processing of lower–case edges in Morris' 2006 algorithm matters!
- If process A_2C_2 first, then need two rounds;
  If process A_1C_1 first, then need only one round.
  If heuristic guesses good nesting order, $O(n^4) \rightarrow O(n^3)$.

  [Hunsberger, 2013]

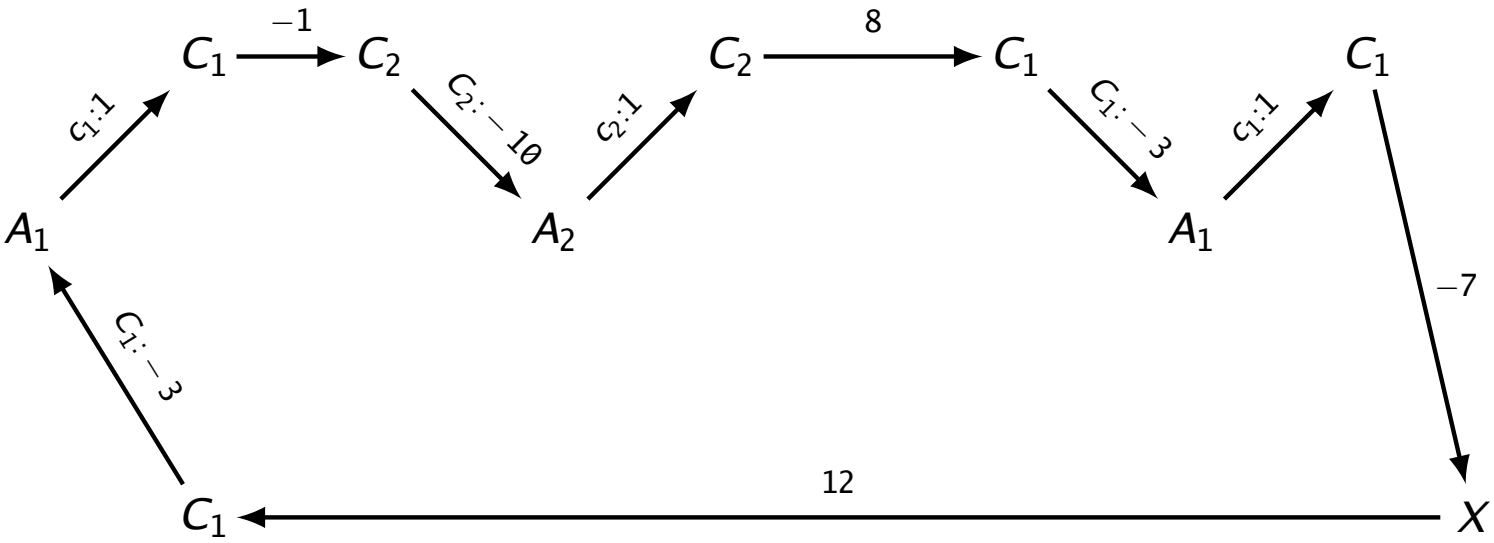
Morris' 2014 $O(n^3)$ DC-checking algorithm

Morris' 2014 $O(n^3)$ DC–checking algorithm

Key insights

- When propagating forward along "allowable" paths in the OU–graph, nested extension sub–paths cause problems
  — unless you know the order of nesting.
- A semi–reducible loop can always be reduced to a loop consisting solely of negative edges.
- Moat edges are always negative edges.
- Propagating *backward* starting from negative edges *automatically resolves the nesting order!*

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

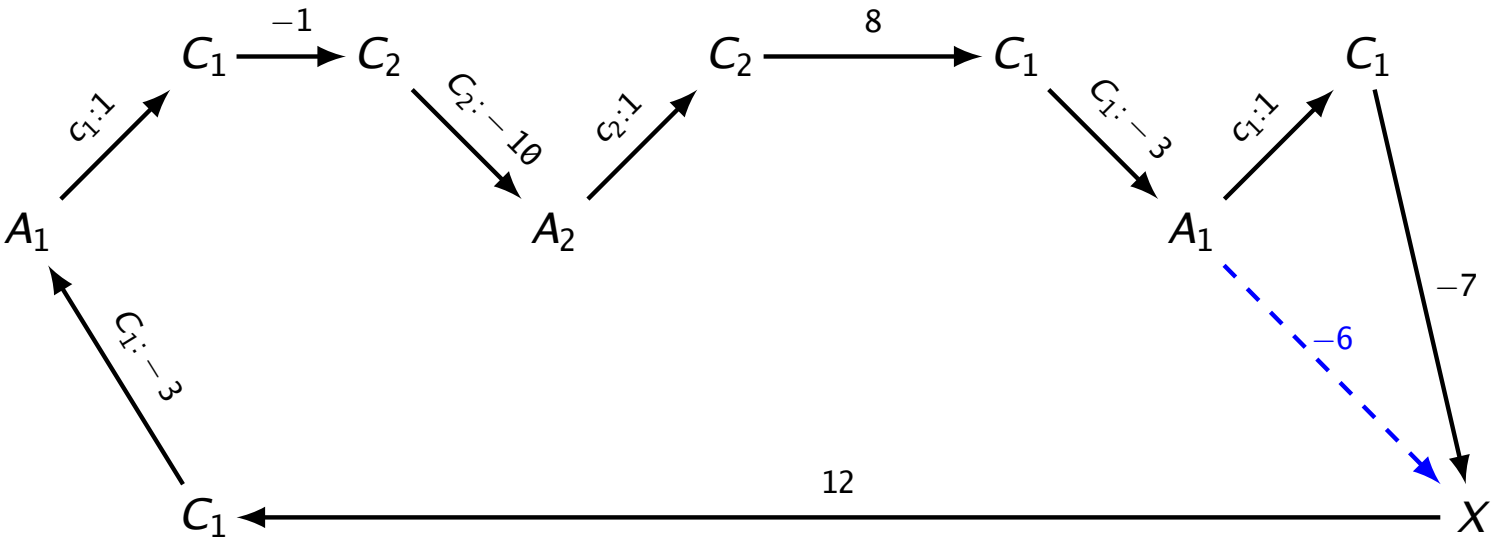
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

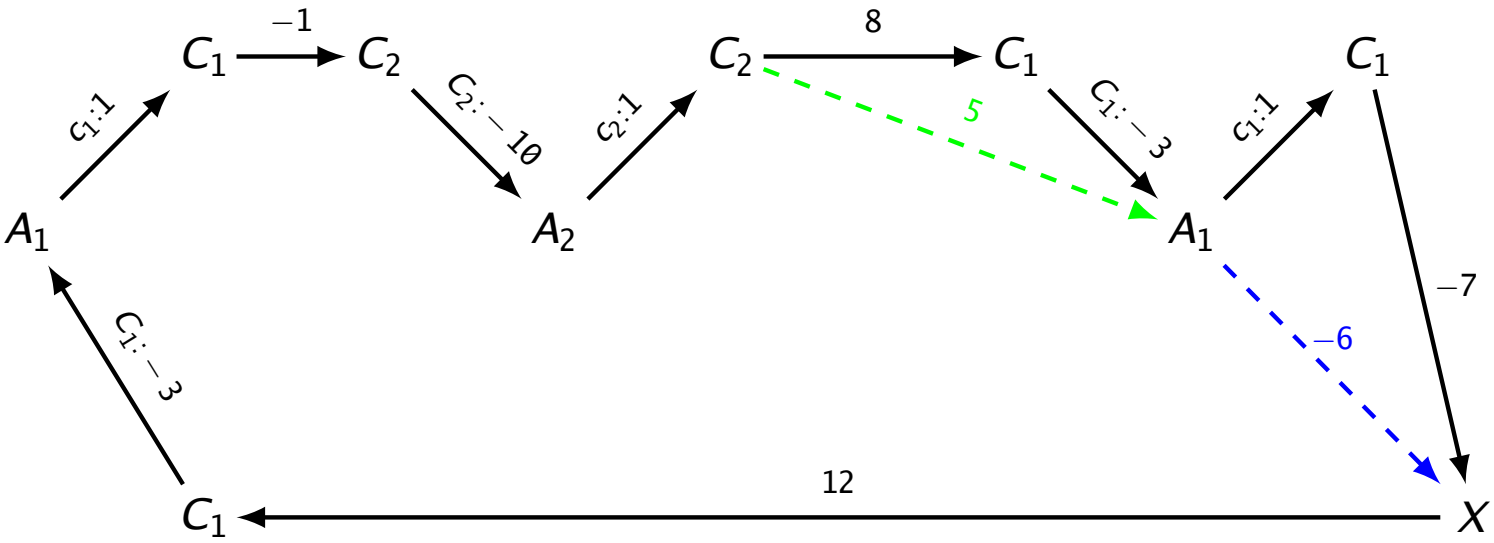
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

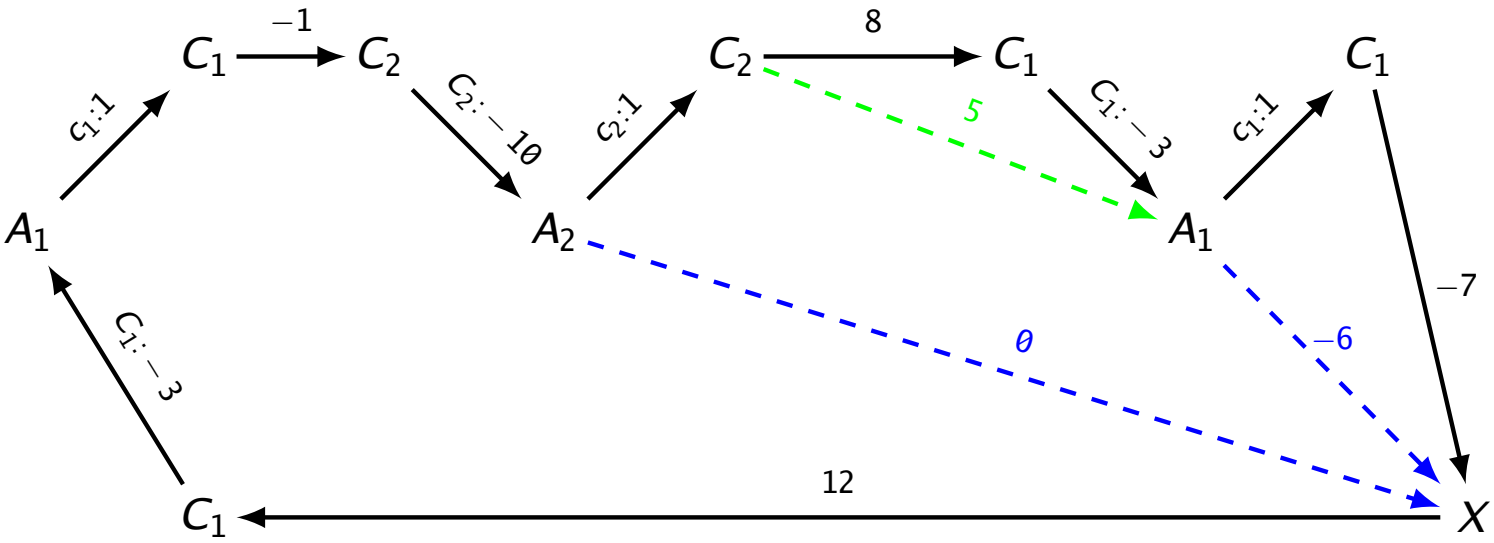
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

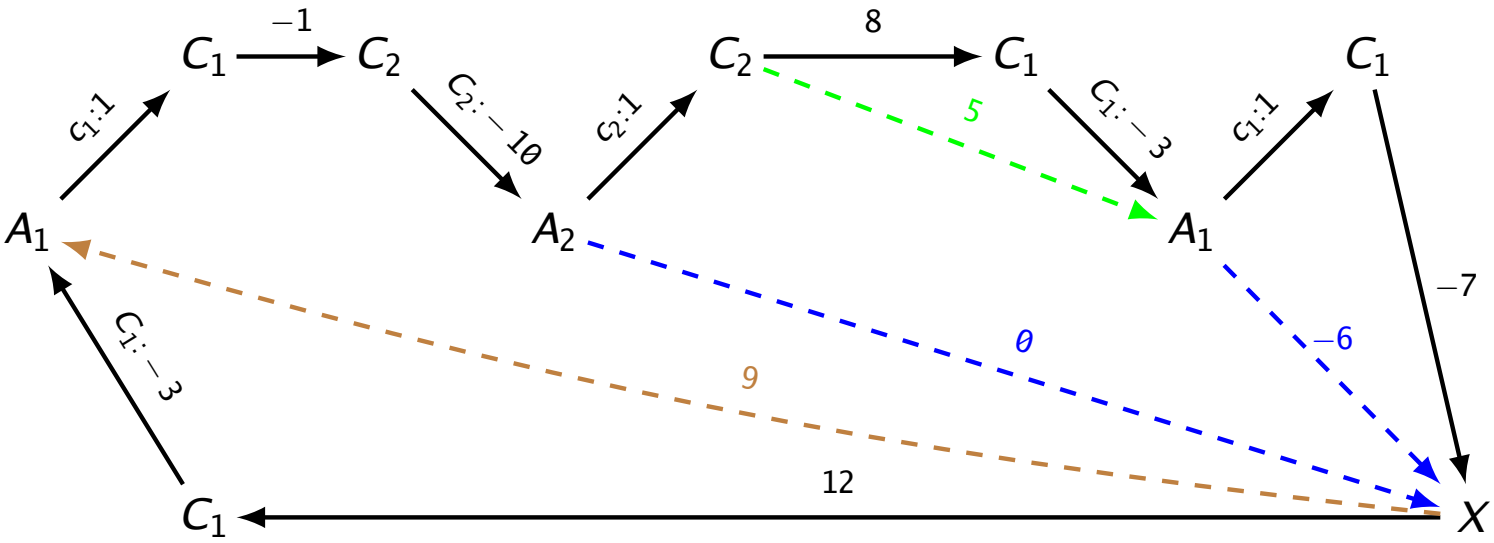
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

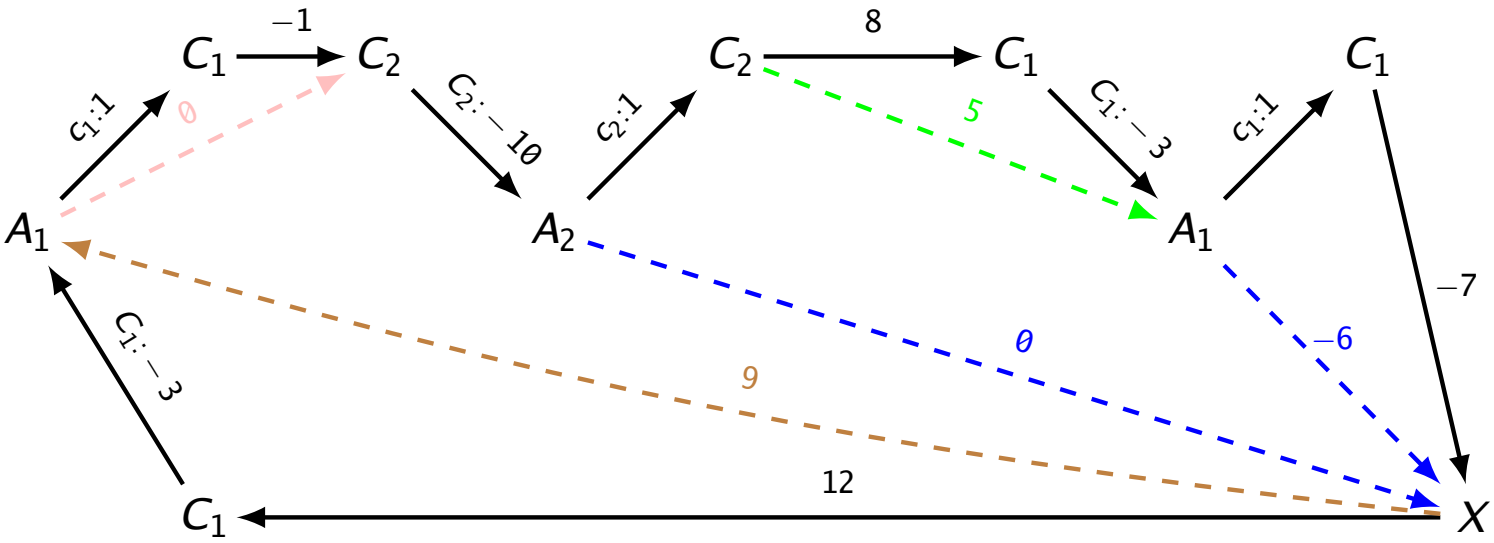
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

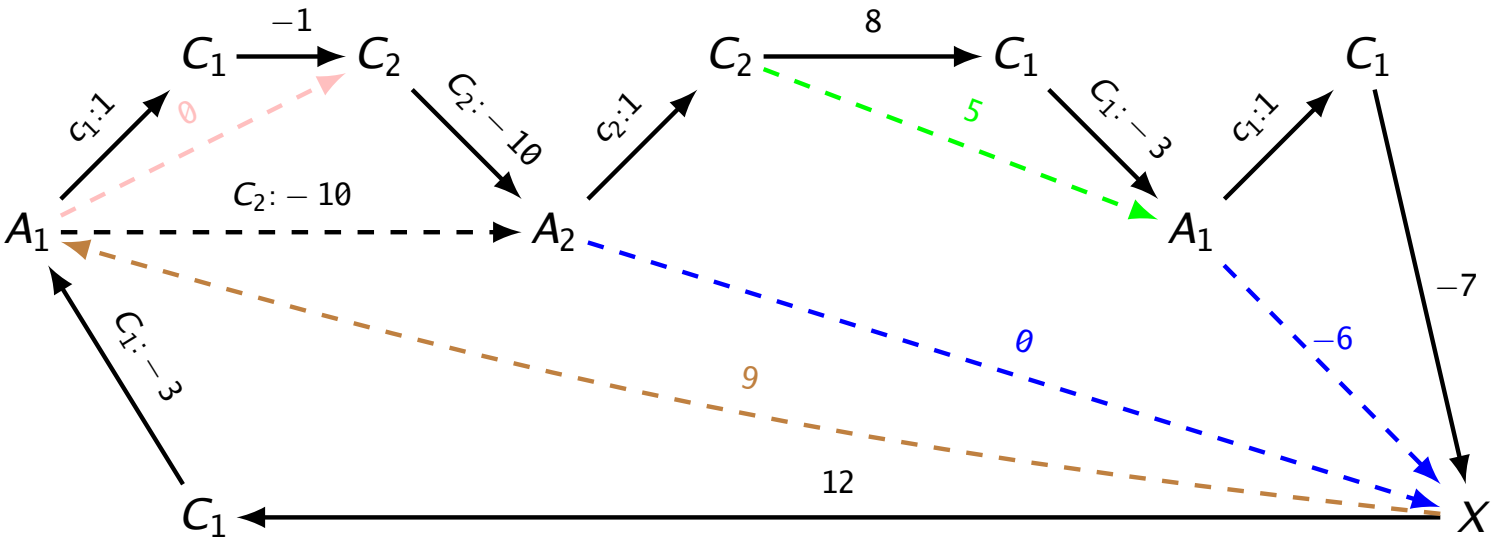
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

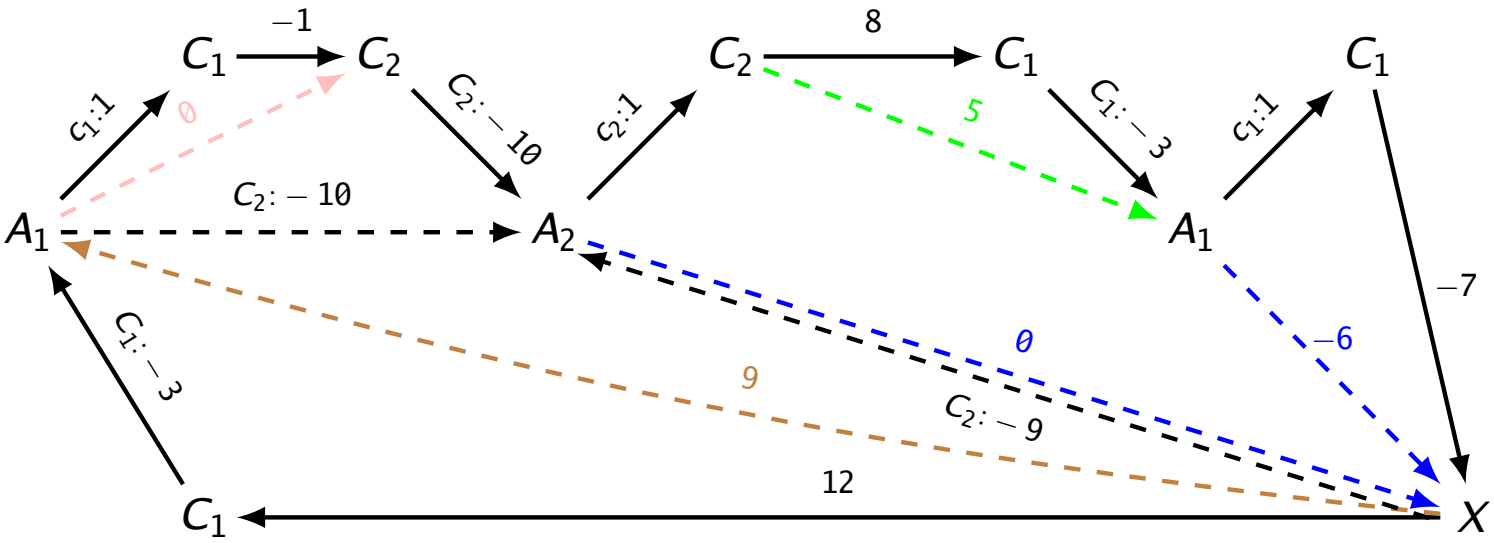
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

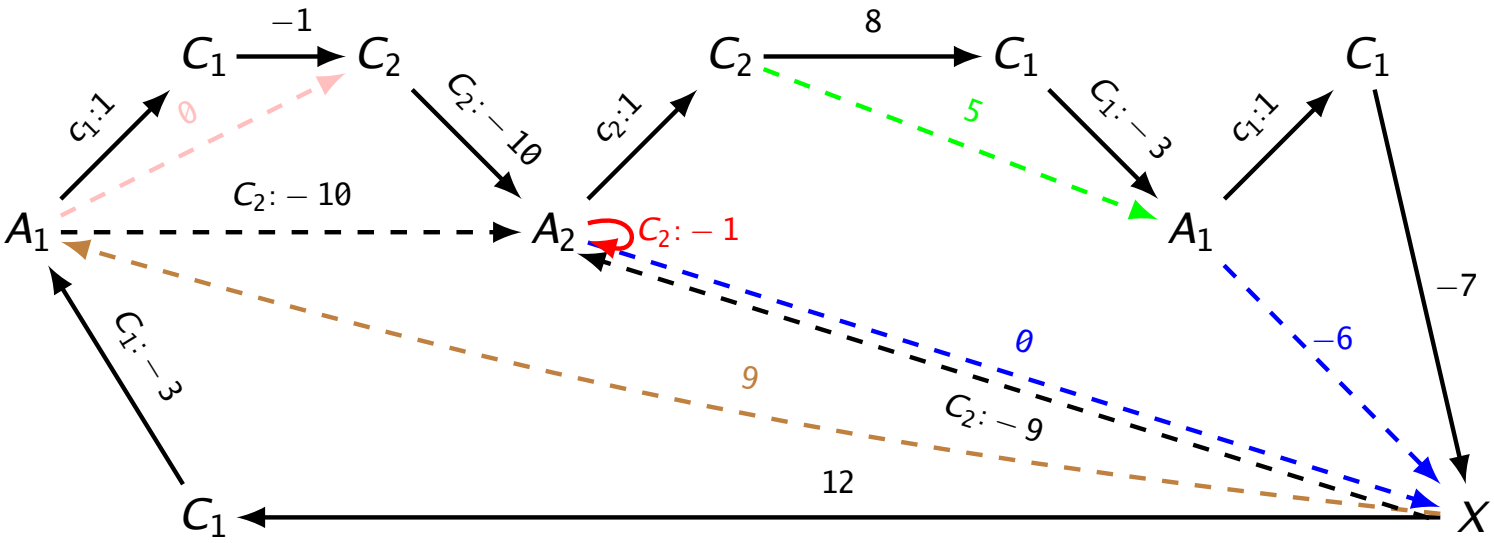
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

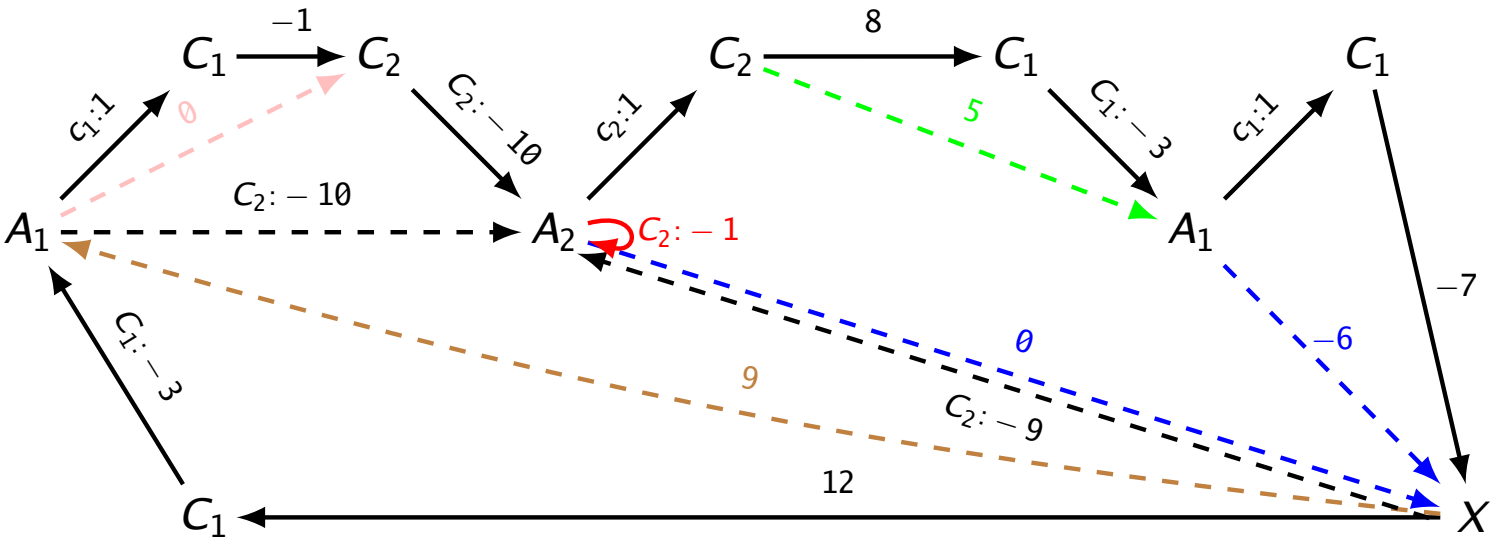
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

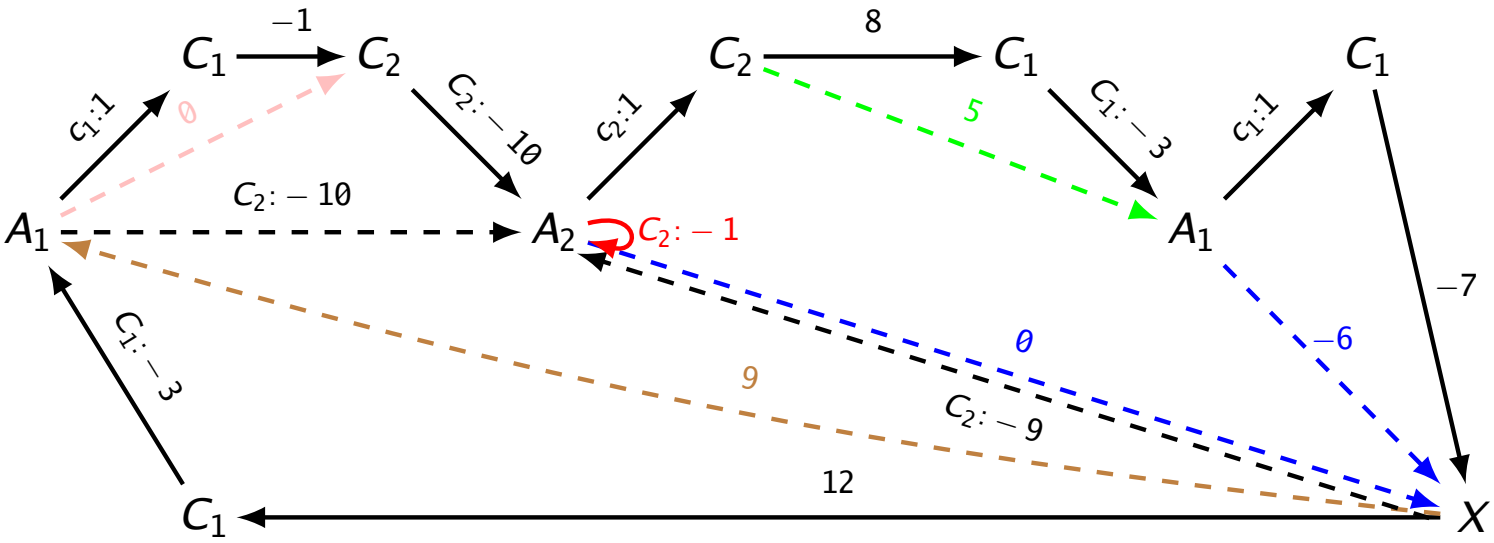
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

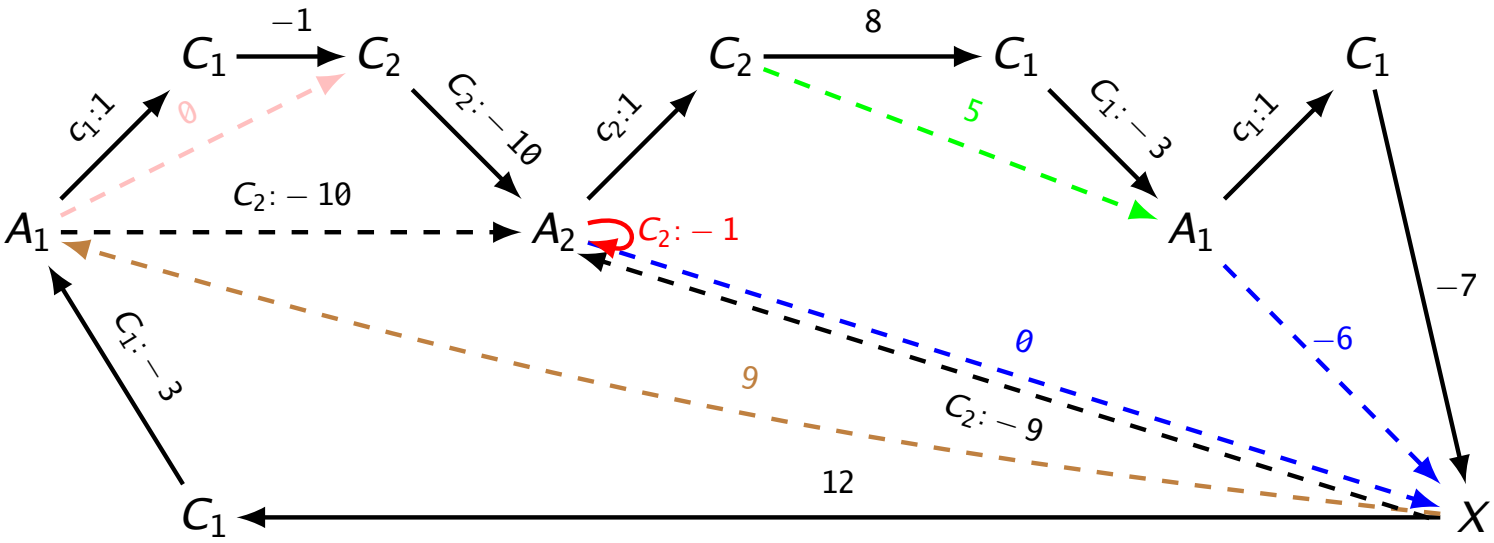
Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

Recursively process that edge.

Finding a Semi–Reducible Negative Loop



For each node that has a negative incoming edge (e.g., X) . . .

Propagate backward along *non–negative* edges . . .

But only as long as path–length stays negative.

If ever encounter another negative edge. . .

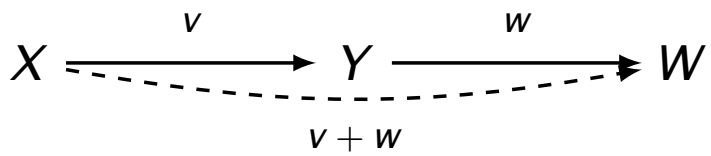
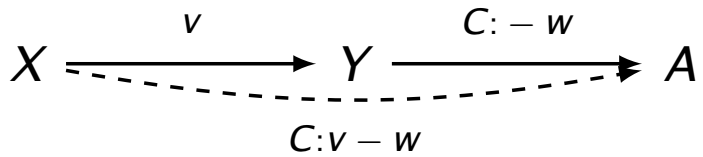
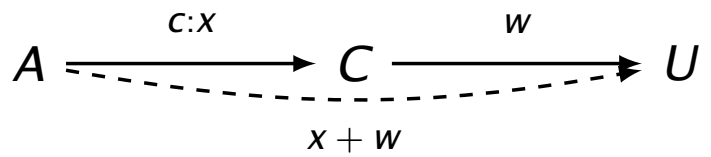
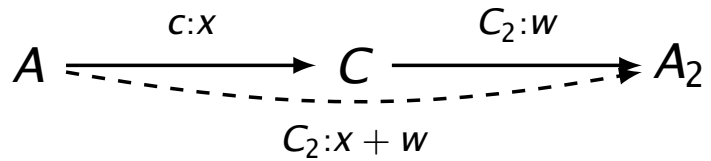
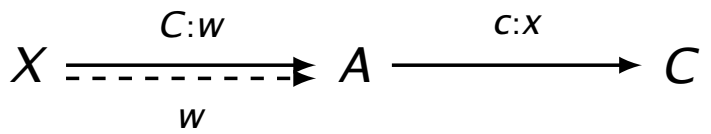
Recursively process that edge.

Morris' 2014 $O(n^3)$ DC–checking Algorithm

- Process each "negative" node only once.
  - Negative node $=$ node with negative incoming edge(s).
- Process each negative node using Dijkstra.
  - No re–weighting required!
  - Only propagate along non–negative edges.
- At most n recursive calls of Dijkstra: $O(nm + n^2 \log n)$.
- Worst case: adds $O(n^2)$ edges
  Thus, $m \rightarrow O(n^2)$ and overall complexity is $O(n^3)$.

Cairo et al.'s 2018 $O(mn + k^2 n + kn \log n)$ DC–checking Algorithm

Recall MM–2005 Propagation Rules

Rule	Graphical Representation	Applicability Conditions
No Case		(none)
Upper Case		(none)
Lower Case		$w < 0$
Cross Case		$C_2 \not\equiv C$, $w < 0$
Label Removal		$w \geq -x$

MMV–2001's General Unordered Reduction

Graphical Representation	Applicability Condition
$V \xrightarrow[-x]{C:w} A \xrightarrow{c:x} C$	$w < -x$ (eqiuv., $-w > x$)
$V \xrightarrow[-3]{C:-8} A \xrightarrow{c:3} C$	$-8 < -3$ (eqiuv., $8 > 3$)

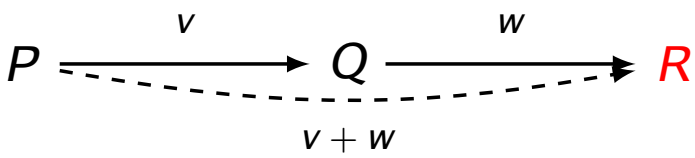
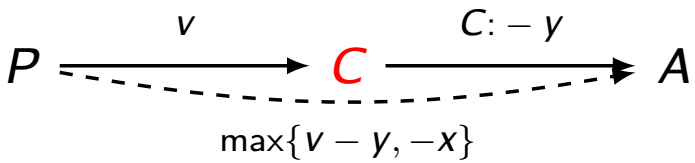
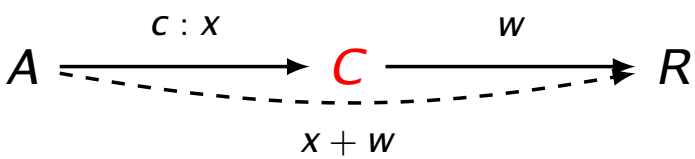
- While C remains unexecuted, V must wait at least $-w$ after A.
- But C cannot execute sooner than x after A.
- Therefore, in every situation, V must wait at least x after A.

The GUR rule is *not* length–preserving.

[Morris et al., 2001]

Cairo et al.'s 2018 DC–checking Algorithm

The RUL$^-$ propagation rules

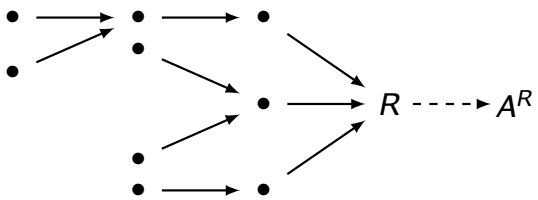
Rule	Graphical representation	Applicability Conditions
RELAX$^-$		$Q \in \mathcal{T}_X, w < y^R - x^R, R \in \mathcal{T}_C$
UPPER$^-$		(none)
LOWER$^-$		$C \not\equiv R, w < y^R - x^R, R \in \mathcal{T}_C$

- Only generates *ordinary* edges!
- Propagations always involve 1–2 contingent time–points ($k < n$).
- UPPER$^-$ rule is *not* length–preserving.

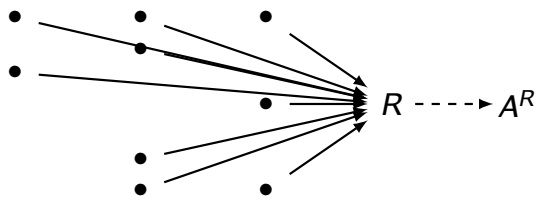
Cairo et al.'s 2018 DC–checking Algorithm
Overview

- INIT: Bellman–Ford to compute a potential function for the LO–graph (i.e., the graph containing only lower–case and ordinary edges).
- Reduces away *upper–case* edges, not lower–case edges.
- For each contingent time–point R:
  - Propagates backward from R using LOWER$^-$ and RELAX$^-$ rules.
  - For each new edge XR, applies UPPER$^-$ rule to XRA^R.
- Backward propagation done using Dijkstra on re–weighted LO–graph.
- After each round, inserts new edges and updates potential function.

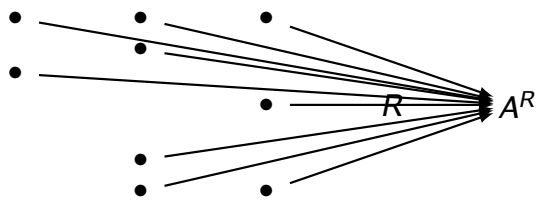
Cairo et al.'s 2018 DC–checking Algorithm
Overview (continued)



(a) Propagating back from R



(b) New edges terminating at R



(c) ApplyUpper

- Uses push–down stacks to ensure that each contingent time–point is processed at most twice; and to detect negative cycles.
- If backward propagation from R encounters an upper–case edge CA, where C has not yet been processed, then it suspends processing of R, and instead processes C.
- Only resumes processing of R when all previously encountered upper–case edges have been processed.

Cairo et al.'s 2018 DC–checking Algorithm
Overview (continued)

- Faster than Morris' 2014 $O(n^3)$ algorithm on sparse graphs, for example, in cases where Morris' algorithm generates $O(n^2)$ edges.

Executing STNUs Efficiently

STN with Uncertainty
Flexible Execution of STNUs

- Lower–case edges are only needed during constraint propagation, to generate additional ordinary and upper–case edges.
- After propagation completes, execution phase can begin, and lower–case edges can be ignored.
- Therefore, the fully–propagated OU–graph contains all the information needed to successfully execute a DC STNU.
- Executing an STNU is similar to executing STNs, except that:
  - Lower bound for X also depends on up to k *wait* constraints.
  - When a contingent C executes, all upper–case edges labeled by C must be deleted, requiring lower bounds to be updated.
- A DC STNU can be flexibly executed, incrementally computing updates in *AllMax* graph using $O(n^2)$–time per execution event, $O(n^3)$–time overall.*

* [Hunsberger, 2013, 2015]

STN with Uncertainty
STNU Dispatchability

- Recall: A projection of an STNU is the STN that results from fixing the durations all contingent links to legal values.
- Definition: An STNU is dispatchable if each of its STN *projections* is dispatchable (as an STN).
- Dispatchability same as for STNs, except that:
  - Contingent time–points are not controllable; and
  - There are *wait* constraints:
    "As long as C unexecuted, X must wait at least 5 after A."
- Theorem: A dispatchable STNU is DC.*

* [Morris, 2014]

- For a DC STNU, Morris' $O(N^3)$–time DC–checking algorithm generates a dispatchable STNU.*
- Corollary: For a DC STNU, the STNU graph generated by exhaustively applying the constraint propagation rules from Morris et al. (2005) is dispatchable.

*[Morris, 2014]

STN with Uncertainty
STNU Summary

- The theory of STNUs (dynamic controllability, dispatchability, flexible execution) has been advanced dramatically over the past few years.
- Many important contributions from Paul Morris and colleagues.
- STNUs are ready for prime time!

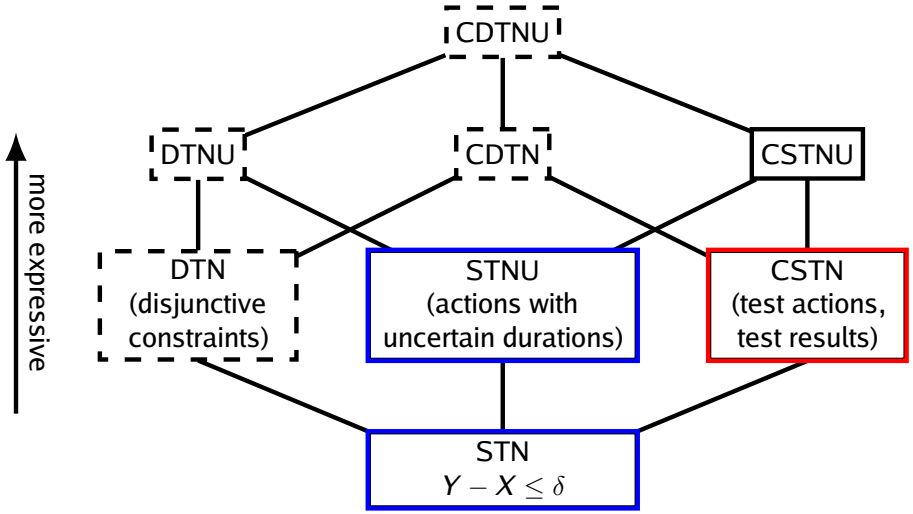
Conditional STNs
Motivation

- Many actions generate information (e.g., medical tests, opening a box, monitoring traffic).
- The generated information is generally not known in advance, but discovered in real time.
- Some actions only make sense in certain scenarios (e.g., don't give drug if test result is negative).
- An execution strategy could be more flexible if it could react dynamically to generated information.

Conditional STNs
Motivation (ctd.)

- Many businesses using *workflow management systems* to automate manufacturing processes.
- Hospitals can use workflows to represent possible treatment pathways for a patient.
- CSTNs can serve as the temporal foundation for workflow management systems.

Conditional STNs (CSTNs)*



- Time–points and constraints as in STNs
- Observation time–points generate truth values for propositional letters
- Time–points and constraints labeled by conjunctions of propositional letters

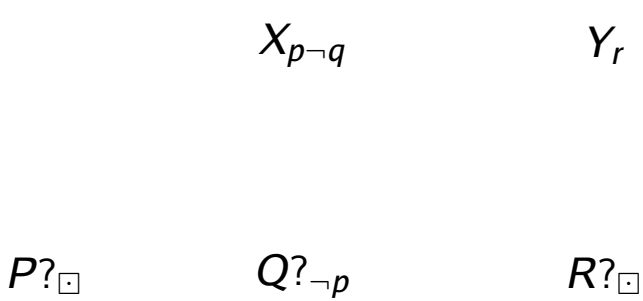
* [Tsamardinos et al., 2003]

Conditional STNs
Propositional Labels

- Propositional letters: $p, q, r, s, t, \ldots$
- Each p has corresponding observation time–point, $P?$.
  Executing $P?$ generates truth value for p
- Label: conjunction of literals (e.g., $p(\neg q)r$)
- A scenario specifies values for *all* letters
  (e.g., $p = true$, $q = false$, $r = true$).
- The real scenario is only revealed incrementally.
- Time–points and constraints* can be labeled;
  they only apply in scenarios where their labels are true.

* [Hunsberger et al., 2015]

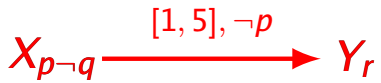
Conditional STNs
Well–definedness (WD) properties*



- WD1 (Label Coherence): the label on a constraint must be satisfiable and must entail the labels on its endpoints.
- WD2 (Node Label Honesty):
  - The label ℓ on a node must entail the labels on all observation time–points whose corresponding prop'l. letters appear in L; and
  - the node must occur $\epsilon \geq 0$ after such observation time–points.
- WD3 (Edge Label Honesty): the label L on an edge must entail the labels on all observation time–points whose corresponding propositional letters appear in L.

* Implicit in Tsamardinos et al. (2003), formalized in Hunsberger et al. (2015).

Conditional STNs
Well–definedness (WD) properties*

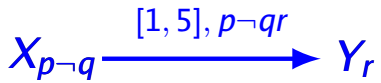


$P?_{\boxdot}$ $Q?_{\neg p}$ $R?_{\boxdot}$

- WD1 (Label Coherence): the label on a constraint must be satisfiable and must entail the labels on its endpoints.
- WD2 (Node Label Honesty):
  - The label ℓ on a node must entail the labels on all observation time–points whose corresponding prop'l. letters appear in L; and
  - the node must occur $\epsilon \geq 0$ after such observation time–points.
- WD3 (Edge Label Honesty): the label L on an edge must entail the labels on all observation time–points whose corresponding propositional letters appear in L.

* Implicit in Tsamardinos et al. (2003), formalized in Hunsberger et al. (2015).

Conditional STNs
Well–definedness (WD) properties*

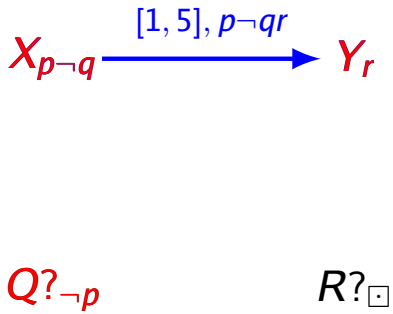


$P?_{\boxdot}$ $Q?_{\neg p}$ $R?_{\boxdot}$

- WD1 (Label Coherence): the label on a constraint must be satisfiable and must entail the labels on its endpoints.
- WD2 (Node Label Honesty):
  - The label ℓ on a node must entail the labels on all observation time–points whose corresponding prop'l. letters appear in L; and
  - the node must occur $\epsilon \geq 0$ after such observation time–points.
- WD3 (Edge Label Honesty): the label L on an edge must entail the labels on all observation time–points whose corresponding propositional letters appear in L.

* Implicit in Tsamardinos et al. (2003), formalized in Hunsberger et al. (2015).

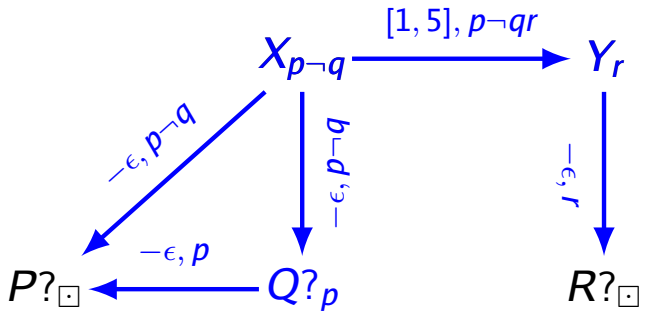
Conditional STNs
Well–definedness (WD) properties*



- WD1 (Label Coherence): the label on a constraint must be satisfiable and must entail the labels on its endpoints.
- WD2 (Node Label Honesty):
  - The label ℓ on a node must entail the labels on all observation time–points whose corresponding prop'l. letters appear in L; and
  - the node must occur $\epsilon \geq 0$ after such observation time–points.
- WD3 (Edge Label Honesty): the label L on an edge must entail the labels on all observation time–points whose corresponding propositional letters appear in L.

* Implicit in Tsamardinos et al. (2003), formalized in Hunsberger et al. (2015).

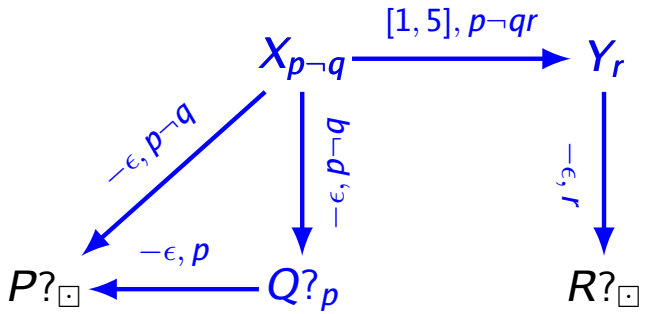
Conditional STNs
Well–definedness (WD) properties*



- WD1 (Label Coherence): the label on a constraint must be satisfiable and must entail the labels on its endpoints.
- WD2 (Node Label Honesty):
  - The label ℓ on a node must entail the labels on all observation time–points whose corresponding prop'l. letters appear in L; and
  - the node must occur $\epsilon \geq 0$ after such observation time–points.
- WD3 (Edge Label Honesty): the label L on an edge must entail the labels on all observation time–points whose corresponding propositional letters appear in L.

* Implicit in Tsamardinos et al. (2003), formalized in Hunsberger et al. (2015).

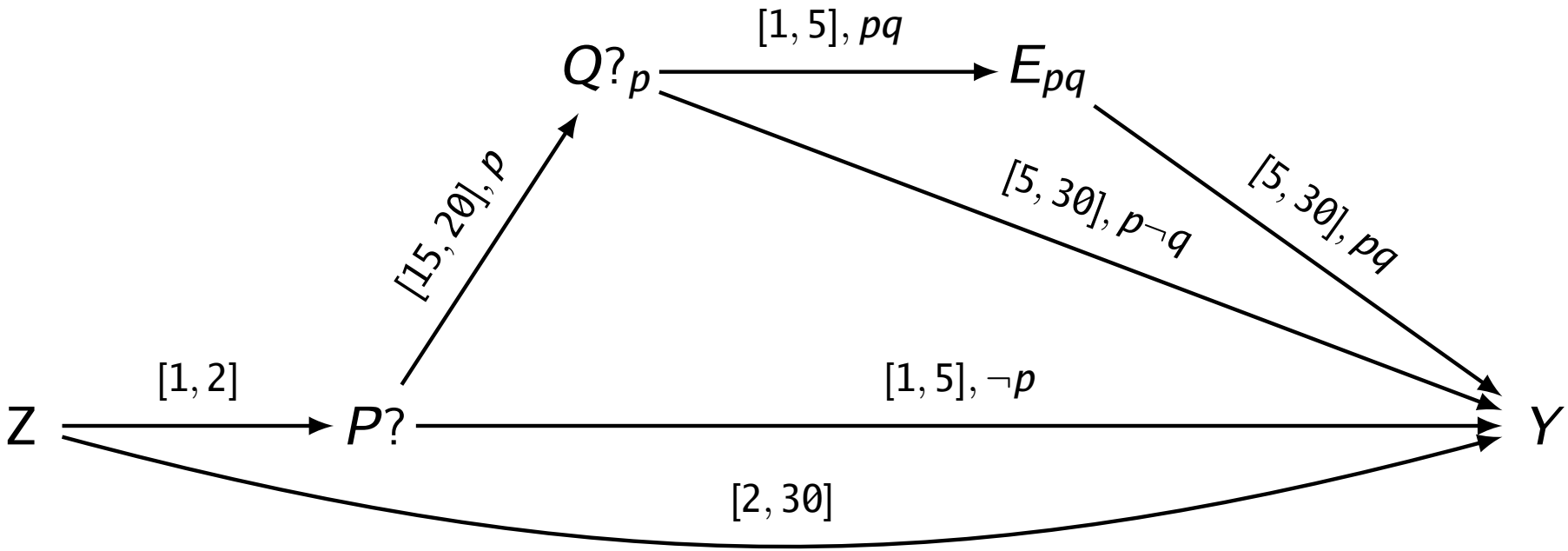
Conditional STNs
Well–definedness (WD) properties*



- WD1 (Label Coherence): the label on a constraint must be satisfiable and must entail the labels on its endpoints.
- WD2 (Node Label Honesty):
  - The label ℓ on a node must entail the labels on all observation time–points whose corresponding prop'l. letters appear in L; and
  - the node must occur $\epsilon \geq 0$ after such observation time–points.
- WD3 (Edge Label Honesty): the label L on an edge must entail the labels on all observation time–points whose corresponding propositional letters appear in L.

* Implicit in Tsamardinos et al. (2003), formalized in Hunsberger et al. (2015).

Conditional STNs
Sample CSTN



$P?$ and $Q?$ represent tests for a patient.
$Q?$ is a *child* of $P?$: only executed in scenarios where $p = true$.

- Dynamic Execution Strategy: execution decisions can react to observations.
- A CSTN is *dynamically consistent* (DC) if there exists a dynamic execution strategy that guarantees that all *relevant* constraints will be satisfied no matter which scenario is incrementally revealed over time.

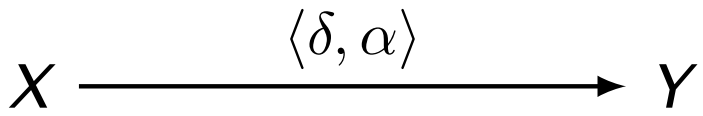
Conditional STNs
Approaches to DC Checking

- Convert to Disjunctive Temporal Problem
  [Tsamardinos et al., 2003]
- Convert to controller–synth. problem for Timed Game Automaton
  [Cimatti et al., 2014]
- Convert to Hyper Temporal Network consistency problem
  [Comin and Rizzi, 2015]
- Propagate labeled constraints
  [Hunsberger et al., 2015]

Conditional STNs
DC Checking via Propagation

- Propagate *labeled* constraints
  — Motivated by related work on STNs with choice [Conrad and Williams, 2011]
- Introduce new kind of literals and labels:
  Q–literals (e.g., $?p$) and *Q–labels* (e.g., $p\neg q(?r)s$)
- Analysis of *negative q–loops* and *negative q–stars*

Conditional STNs
Labeled Constraints



$Y - X \leq \delta$ must hold in every scenario where α is true.

(If $\alpha = \boxdot$, then $Y - X \leq \delta$ must hold in all scenarios.)

Conditional STNs
Propagation Rules for CSTNs

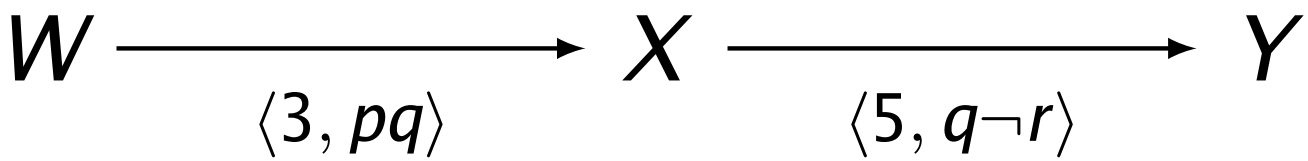
Labeled Propagation: LP and qLP

Label Modification: R_0 and qR_0

Label "Spreading": R_3^* and qR_3^*

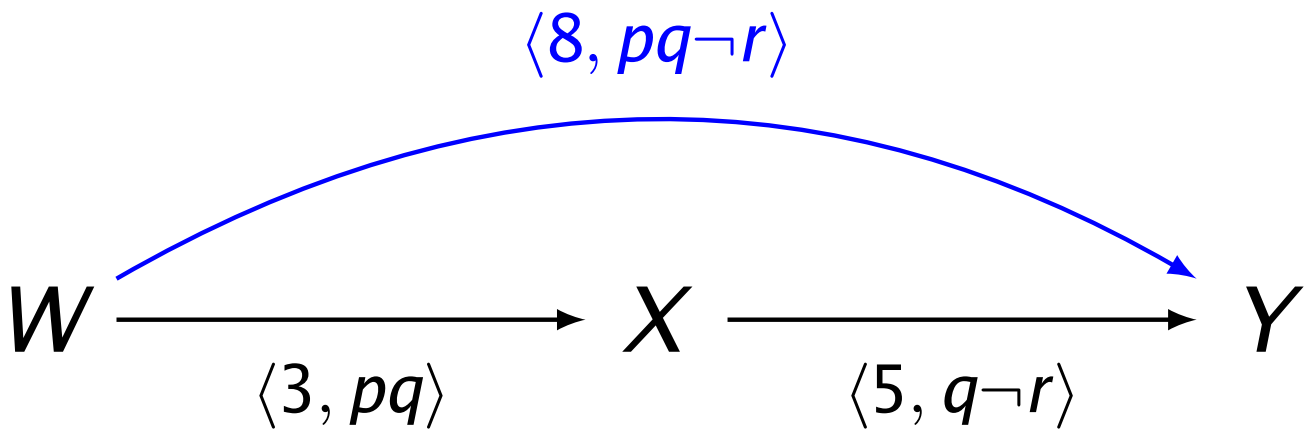
(The "q" rules propagate q–labeled constraints.)

Conditional STNs
The LP Rule



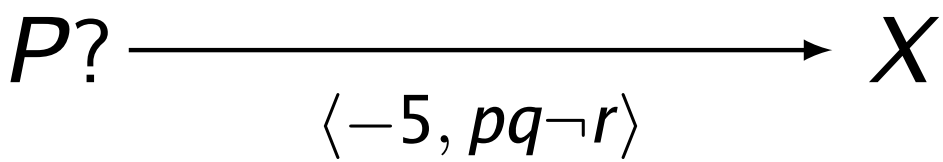
Labels of two pre–existing edges are conjoined;
The resulting label must be consistent.

Conditional STNs
The LP Rule



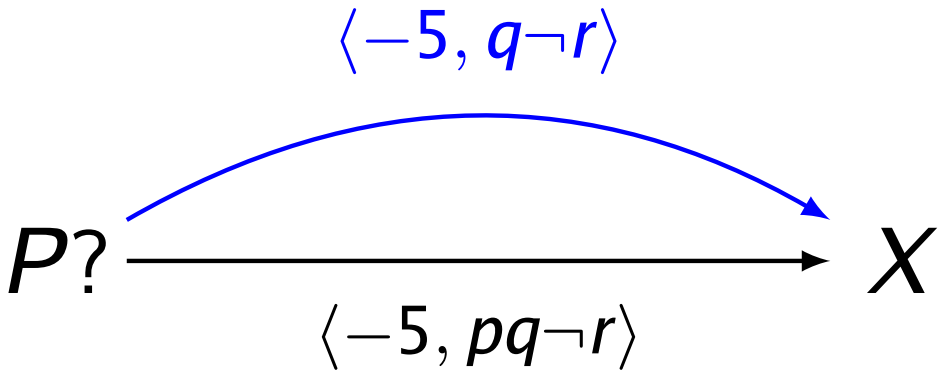
Labels of two pre–existing edges are conjoined;
The resulting label must be consistent.

Conditional STNs
The R_0 Rule



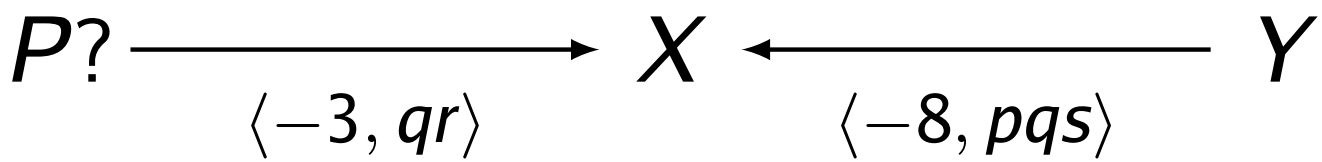
Edge weight must be negative;
Any occurrence of p (or $\neg p$) removed from label.

Conditional STNs
The R_0 Rule



Edge weight must be negative;
Any occurrence of p (or $\neg p$) removed from label.

Conditional STNs
The R_3^* Rule



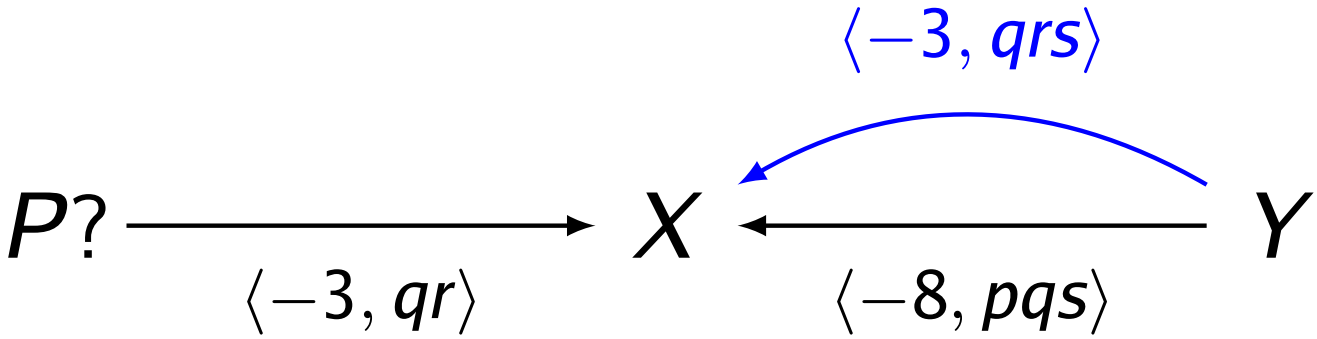
Pre–existing labels must be consistent;

Generated label is conjunction of pre–existing labels
— minus any occurrence of p (or $\neg p$);

Lefthand weight must be negative;

Generated weight is max of pre–existing weights.

Conditional STNs
The R_3^* Rule



Pre–existing labels must be consistent;

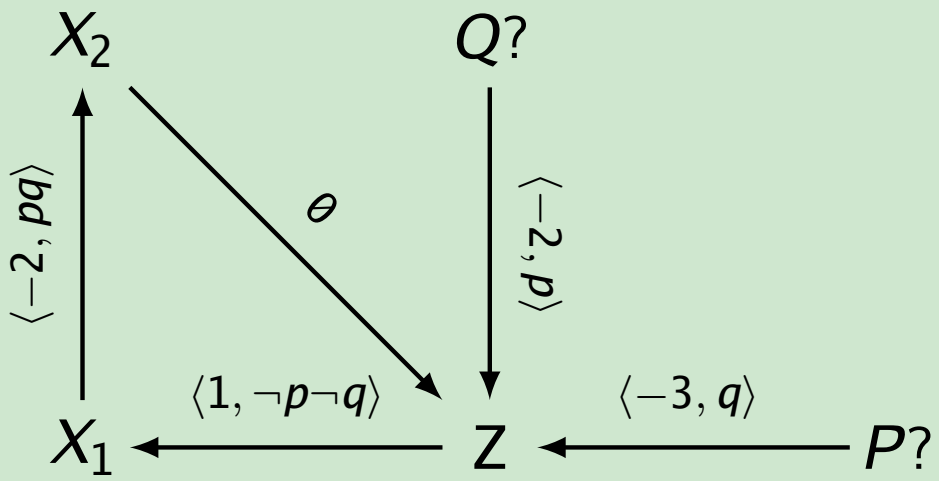
Generated label is conjunction of pre–existing labels
— minus any occurrence of p (or $\neg p$);

Lefthand weight must be negative;

Generated weight is max of pre–existing weights.

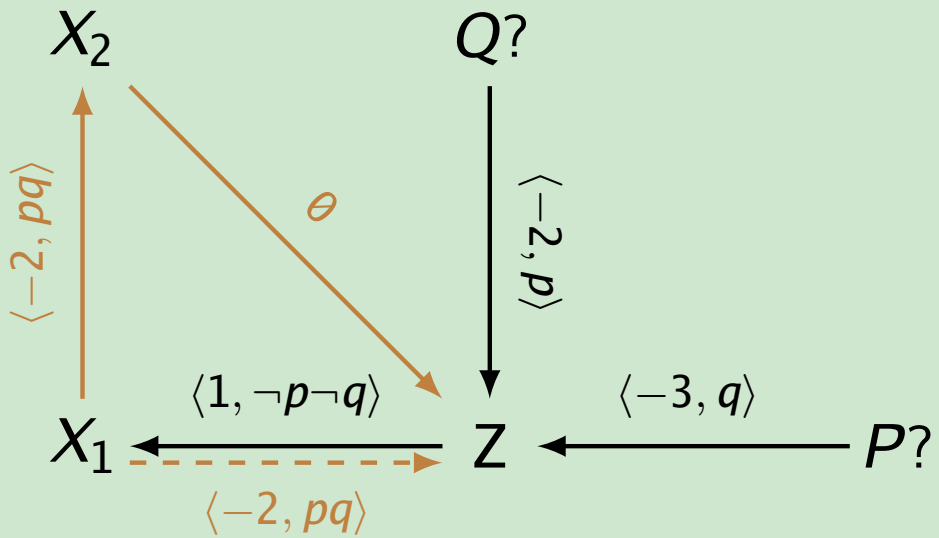
Conditional STNs
Example: Non–DC Instance

Example 2 (Non–DC Instance)



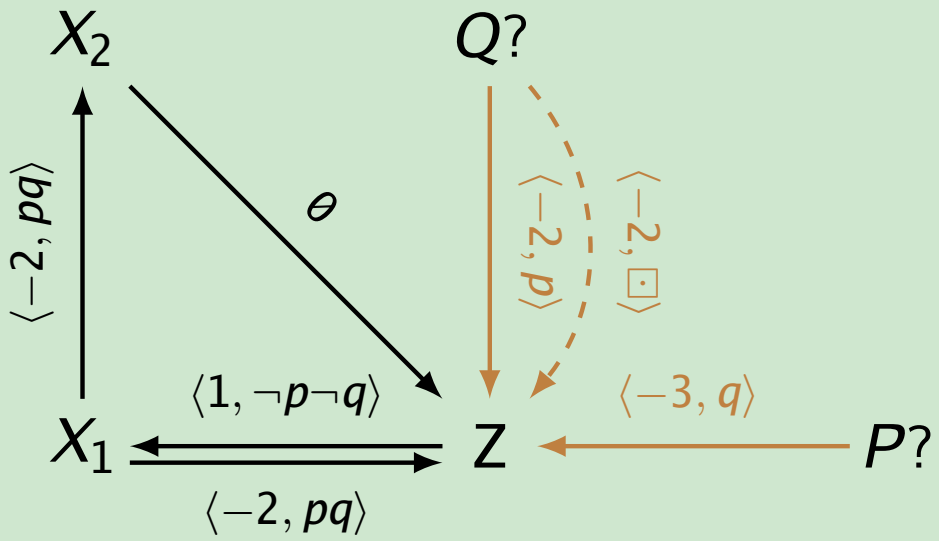
Conditional STNs
Example: Non–DC Instance

Example 2 (Non–DC Instance)



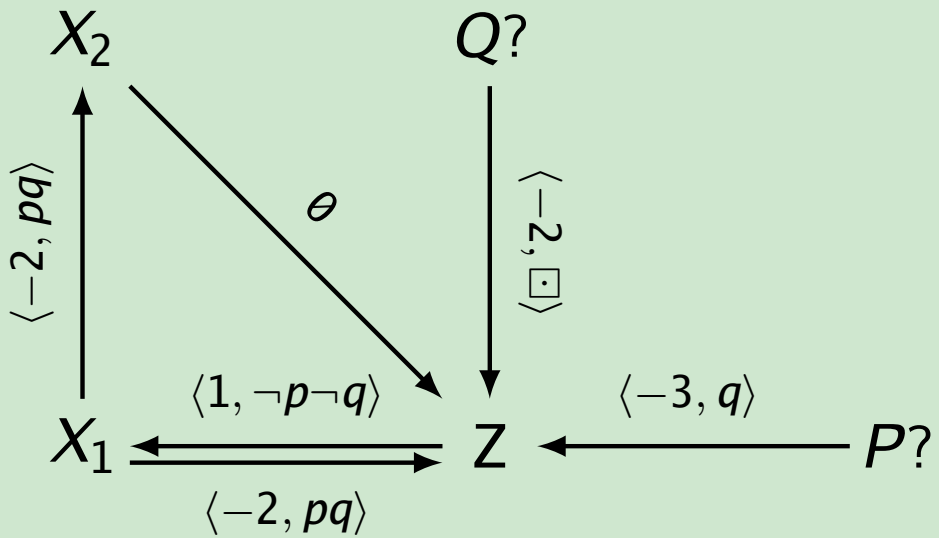
Conditional STNs
Example: Non–DC Instance

Example 2 (Non–DC Instance)



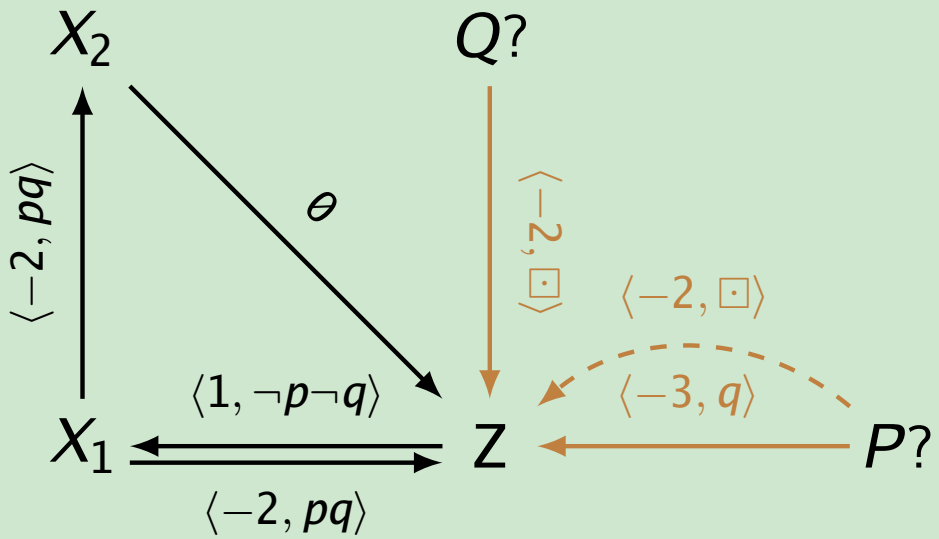
Conditional STNs
Example: Non–DC Instance

Example 2 (Non–DC Instance)



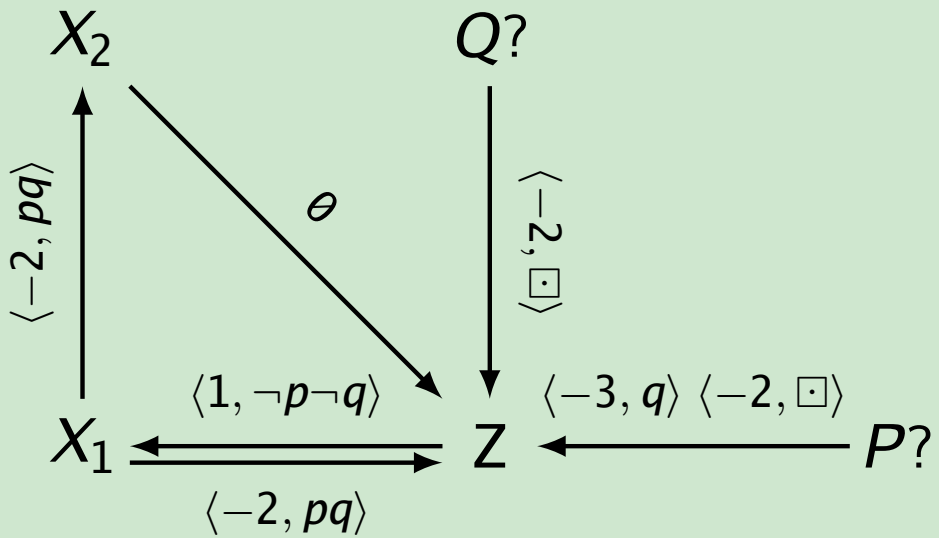
Conditional STNs
Example: Non–DC Instance

Example 2 (Non–DC Instance)



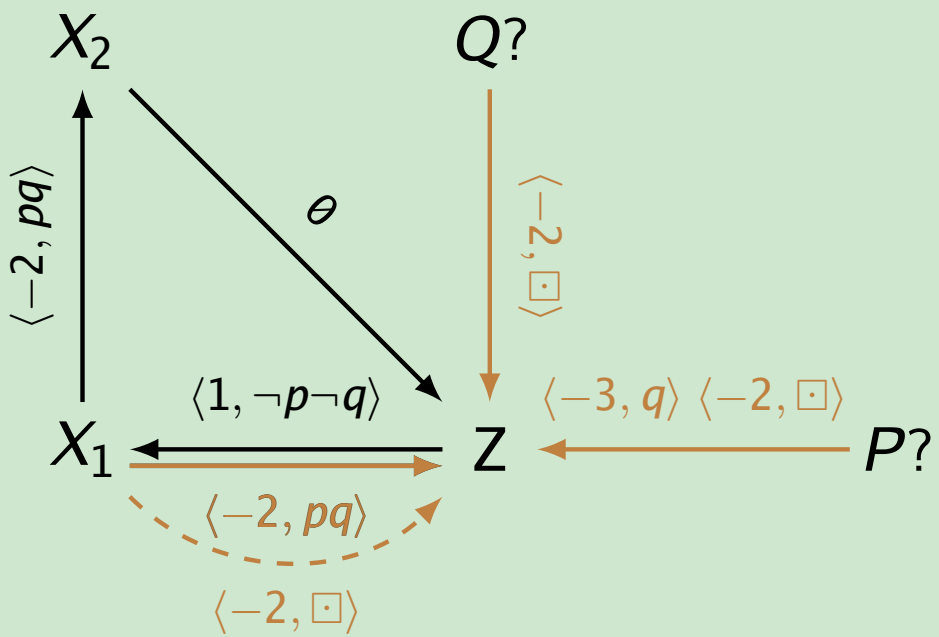
Conditional STNs
Example: Non–DC Instance

Example 2 (Non–DC Instance)



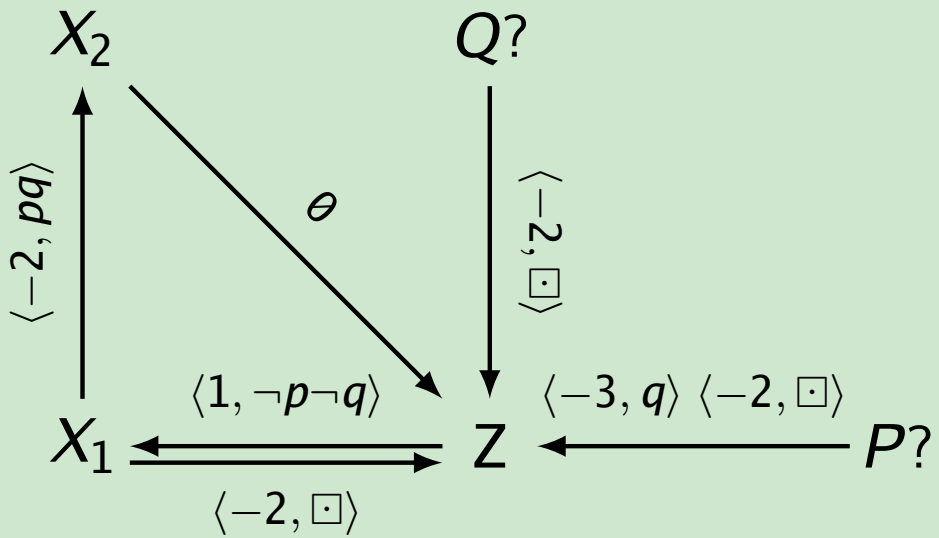
Conditional STNs
Example: Non–DC Instance

Example 2 (Non–DC Instance)

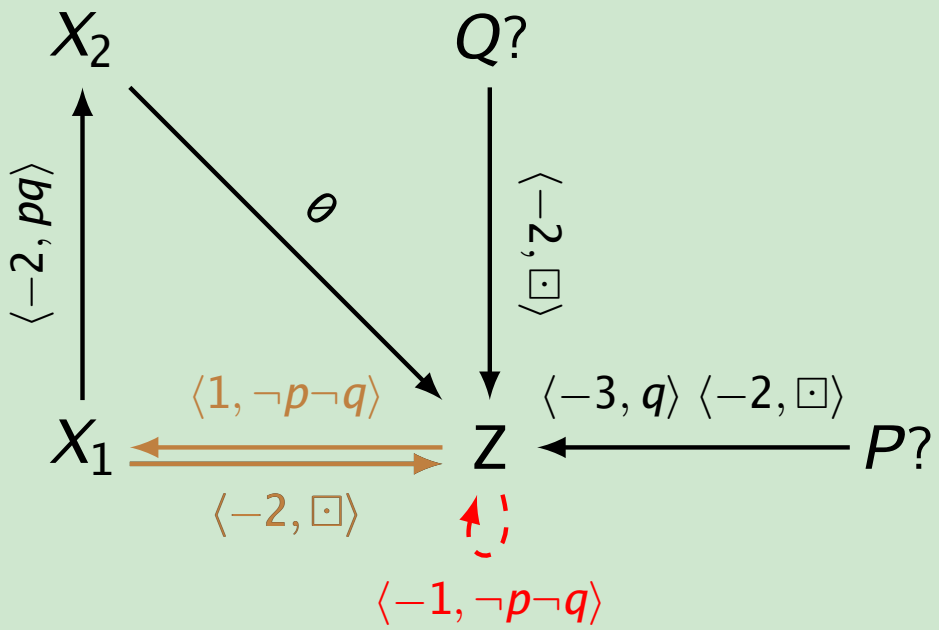


Conditional STNs
Example: Non–DC Instance

Example 2 (Non–DC Instance)



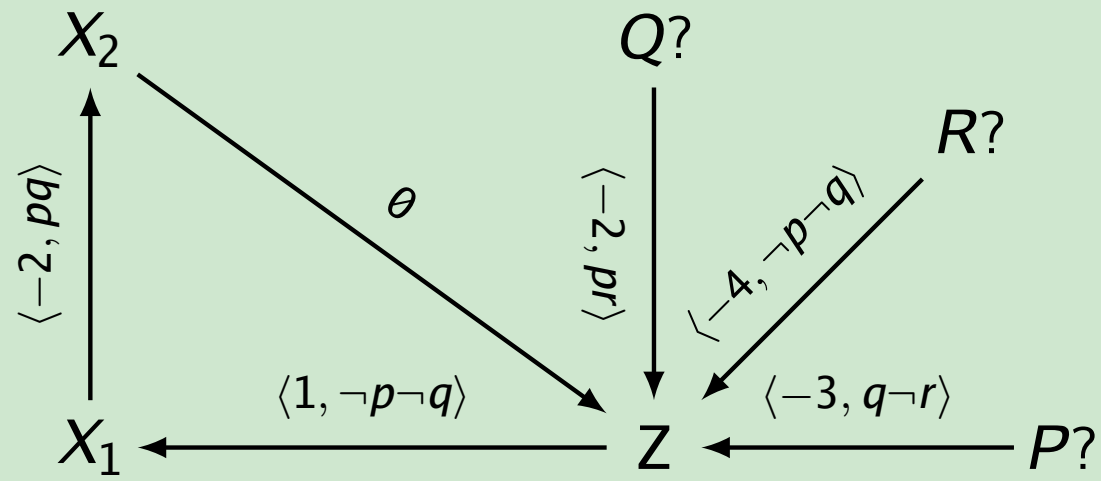
Example 2 (Non–DC Instance)



There is a scenario, $\neg p \neg q$, in which there exits a negative loop!
Therefore, the CSTN is not DC!

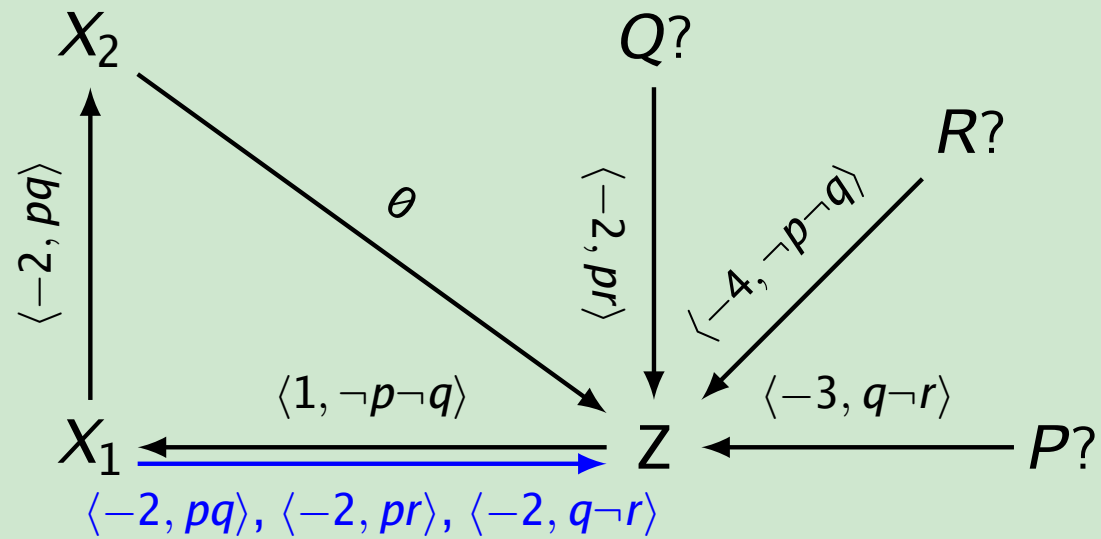
- Propagating along consistent labels is insufficient

Example 3 (Non–DC instance, but LP, R_0 and R_3^* not enough)



- Propagating along consistent labels is insufficient

Example 3 (Non–DC instance, but LP, R_0 and R_3^* not enough)

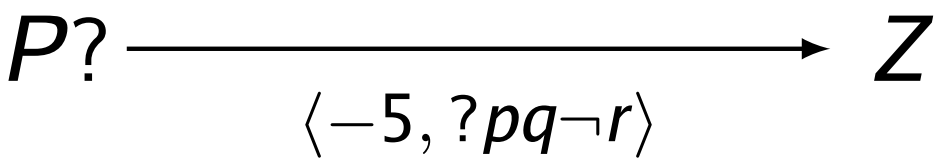


Conditional STNs
Propagating Q–Labels

- Propagating along consistent labels is insufficient
- *Q–labels*: contain literals such as $?p$.
- A constraint labeled by $?p$ must hold as long as p's value is unknown (i.e., as long as $P?$ remains unexecuted).
- Conjunction operation generalized to cover q–labels:
  $p \wedge \neg p \equiv ?p$; $p \wedge ?p \equiv ?p$; $\neg p \wedge ?p \equiv ?p$; etc.
- Q–labels only needed on *lower–bound* constraints*
  (i.e., edges pointing at Z).

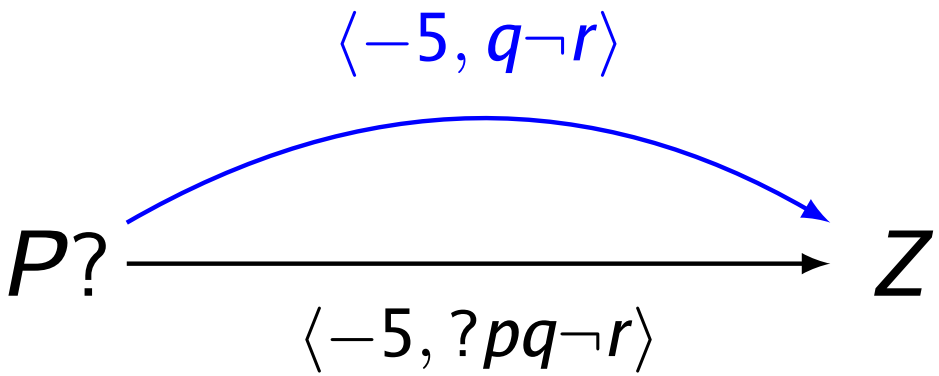
* [Hunsberger and Posenato, 2018b]

Conditional STNs
The qR_0 Rule



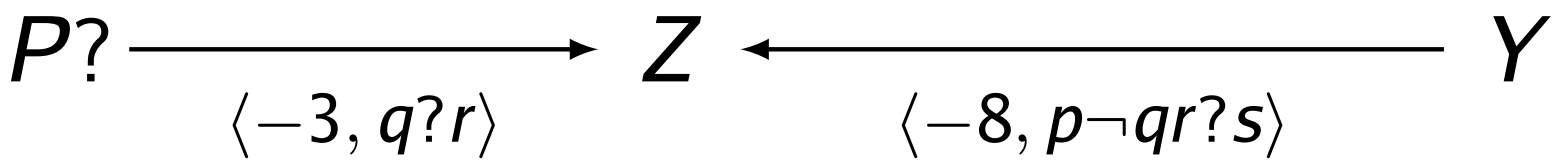
- Edge must terminate at Z;
- Edge weight must be negative;
- Any occurrence of p (or $\neg p$ or $?p$) removed from label.

Conditional STNs
The qR_0 Rule



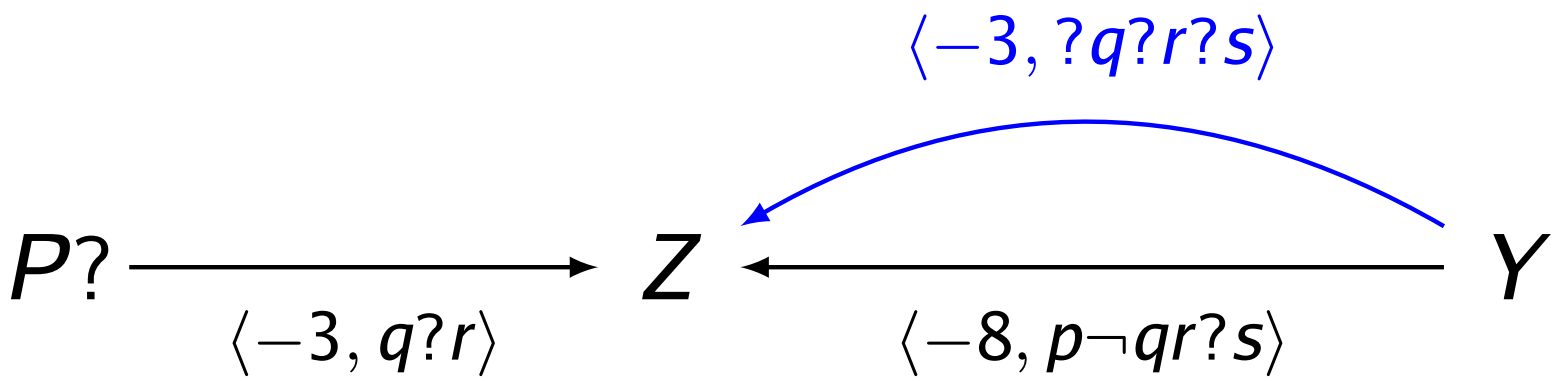
- Edge must terminate at Z;
- Edge weight must be negative;
- Any occurrence of p (or $\neg p$ or $?p$) removed from label.

Conditional STNs
The qR_3^* Rule



- Labels need not be consistent;
- Lefthand weight must be negative;
- Generated weight is max of pre–existing weights.

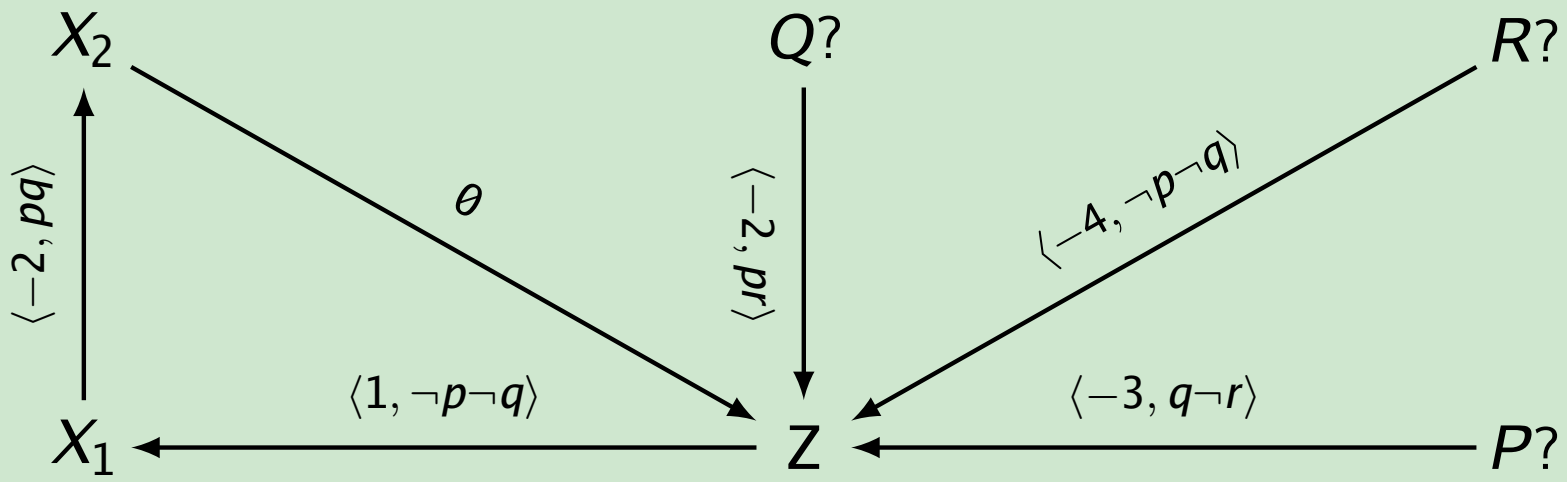
Conditional STNs
The qR_3^* Rule



- Labels need not be consistent;
- Lefthand weight must be negative;
- Generated weight is max of pre–existing weights.

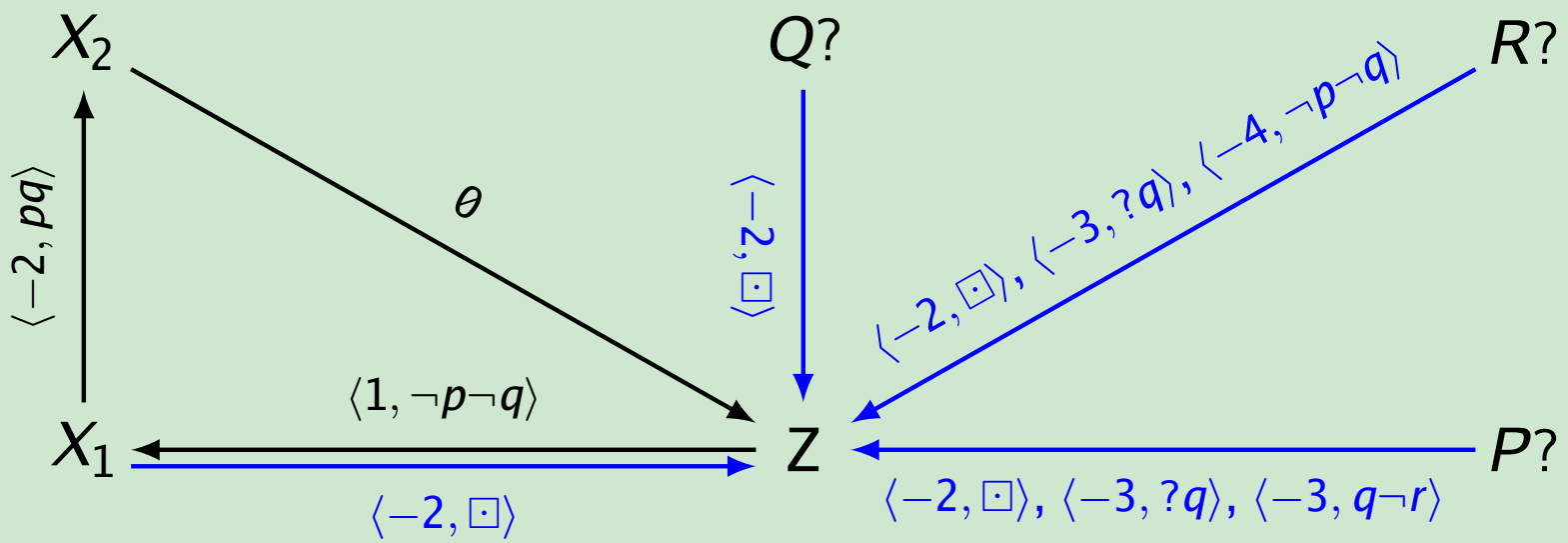
- Propagating along q–labels *is* sufficient!

Example 4 (Non–DC instance confirmed by rules LP, qR_0 and qR_3^*)



- Propagating along q–labels *is* sufficient!

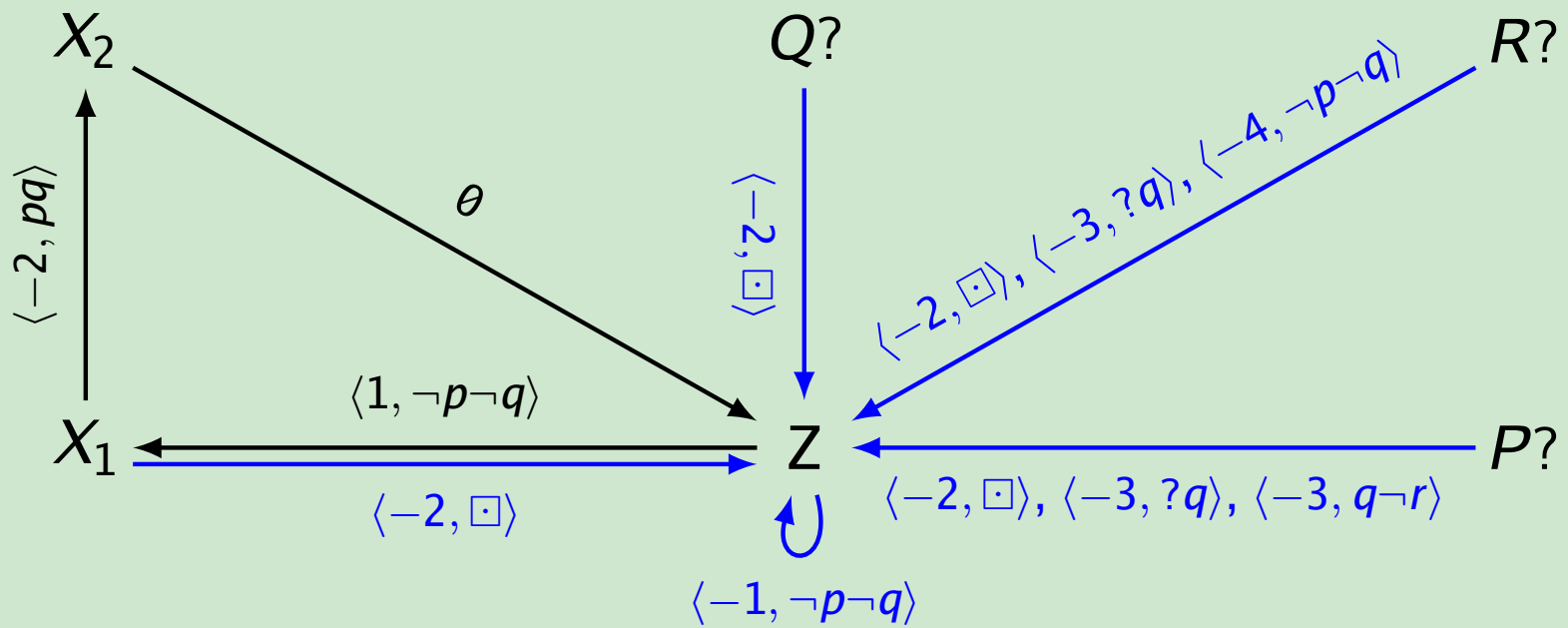
Example 4 (Non–DC instance confirmed by rules LP, qR_0 and qR_3^*)



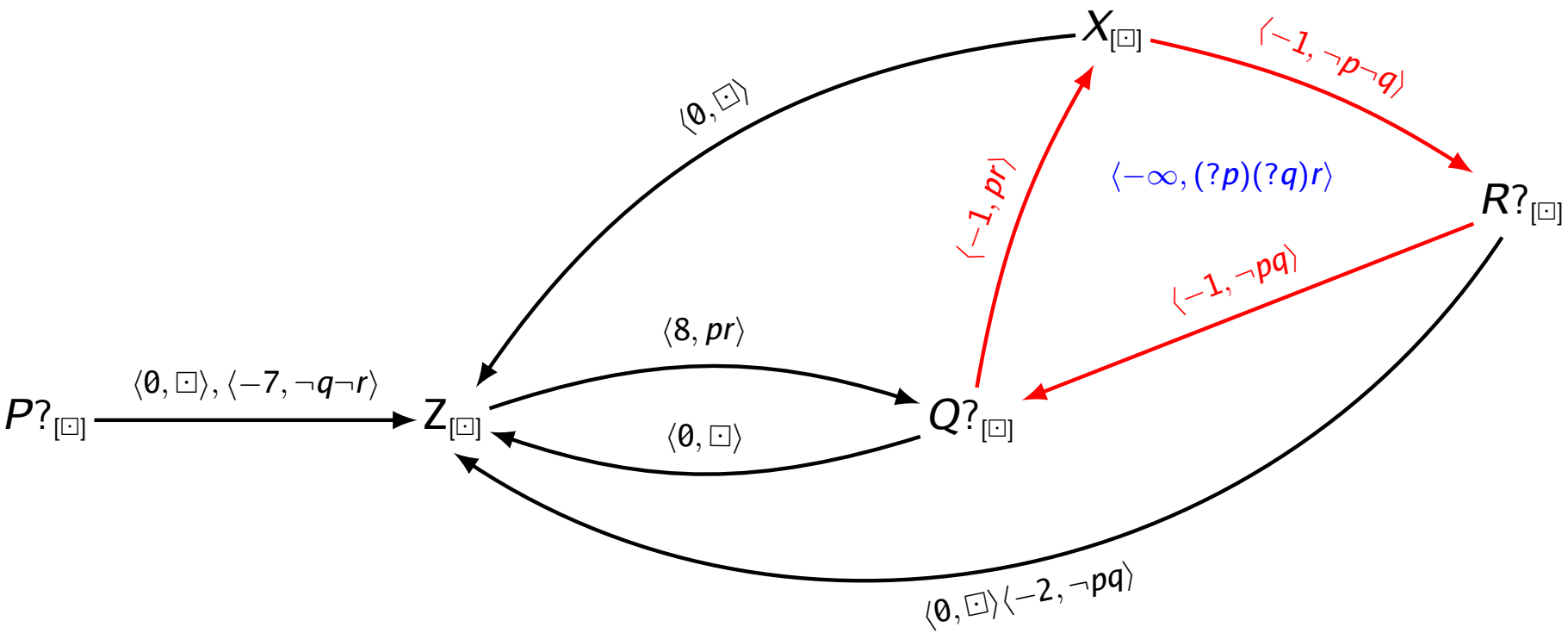
- Incidentally, the blue edges form a “negative q–star”.

- Propagating along q–labels *is* sufficient!

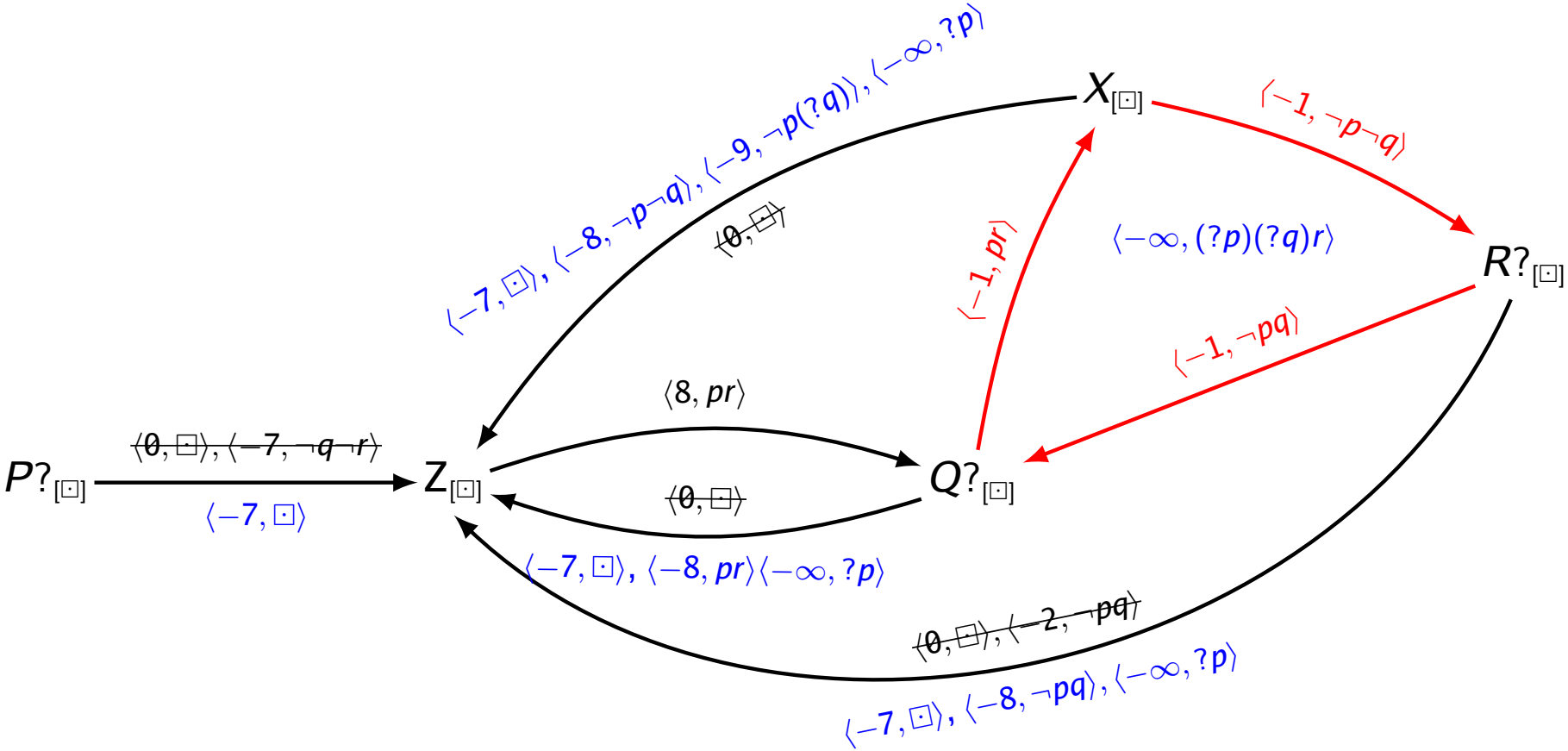
Example 4 (Non–DC instance confirmed by rules LP, qR_0 and qR_3^*)



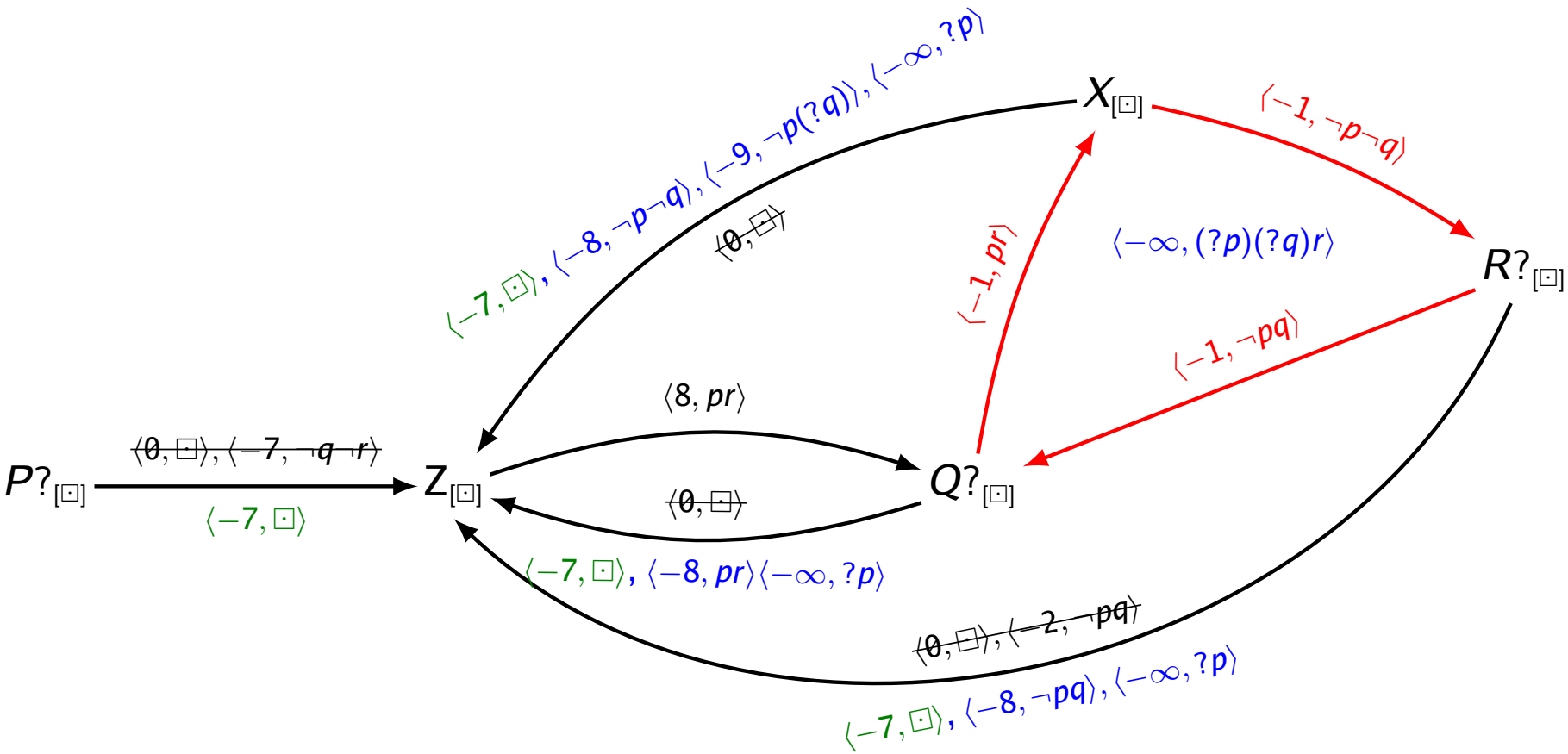
Conditional STNs
Negative Q–Loop Example



Conditional STNs
Negative Q–Loop Example



Conditional STNs
Negative Q–Loop Example



The *Spreading Lemma*

The minimum lower–bound constraint $\langle -7, \boxdot \rangle$
has spread to *all* unexecuted time–points. [Hunsberger et al., 2015]

Conditional STNs
DC–Checking Algorithm for CSTNs

- The DC–Checking Algorithm exhaustively propagates constraints using LP, qLP , and qR_3^*.
- Returns `NO` if any negative self–loop with a consistent label is ever found; otherwise returns `YES`.
- In positive cases, constructs *earliest–first* strategy, which is viable due to the Spreading Lemma.
- Although exponential–time in the worst case, shown to be practical across a variety of sample networks.

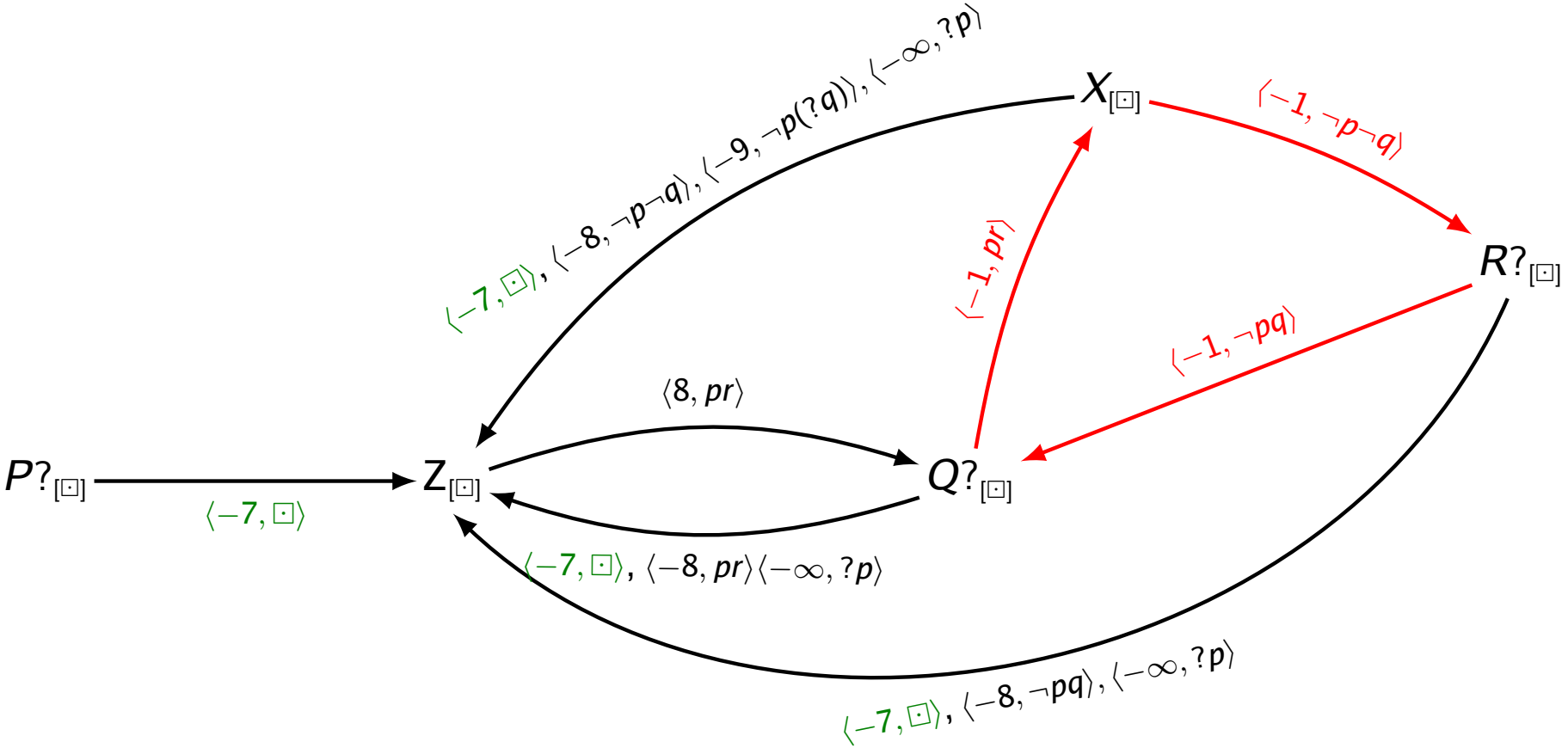
[Hunsberger and Posenato, 2018b; Hunsberger et al., 2015]

Conditional STNs
The *Earliest–First* Strategy

- Keep track of *current partial scenario* (CPS), π.
  Initially $\pi = \boxdot$.
- After each execution event, compute *effective lower bound* (ELB) for each as–yet–unexecuted time–point.
- $ELB(X, \pi)$ restricts attention to lower bounds for X whose labels are applicable to π.
- Next time–point to execute is the one with the min. *ELB* value.

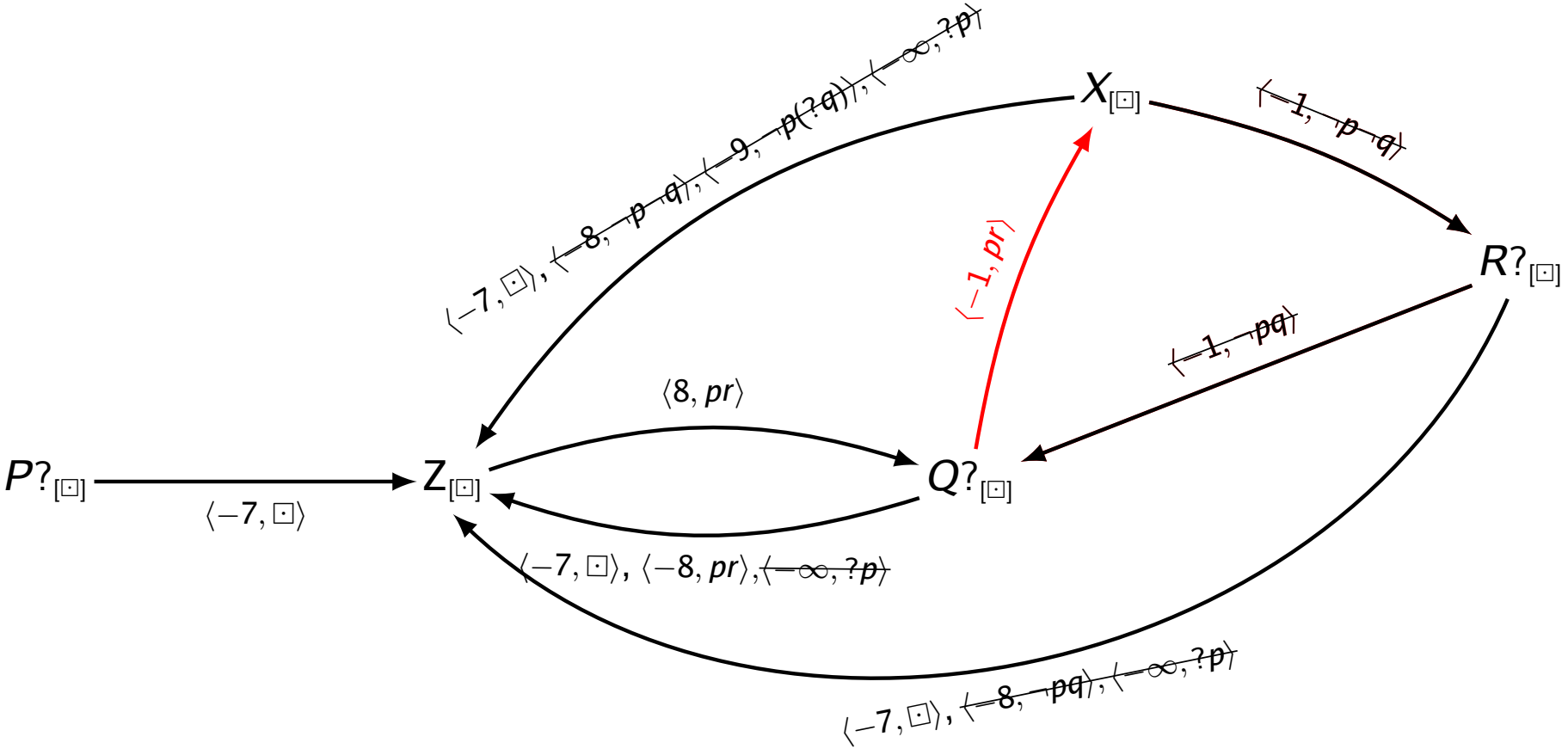
Conditional STNs
Sample Execution

$\pi = \boxdot, \quad Z = 0, \quad ELB(P?, \boxdot) = -7; \quad$ execute $P? = 7$.



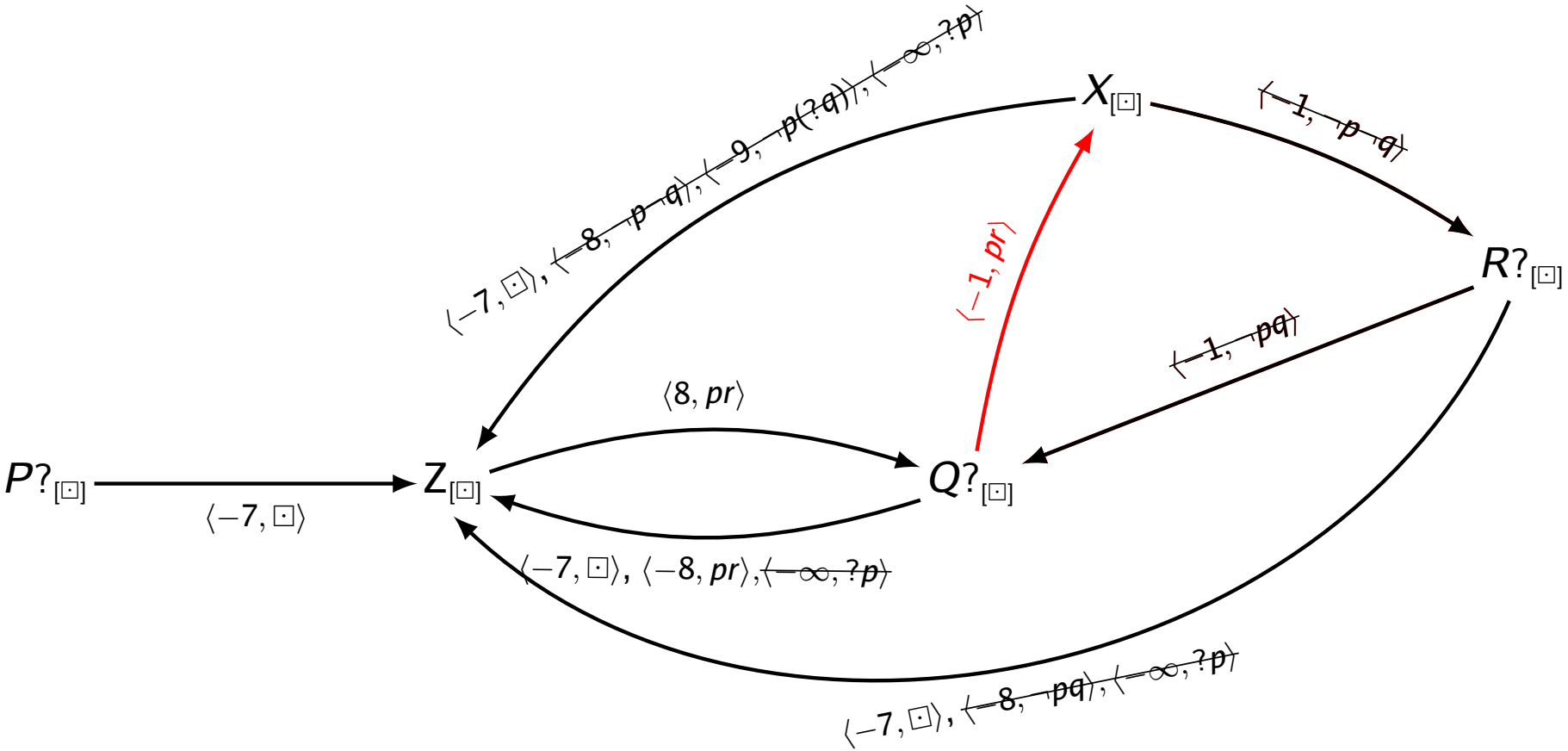
Conditional STNs
Sample Execution

Suppose $p = true$. $\pi = p$; $ELB(X, p) = 7 = ELB(R?, p)$.
So execute $X = 7$ and $R? = 7$.



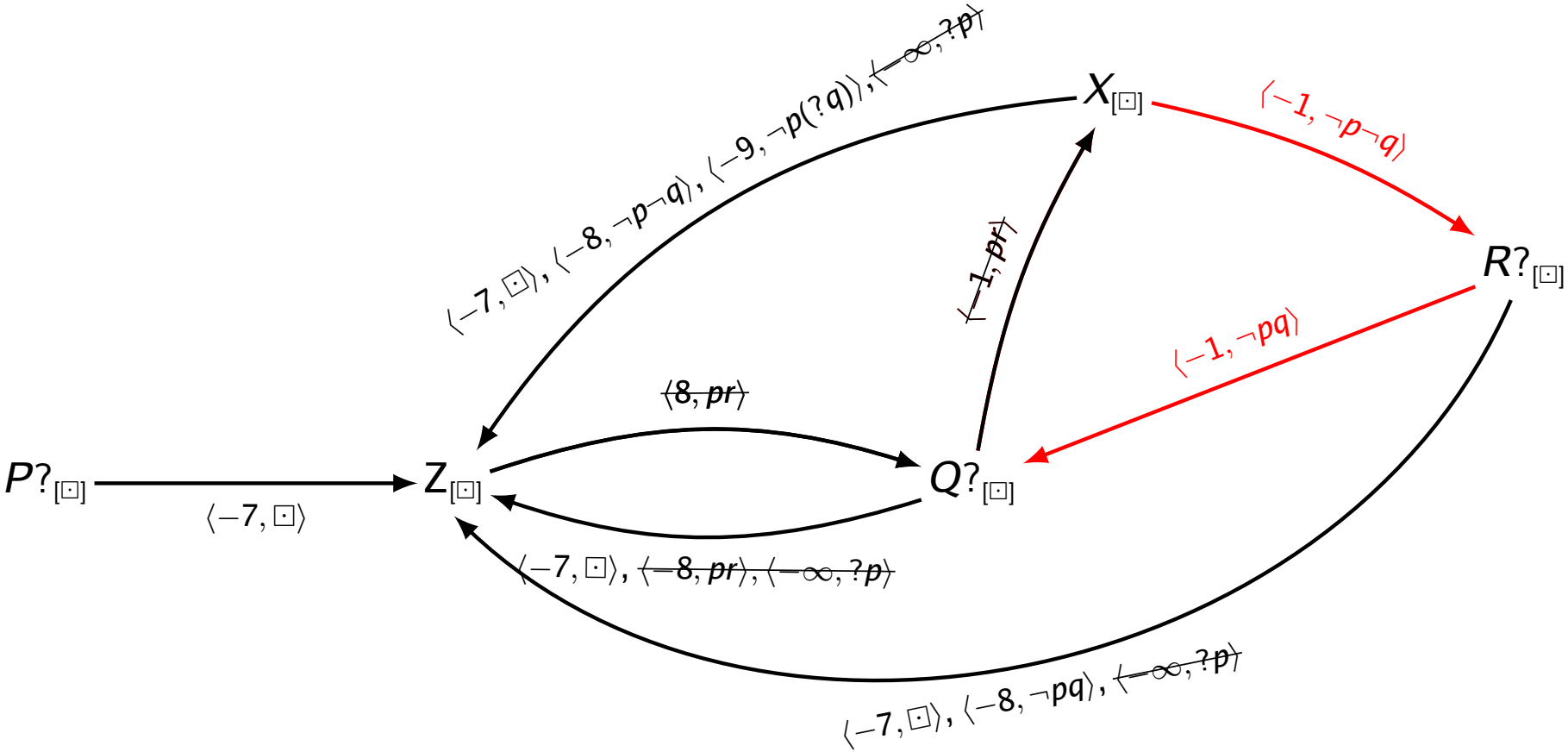
Conditional STNs
Sample Execution

Suppose $r = true$. $\pi = pr$; $ELB(Q?, p) = 8$.
So execute $Q? = 8$.



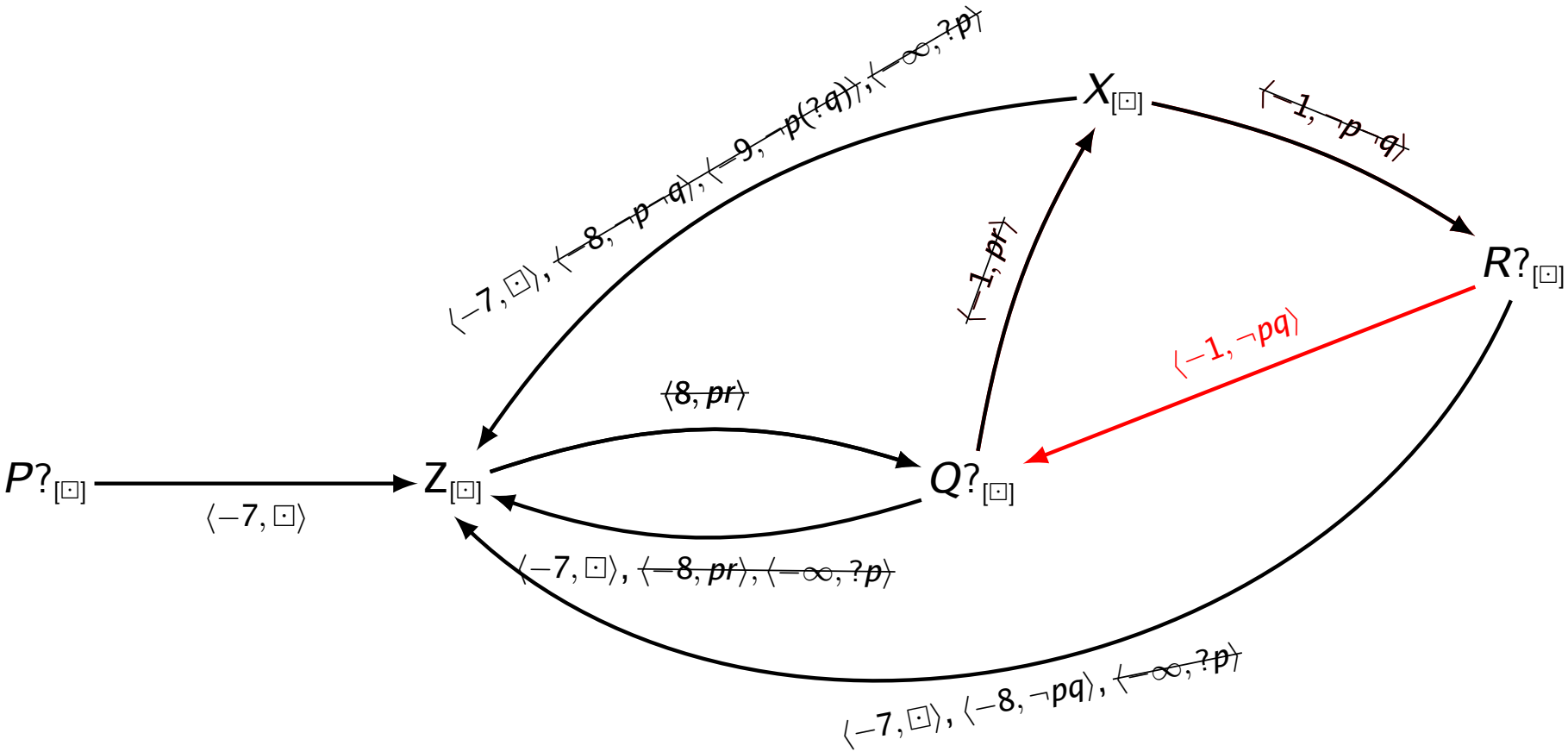
Conditional STNs
Alternative Execution

Suppose $p = \mathit{false}$. $\pi = \neg p$; $ELB(Q?, \neg p) = 7$
So execute $Q? = 7$.



Conditional STNs
Alternative Execution

Suppose $q = true$. $\pi = \neg pq$; $ELB(X, \neg pq) = 7$.
So execute $X = 7$. Afterward, execute $R? = 8$.



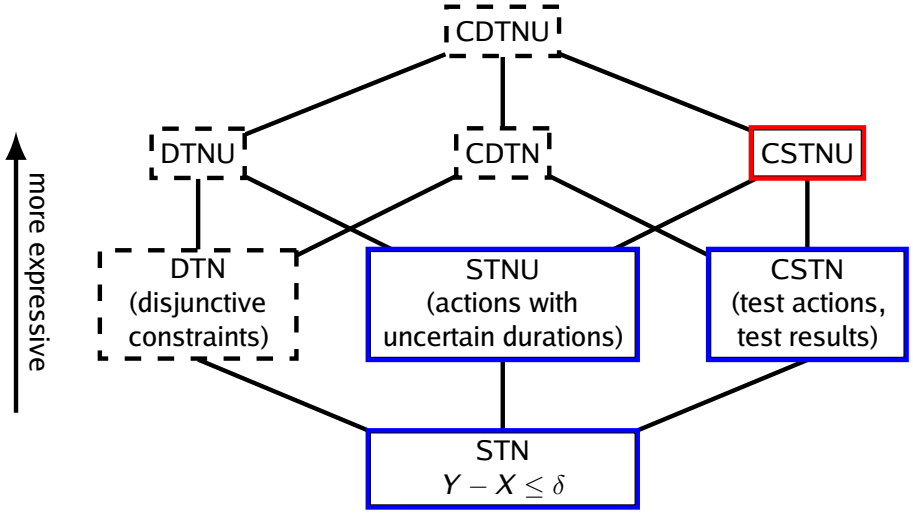
Conditional STNs
Bounded Reaction Time

- ϵ–dynamic consistency requires bounded reaction time $\epsilon > 0$ [Comin and Rizzi, 2015].
- Propagation–based ϵ–DC checking algorithm [Hunsberger and Posenato, 2016].
- Semantics of instantaneous reactivity for CSTNs [Cairo et al., 2016].

Conditional STNs
CSTN Summary

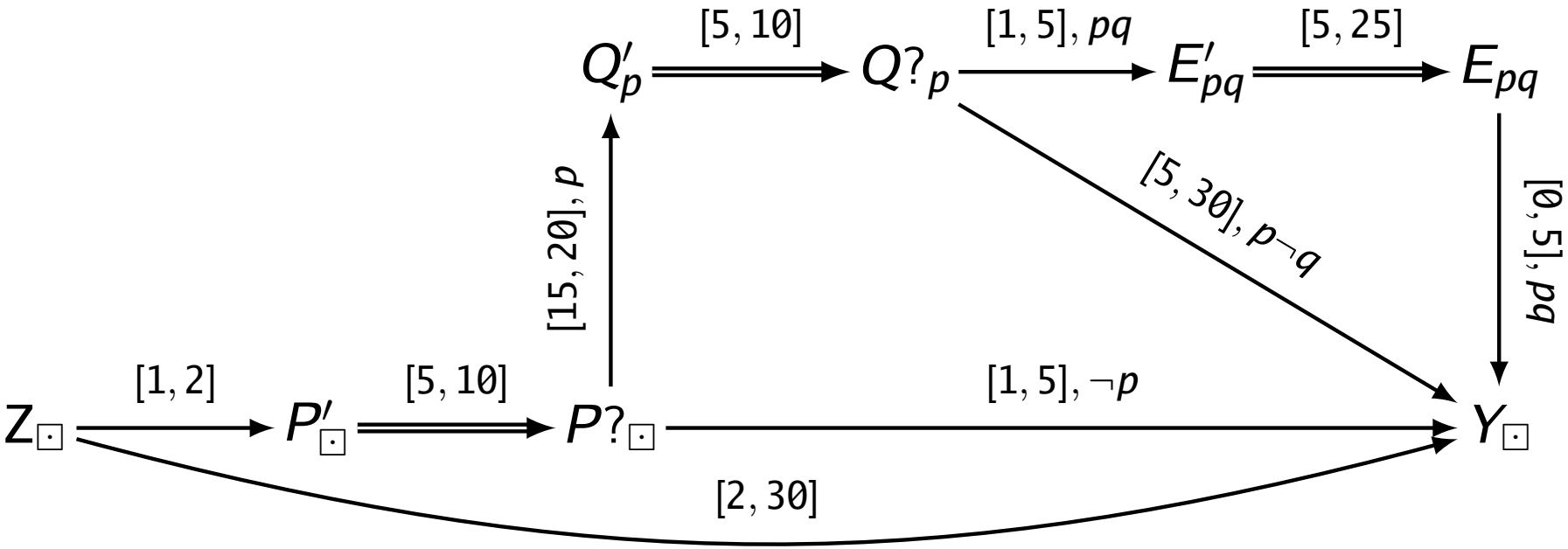
- Theory of dynamic consistency for CSTNs very solid:
  - instantaneous vs. non–instantaneous reactivity
  - bounded reaction time.
- Several proposed DC–checking algorithms: all exponential
  —but propagation–based algorithm shows promise.
- More work to do on flexible execution.

Conditional STNs with Uncertainty
CSTNUs



- A *Conditional Simple Temporal Network with Uncertainty* (CSTNU) combines contingent links from STNUs and observation time–points from CSTNs.

Conditional STNs with Uncertainty
Sample CSTNU



- Contingent links $(P', [5, 10], P?)$ and $(Q', [5, 10], Q?)$ represent tests for a patient.
- Contingent link $(E', [5, 25], E)$ represents the emergency therapy.
- Contingent links have no propositional labels,
  but the labels on their endpoints must be the same.

Conditional STNs with Uncertainty (CSTNUs)
Dynamic Controllability

- Dynamic Execution Strategy: execution decisions may react to observations and contingent durations.
- A CSTNU is *dynamically controllable* if there exists a dynamic execution strategy that guarantees that all *relevant* constraints will be satisfied no matter which scenario is incrementally revealed over time, and no matter how the contingent durations turn out.

Conditional STNs with Uncertainty (CSTNUs)
DC–Checking for CSTNUs

- Convert to Timed Game Automaton [Cimatti et al., 2014]
- Propagate labeled constraints [Hunsberger and Posenato, 2018a]

Conditional STNs with Uncertainty (CSTNUs)
DC Checking via Propagation

- Propagate *labeled* constraints as done for CSTN
- Propagate also upper–case and lower–case (contingent) edges as done for STNU considering labeled constraints

Conditional STNs with Uncertainty (CSTNUs)
DC Checking via Propagation

- Propagate *labeled* constraints as done for CSTN
- Propagate also upper–case and lower–case (contingent) edges as done for STNU considering labeled constraints

The mixing of these two kind of propagations requires extending the STNU–concept of upper–case values!

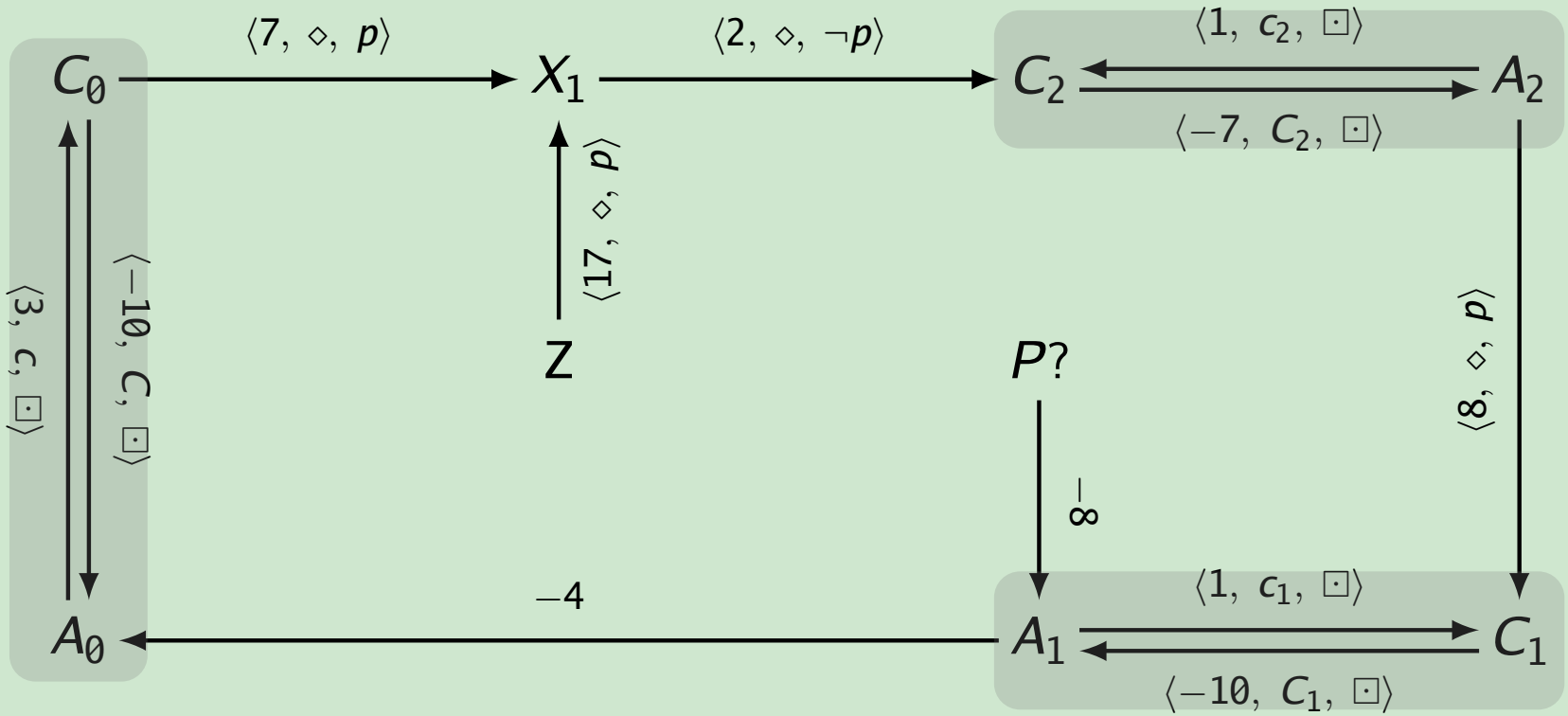
Generalizing Upper–Case Labels

- Given contingent time–points $C_1, C_2, \ldots, C_k$, their names are called Upper Case (UC) alphabetic–letters (a–letters).
- An UC alphabetic label (a–label) is a set of a–letters:
  - is empty, notated as $\diamond$; or
  - contains *one or more* UC a–letters, notated as $C_{i_1} \ldots C_{i_m}$.
- For any UC a–labels $\aleph, \aleph'$, their *conjunction* is given by their union (i.e., $\aleph\aleph' = \aleph \cup \aleph'$).

Generalizing labeled values

- Each edge is annotated by a triple, called a *labeled value*
- A *labeled value* is a triple, $\langle \delta,\ \aleph,\ \alpha \rangle$, where:
  - $\delta \in \mathbb{R}$
  - $\aleph$ is an a–label
  - α is a propositional label (from CSTN)

Example 5 (A CSTNU represented using labeled values)



Edge value v is a shorthand for $\langle v, \diamond, \boxdot \rangle$

Conditional STNs with Uncertainty (CSTNUs)
Propagation Rules for CSTNUs

Forward Upper Case Propagation:	z!
Labeled Extended Propagation:	zLP/Nc/Uc
Cross Case and Lower Case Propagation:	zLc/Cc
Label Removal:	zLR
Label Modification:	zqR_0
Label "Spreading":	zqR_3^*

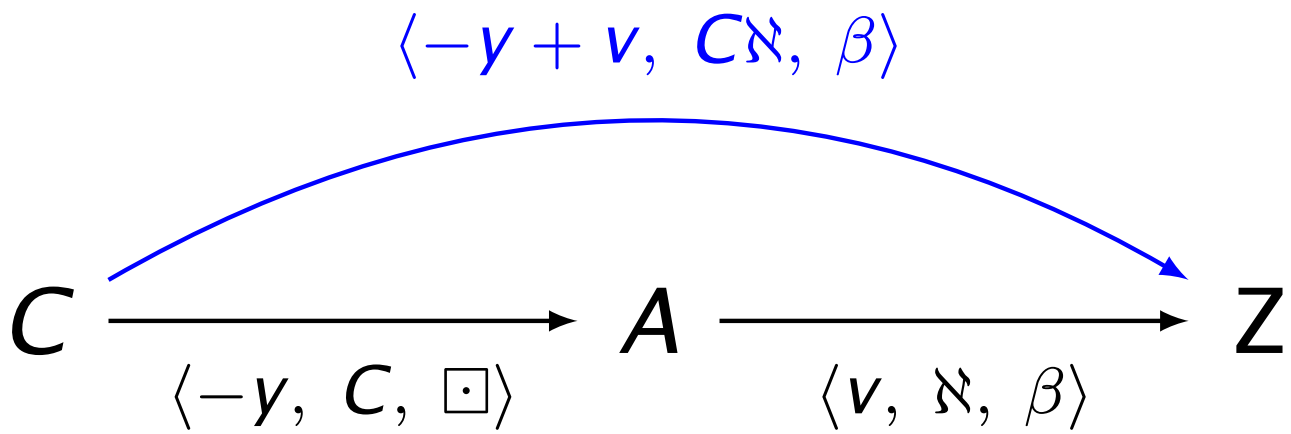
The *z*! rule can generate edges with multiple UC letters.

Conditions:

- $-y + v < 0$
- β does not contain unknown literals.

Conditional STNs with Uncertainty (CSTNUs)
The z! rule

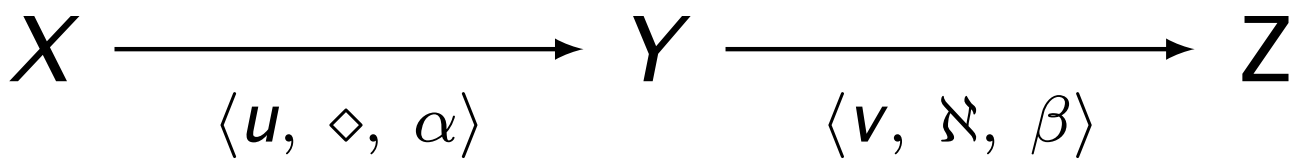
The *z*! rule can generate edges with multiple UC letters.



Conditions:

- $-y + v < 0$
- β does not contain unknown literals.

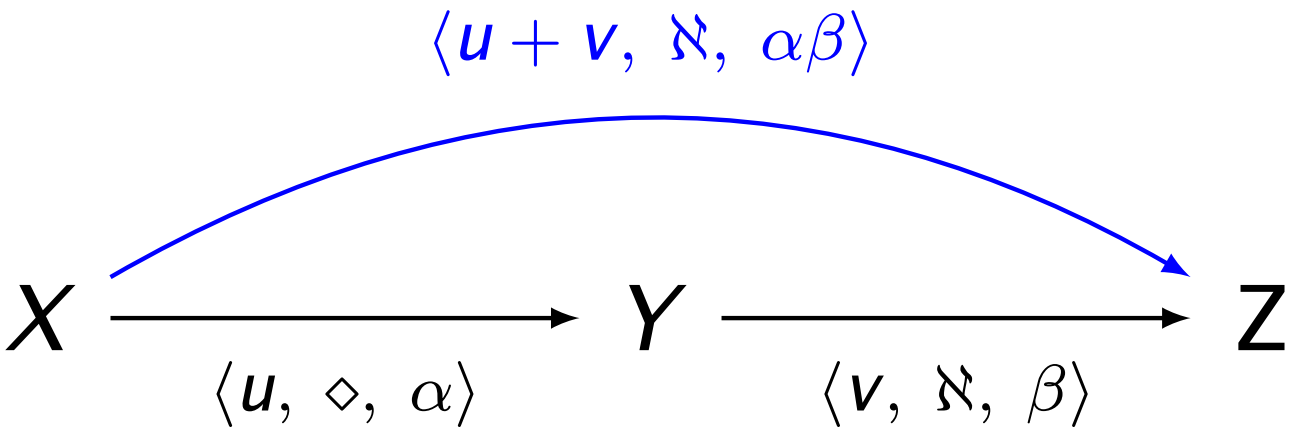
Conditional STNs with Uncertainty (CSTNUs)
The zLP/Nc/Uc Rule



Conditions:

- $u + v < 0$
- α and β must be consistent.

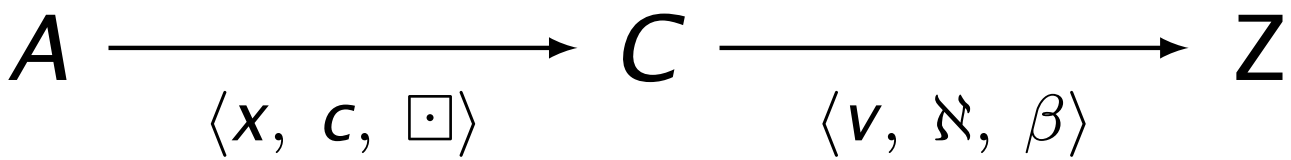
Conditional STNs with Uncertainty (CSTNUs)
The zLP/Nc/Uc Rule



Conditions:

- $u + v < 0$
- α and β must be consistent.

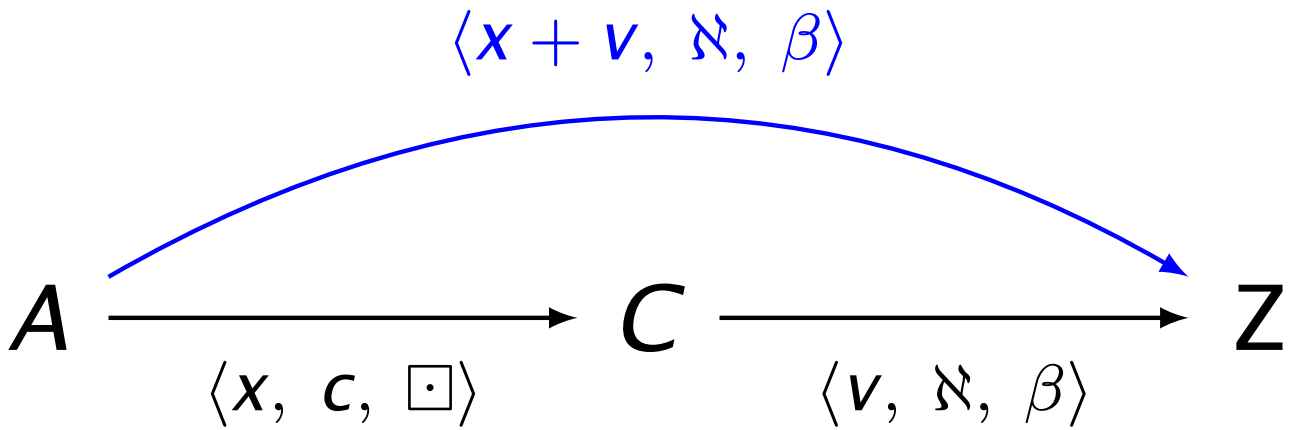
Conditional STNs with Uncertainty (CSTNUs)
The zLc/Cc rule



Conditions:

- $x + v < 0$
- $C \notin \aleph$
- β does not contain unknown literals.

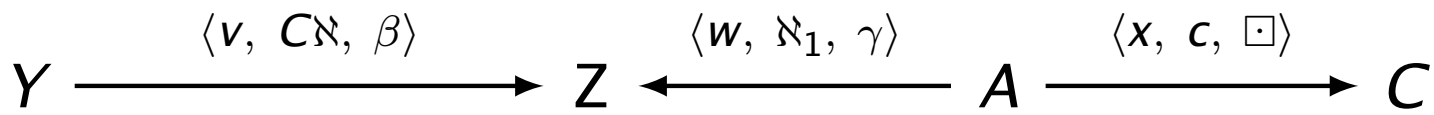
Conditional STNs with Uncertainty (CSTNUs)
The zLc/Cc rule



Conditions:

- $x + v < 0$
- $C \notin \aleph$
- β does not contain unknown literals.

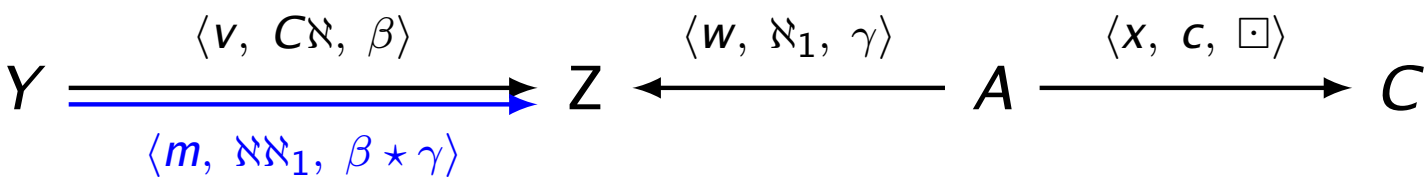
Conditional STNs with Uncertainty (CSTNUs)
The zLR rule



Conditions:

- $C \notin \aleph\aleph_1$
- β, γ can contain unknown literals.

Conditional STNs with Uncertainty (CSTNUs)
The zLR rule

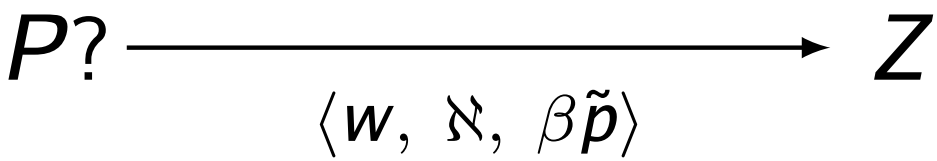


where $m = \max\{v, w - x\}$.

Conditions:

- $C \notin \aleph\aleph_1$
- β, γ can contain unknown literals.

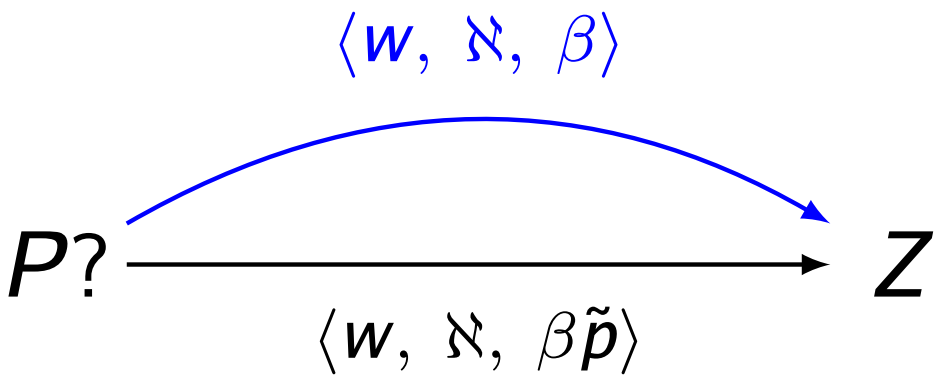
Conditional STNs with Uncertainty (CSTNUs)
The zqR$_\emptyset$ rule



Conditions:

- $w < 0$
- β can contain unknown literals
- $\tilde{p} \in \{p, \neg p, ?p\}$

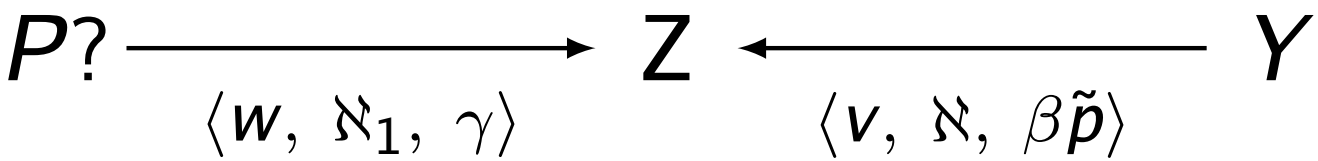
Conditional STNs with Uncertainty (CSTNUs)
The zqR$_0$ rule



Conditions:

- $w < 0$
- β can contain unknown literals
- $\tilde{p} \in \{p, \neg p, ?p\}$

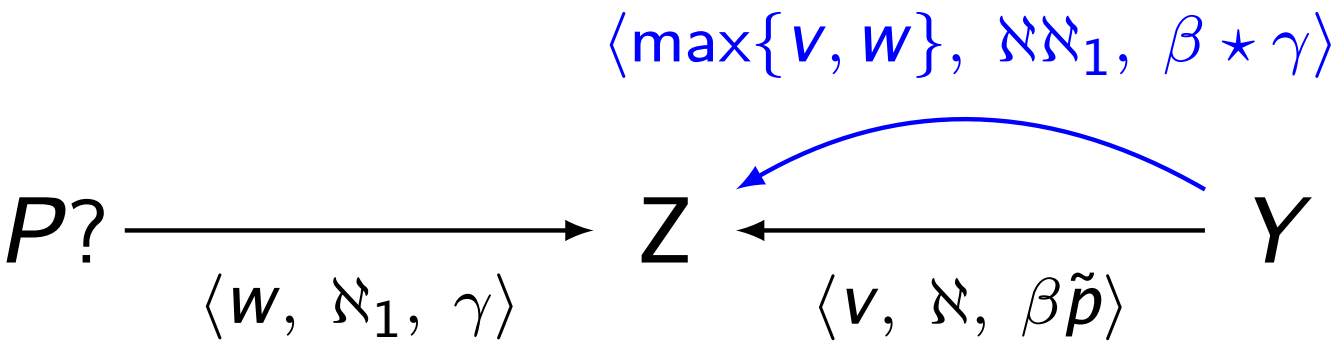
Conditional STNs
The zqR$_3^*$ rule



Conditions:

- β, γ can contain unknown literals
- $\tilde{p} \in \{p, \neg p, ?p\}$

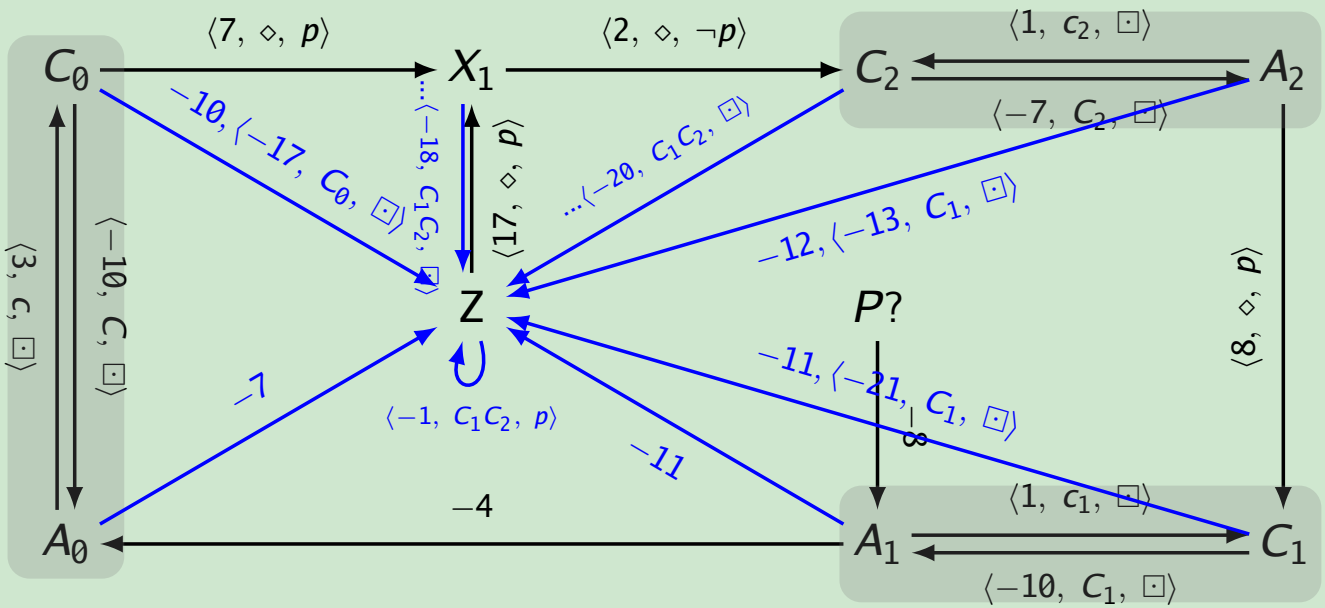
Conditional STNs
The zqR$_3^*$ rule



Conditions:

- β, γ can contain unknown literals
- $\tilde{p} \in \{p, \neg p, ?p\}$

Example 6 (A snapshot of CSTNU DC–checking algorithm)



- This CSTNU is not DC: if C_0 occurs at its minimum, while C_1 and C_2 at their maximum, then X cannot be set satisfying all constraints.
- After 123 propagations, the CSTNU contains an explicit negative loop at Z.
- The blue constraints are some of those determined by the algorithm before the negative loop is discovered.

Conditional STNs with Uncertainty (CSTNUs)
DC–Checking Algorithm for CSTNUs

- The DC–Checking Algorithm does exhaustive propagation
- Returns `NO` if any negative loop with a consistent label is ever found; otherwise, returns `YES`.
- In positive cases, constructs *earliest–first* strategy, which is viable due to the spreading lemma for CSTNUs.
- The algorithm has exponential–time complexity in the worst case.
- Currently we are working on some rule optimizations for making it as practical for a variety of sample networks as the CSTN DC–checking algorithm.

Conditional STNs with Uncertainty (CSTNUs)
CSTNU Summary

- Theory of dynamic controllability for CSTNUs has a solid foundation.
- Two competing DC–checking algorithms*, both exponential.
- Propagation–based algorithms show promise, but require further investigation.
- Alternatives to earliest–first strategy?

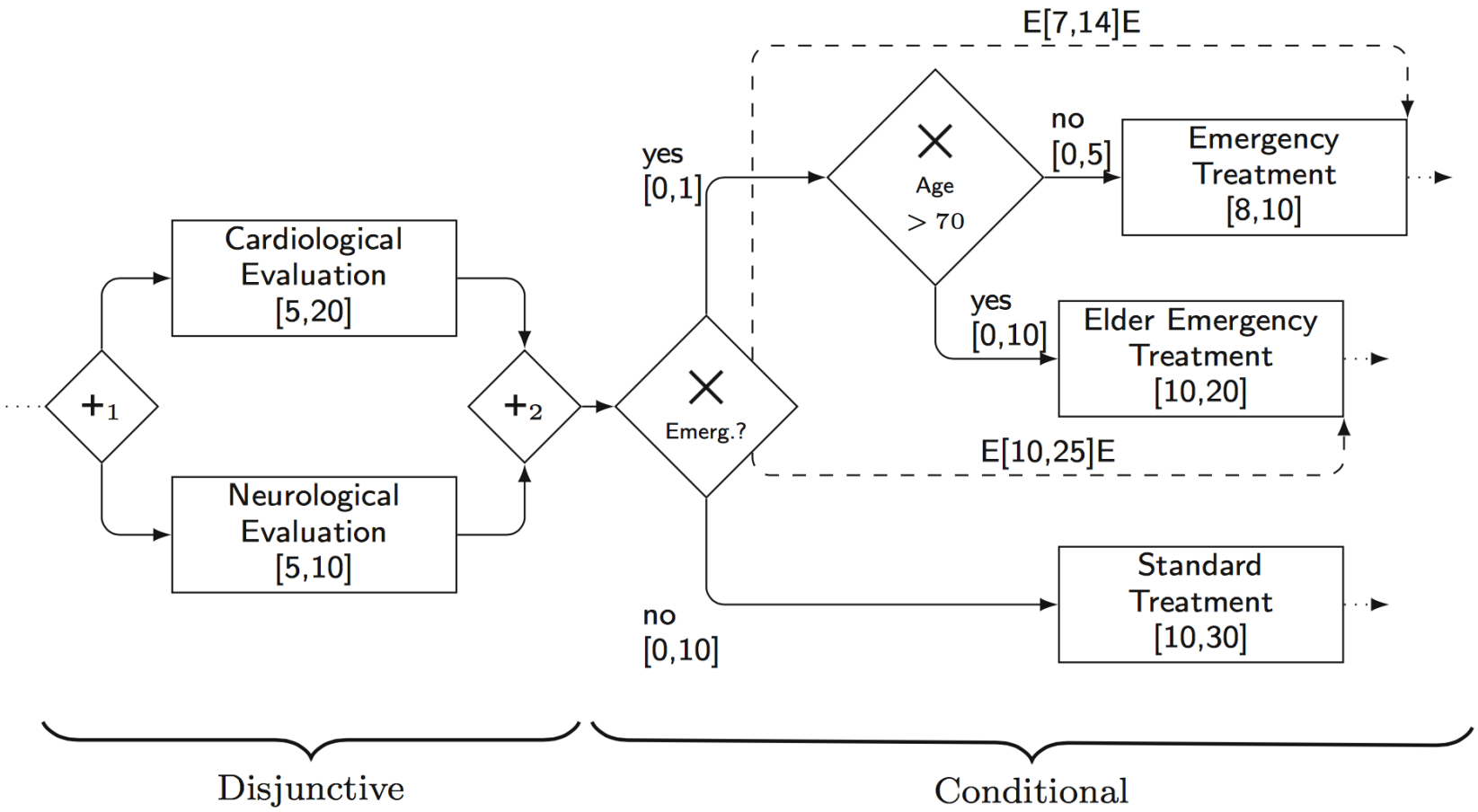
* [Hunsberger and Posenato, 2018a]

CDTNUs
Adding Disjunction to CSTNUs

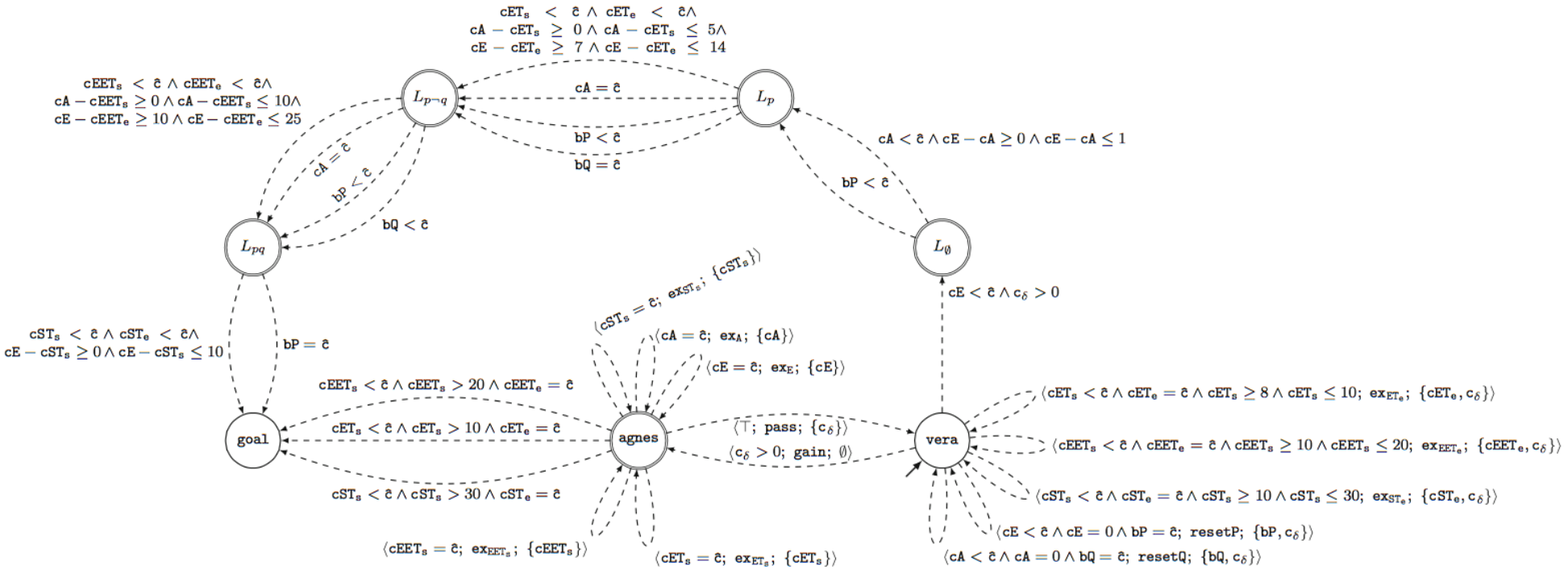
- A *Conditional Disjunctive Temporal Network with Uncertainty* (CDTNU) augments a CSTNU to include disjunctive constraints.
- Possible to convert the DC–checking problem for CDTNUs into a *controller–synthesis* problem for *Timed Game Automata* (TGAs)*.
- Highlights connections between temporal networks and TGAs, but algorithm not yet practical.

* [Cimatti et al., 2016]

CDTNUs
Sample Workflow



CDTNUs
TGA Encoding of Workflow



Conclusions

- Theoretical foundations for a variety of temporal networks are quite solid.
- STNs have been incorporated into planning and scheduling applications for over a decade.
- $O(N^3)$–time DC–checking/dispatchability algorithms for STNUs makes them ready for prime time.
- Propagation–based algorithms for CSTNs and CSTNUs show promise.

References I

Massimo Cairo, Carlo Comin, and Romeo Rizzi. Instantaneous reaction–time in dynamic–consistency checking of conditional simple temporal networks. In 23rd Int. Symp. on Temporal Representation and Reasoning (TIME 2016), pages 80–89, 2016. doi: 10.1109/TIME.2016.16.

Massimo Cairo, Luke Hunsberger, and Romeo Rizzi. Faster dynamic controllablity checking for simple temporal networks with uncertainty. In 25th Int. Symp. on Temporal Representation and Reasoning (TIME 2018), volume 120 of LIPIcs, pages 8:1–8:16, 2018. doi: 10.4230/LIPIcs.TIME.2018.8.

Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In Andrea Cesta, Carlo Combi, and Francois Laroussinie, editors, 21st Int. Symp. on Temporal Representation and Reasoning (TIME 2014), pages 27–36, 2014. doi: 10.1109/TIME.2014.21.

Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via timed game automata. Acta Informatica, 53(6–8):681–722, 2016. ISSN 1432–0525. doi: 10.1007/s00236-016-0257-2.

Carlo Comin and Romeo Rizzi. Dynamic consistency of conditional simple temporal networks via mean payoff games: a singly–exponential time dc–checking. In 22st Int. Symp. on Temporal Representation and Reasoning (TIME 2015), pages 19–28, 2015. doi: 10.1109/TIME.2015.18.

References II

Patrick R. Conrad and Brian C. Williams. Drake: An efficient executive for temporal plans with choice. Journal of Artificial Intelligence Research (JAIR), 42:607–659, 2011. URL http://dx.doi.org/10.1613/jair.3478.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. The MIT Press, 2001.

Rina Dechter, Itay Meiri, and J. Pearl. Temporal Constraint Networks. Artificial Intelligence, 49(1–3):61–95, 1991. doi: 10.1016/0004-3702(91)90006-6.

Shimon Even and Hillel Gazit. Updating Distances in Dynamic Graphs. Methods of Operations Research, 49:371–387, 1985.

Luke Hunsberger. Group Decision Making and Temporal Reasoning. PhD thesis, Harvard University, 2002. Available as Harvard Technical Report TR–05–02.

Luke Hunsberger. Fixing the semantics for dynamic controllability and providing a more practical characterization of dynamic execution strategies. In TIME 2009, pages 155–162, 2009. doi: 10.1109/TIME.2009.25.

Luke Hunsberger. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In Nicolas Markey and Jef Wijsen, editors, Seventeenth Int. Symp. on Temporal Representation and Reasoning (TIME–2010), pages 121–128, 2010. doi: 10.1109/TIME.2010.16.

References III

Luke Hunsberger. Magic Loops in Simple Temporal Networks with Uncertainty–Exploiting Structure to Speed Up Dynamic Controllability Checking. In 5th Int. Conf. on Agents and Artificial Intelligence, ICAART 2013, volume 2, pages 157–170, 2013.

Luke Hunsberger. Magic Loops and the Dynamic Controllability of Simple Temporal Networks with Uncertainty. In Joaquim Filipe and Ana Fred, editors, Agents and Artificial Intelligence, volume 449 of Communications in Computer and Information Science (CCIS), pages 332–350. Springer, 2014.

Luke Hunsberger. Efficient execution of dynamically controllable simple temporal networks with uncertainty. Acta Informatica, 53(2):89–147, 2015. doi: 10.1007/s00236-015-0227-0.

Luke Hunsberger and Barbara J. Grosz. A combinatorial auction for collaborative planning. In 4th Int. Conf. on Multi–Agent Systems (ICMAS–2000), 2000.

Luke Hunsberger and Roberto Posenato. Checking the dynamic consistency of conditional temporal networks with bounded reaction times. In Amanda Jane Coles, Andrew Coles, Stefan Edelkamp, Daniele Magazzeni, and Scott Sanner, editors, 26th Int. Conf. on Automated Planning and Scheduling, ICAPS 2016, pages 175–183, 2016. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13108.

Luke Hunsberger and Roberto Posenato. Sound–and–Complete Algorithms for Checking the Dynamic Controllability of Conditional Simple Temporal Networks with Uncertainty. In 25th Int. Symp. on Temporal Representation and Reasoning (TIME 2018), volume 120 of LIPIcs, pages 14:1–14:17, 2018a. doi: 10.4230/LIPIcs.TIME.2018.14.

References IV

Luke Hunsberger and Roberto Posenato. Simpler and faster algorithm for checking the dynamic consistency of conditional simple temporal networks. In 26th Int. Joint Conf. on Artificial Intelligence, (IJCAI–2018), pages 1324–1330, 2018b. doi: 10.24963/ijcai.2018/184.

Luke Hunsberger, Roberto Posenato, and Carlo Combi. A sound–and–complete propagation–based algorithm for checking the dynamic consistency of conditional simple temporal networks. In Fabio Grandi, Martin Lange, and Alessio Lomuscio, editors, 22nd Int. Symp. on Temporal Representation and Reasoning (TIME 2015), pages 4–18, 2015. doi: 10.1109/TIME.2015.26.

James C. Boerkoel Jr. and Edmund H. Durfee. Decoupling the multiagent disjunctive temporal problem. In 27th AAAI Conf. on Artificial Intelligence, 2013.

Paul Morris. A Structural Characterization of Temporal Dynamic Controllability. In Principles and Practice of Constraint Programming (CP 2006), volume 4204, pages 375–389, 2006. doi: 10.1007/11889205_28.

Paul Morris. Dynamic controllability and dispatchability relationships. In Integration of AI and OR Techniques in Constraint Programming, volume 8451 of LNCS, pages 464–479. 2014. doi: 10.1007/978-3-319-07046-9_33.

Paul Morris. The mathematics of dispatchability revisited. In 26th Int. Conf. on Automated Planning and Scheduling (ICAPS–2016), pages 244–252, 2016.

References V

Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In 20th National Conf. on Artificial Intelligence (AAAI–05), pages 1193–1198, 2005.

Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In 17th Int. Joint Conf. on Artificial Intelligence (IJCAI–01), pages 494–502, 2001.

Kiriakos Simon Mountakis, Tomas Klos, and Cees Witteveen. Dynamic temporal decoupling. In 14th Int. Conf. Integration of AI and OR Techniques in Constraint Programming, volume 10335 of Lecture Notes in Computer Science (LNCS), pages 328–343, 2017. doi: 10.1007/978-3-319-59776-8_27.

Nicola Muscettola, Paul H. Morris, and Ioannis Tsamardinos. Reformulating Temporal Plans for Efficient Execution. In 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR–98), pages 444–452, 1998.

Léon Planken. Incrementally Solving the STP by Enforcing Partial Path Consistency. In 27th PlanSIG Work., pages 87–94, 2008. URL https://pdfs.semanticscholar.org/fb41/21c05cc8a55301d385598f5c24d64cf703a4.pdf.

G. Ramalingam and Thomas Reps. On the Computational Complexity of Dynamic Graph Problems. Theoretical Computer Science, 158:233–277, 1996.

G. Ramalingam, J. Song, L. Joskowicz, and R. E. Miller. Solving Systems of Difference Constraints Incrementally. Algorithmica, 23(3):261–275, 1999. ISSN 1432–0541. doi: 10.1007/PL00009261.

References VI

Hans Rohnert. A Dynamization of the All Pairs Least Cost Path Problem. In Kurt Mehlhorn, editor, 2nd Symp. of Theoretical Aspects of Computer Science (STACS 85), volume 182 of Lecture Notes in Computer Science (LNCS), pages 279–286. Springer, 1985. doi: 10.1007/BFb0024016.

Ioannis Tsamardinos, Nicola Muscettola, and Paul Morris. Fast Transformation of Temporal Plans for Efficient Execution. In 15th National Conf. on Artificial Intelligence (AAAI–98), pages 254–261, 1998.

Ioannis Tsamardinos, Thierry Vidal, and Martha E. Pollack. CTP: A new constraint–based formalism for conditional, temporal planning. Constraints, 8:365–388, 2003. doi: 10.1023/A:1025894003623.

Lin Xu and Berthe Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. In 10th Int. Symp. on Temporal Representation and Reasoning and 4th Int. Conf. on Temporal Logic (TIME–ICTL–2003), pages 210–220, 2003. doi: 10.1109/TIME.2003.1214898.