Faster Dynamic–Consistency Checking for Conditional Simple Temporal Networks

Luke Hunsberger[1] Roberto Posenato[2]

[1]Department of Computer Science, Vassar College, Poughkeepsie, NY, USA

[2]Department of Computer Science, University of Verona, Italy

ICAPS 2020

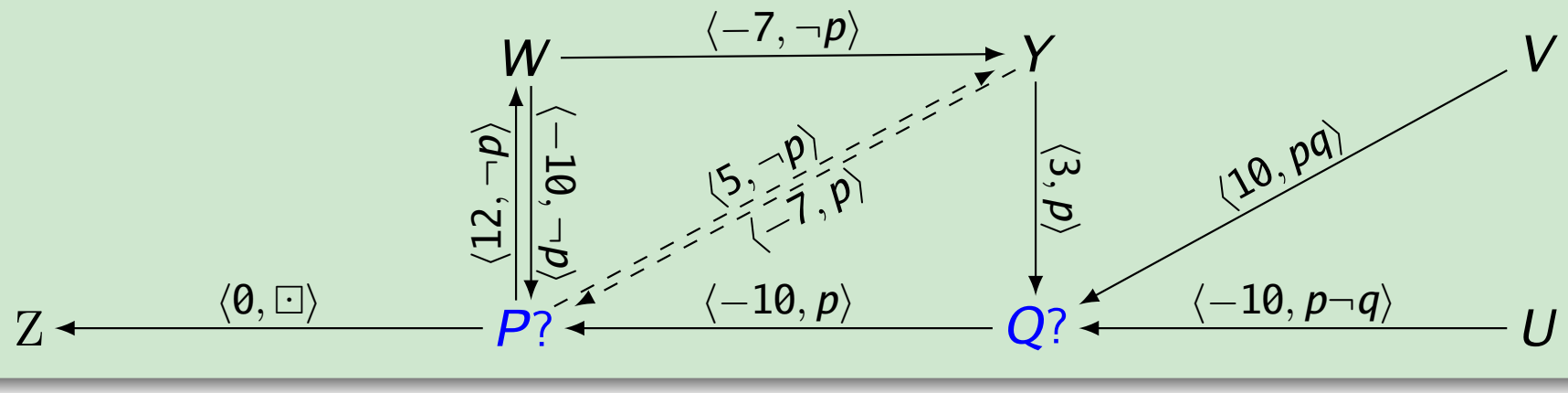
Conditional Simple Temporal Networks (CSTNs)

Tsamardinos et al. (2003) [6], Hunsberger et al. (2015) [5]

A CSTN is like a Simple Temporal Network (STN), except that:

- Some time–points are observation time–points (OTPs), and
- Constraints can be labeled by conjunctions of propositional literals.

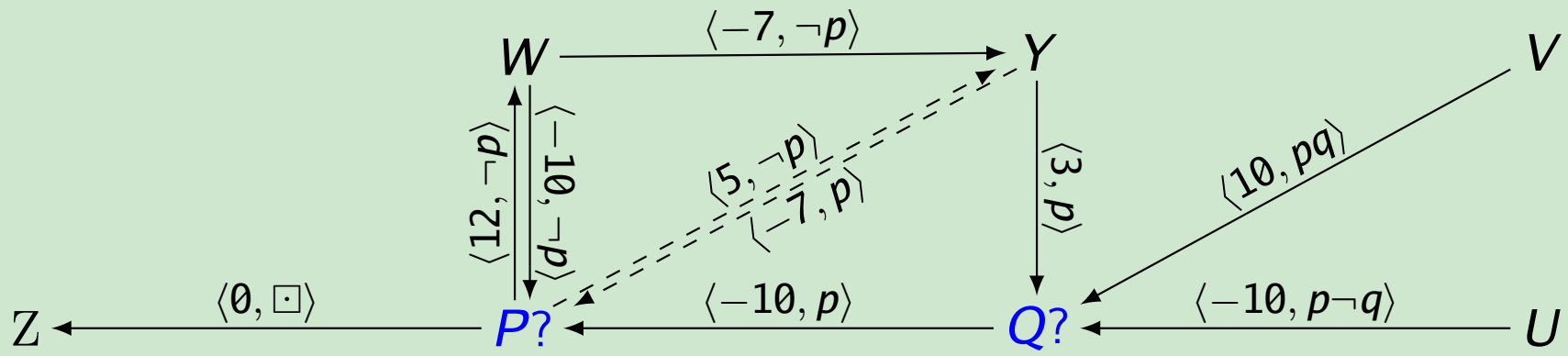
Example



CSTNs-continued

- Each OTP $P?$ has a corresponding propositional letter p. Executing $P?$ generates a truth value for p.
- As OTPs are executed and truth values incrementally revealed, a complete *scenario* is eventually determined.
- A constraint labeled by, say, $p(\neg q)$ need only hold in scenarios where p is *true*, and q is *false*.

Example



Dynamic Consistency of CSTNs

- A CSTN is *dynamically consistent* (DC) if:
  - there exists a *strategy* for executing its time–points ...
  - such that all *relevant* constraints will be satisfied ...
  - no matter which scenario is eventually revealed.
- The π–DC semantics of Cairo et al. (2017) [1] allows execution strategies that can react instantaneously to observations.
- Existing π–DC–checking algorithms
  - The $\mathtt{HP}_{18}$ Algorithm (Hunsberger & Posenato, 2018) [3]
  - The $\mathtt{HP}_{19}$ Algorithm (Hunsberger & Posenato, 2019) [4]

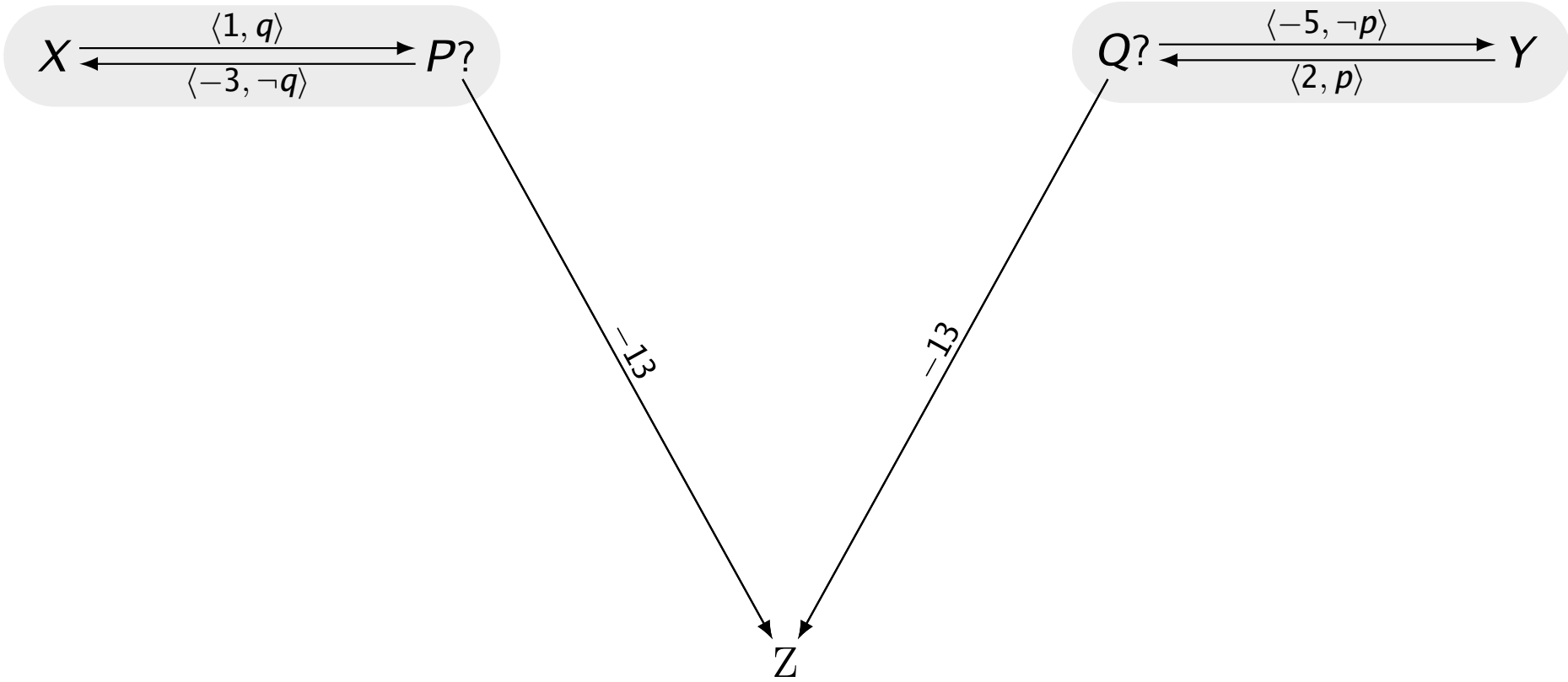
The $\mathtt{HP}_{18}$ Algorithm

Hunsberger & Posenato (2018) [3]

- Only generates constraints involving Z (the zero time–point)
- Propagation can generate *q–labeled* constraints (e.g., $(?p)q(\neg r)$).
- A constraint labeled by $(?p)q(\neg r)$ must hold as long as p is not yet known, q is true or not yet known, and r is false or not yet known.
- The $\star$ operator generalizes conjunction to q–labeled constraints. *Example:* $(p(\neg q)r(?t)) \star ((\neg p)q(\neg s)) = (?p)(?q)r(\neg s)(?t)$
- The $\mathtt{HP}_{18}$ algorithm is sound and complete, but can get bogged down cycling through *negative q–loops*.

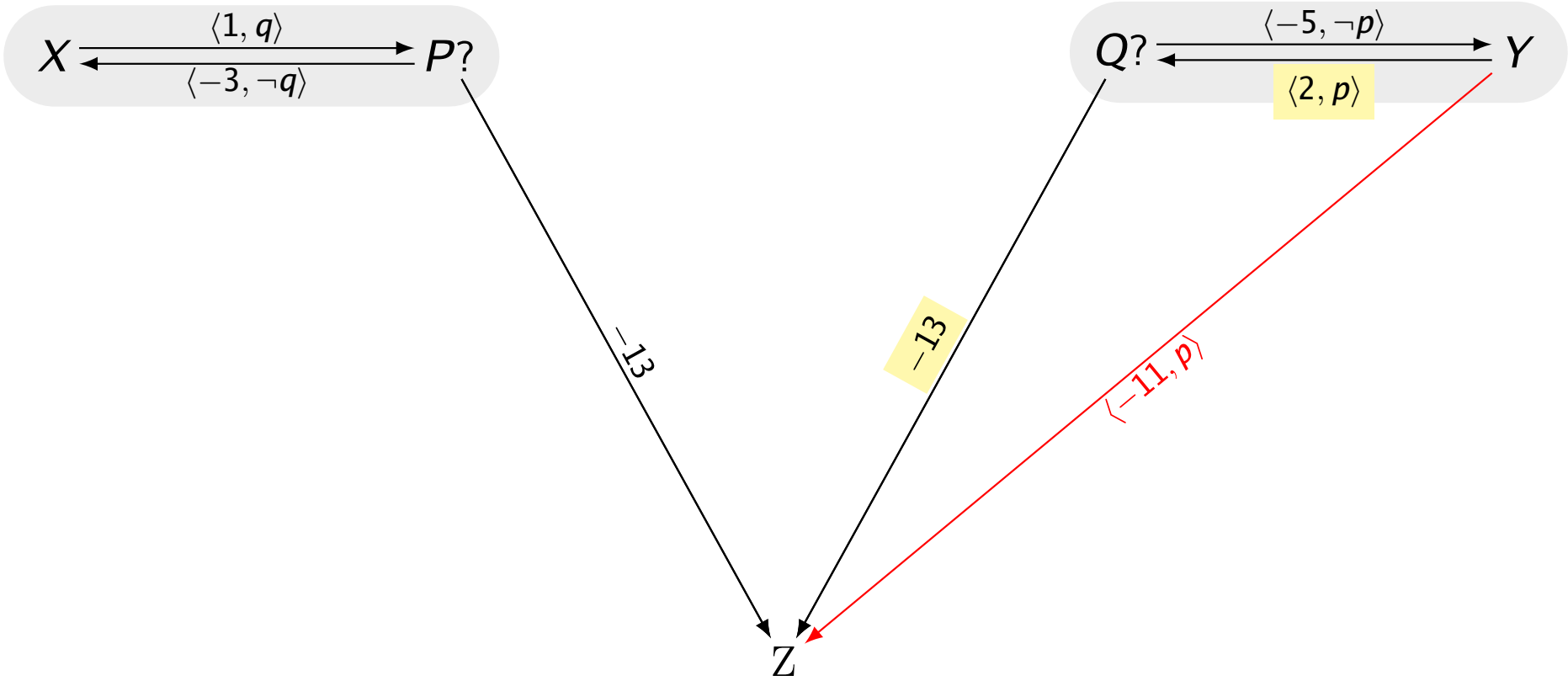
The $\mathtt{HP}_{18}$ Algorithm

Simulation



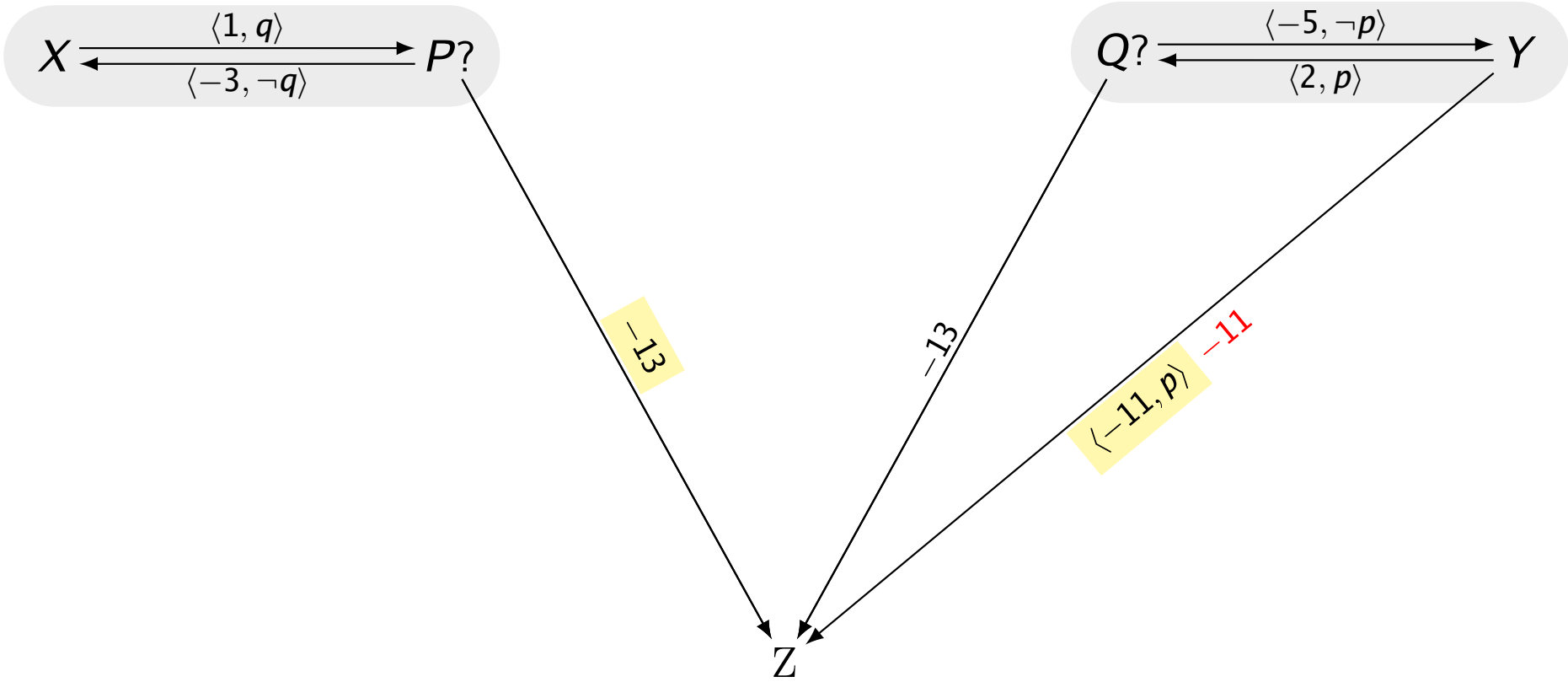
The HP_{18} Algorithm

Simulation



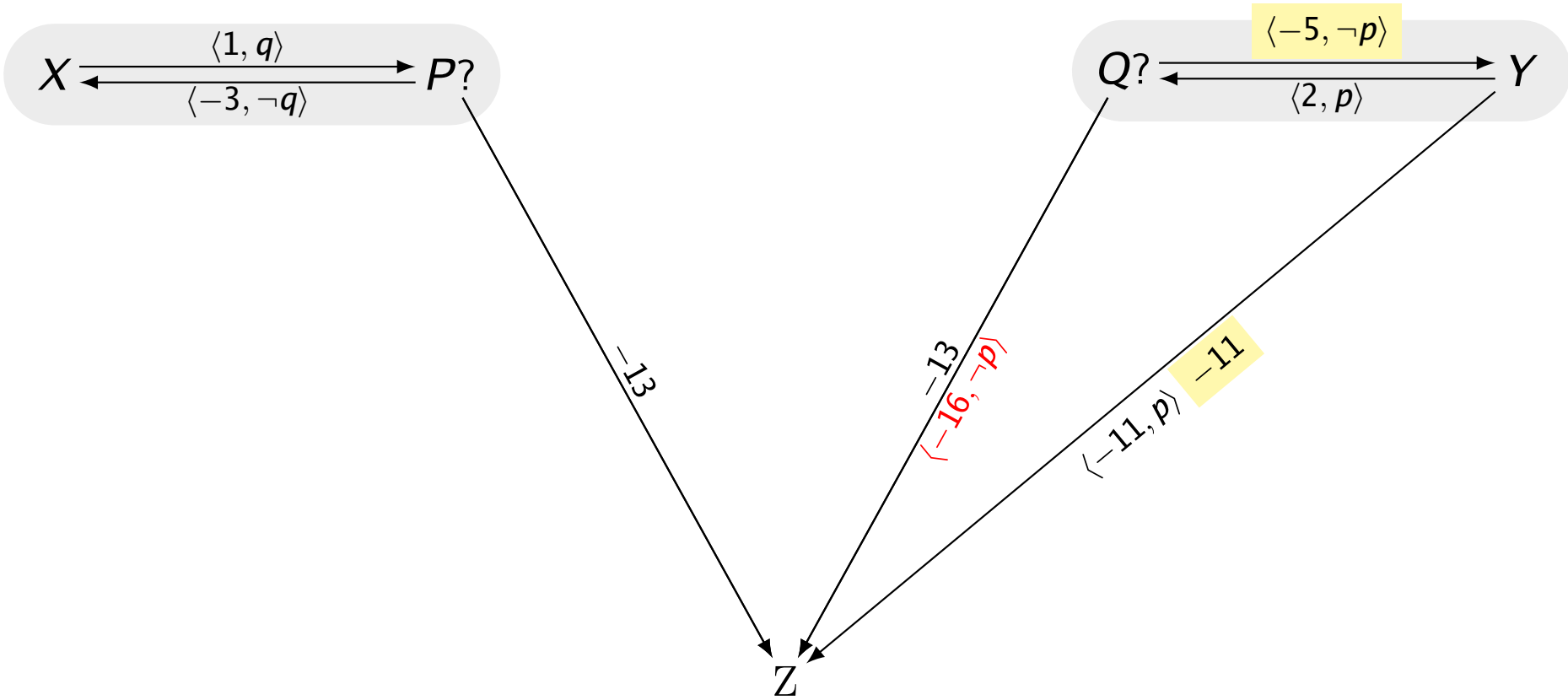
The HP_{18} Algorithm

Simulation



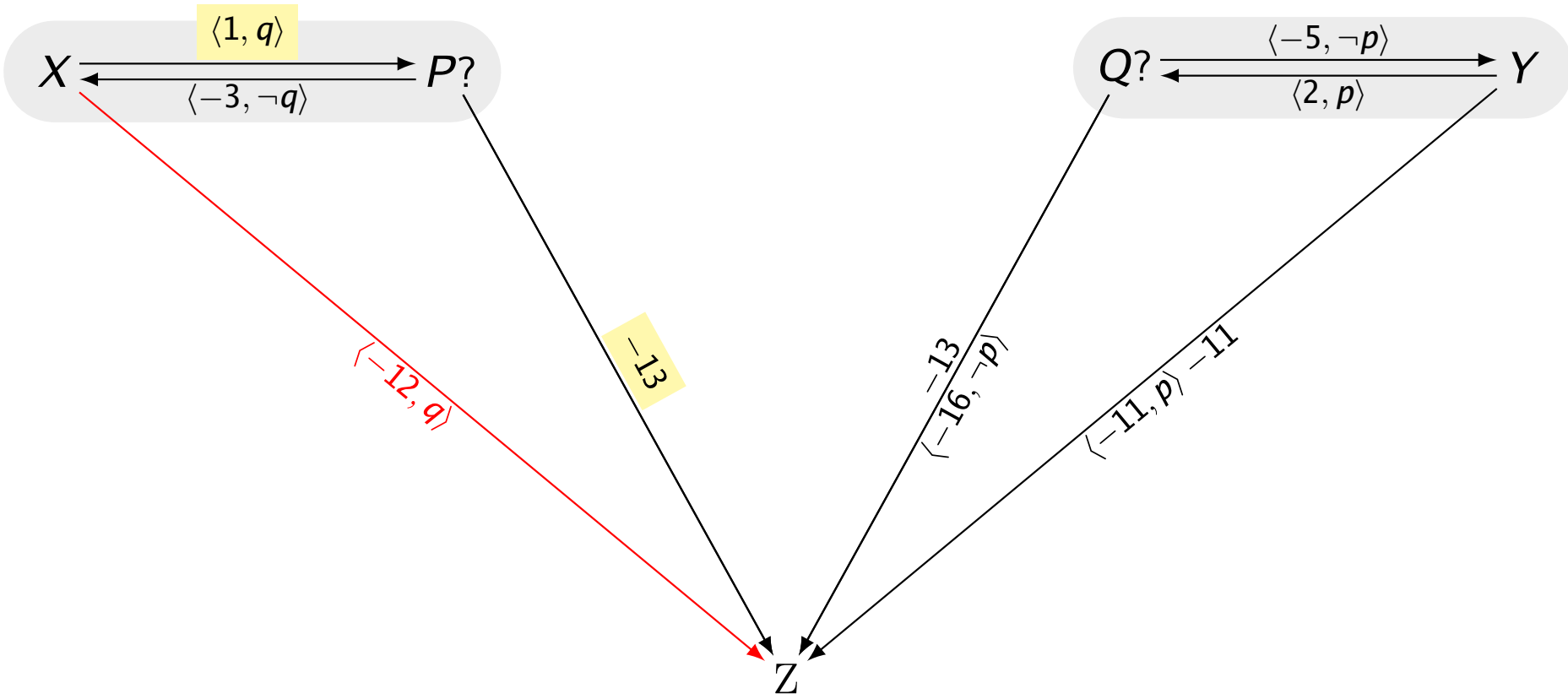
The HP_{18} Algorithm

Simulation



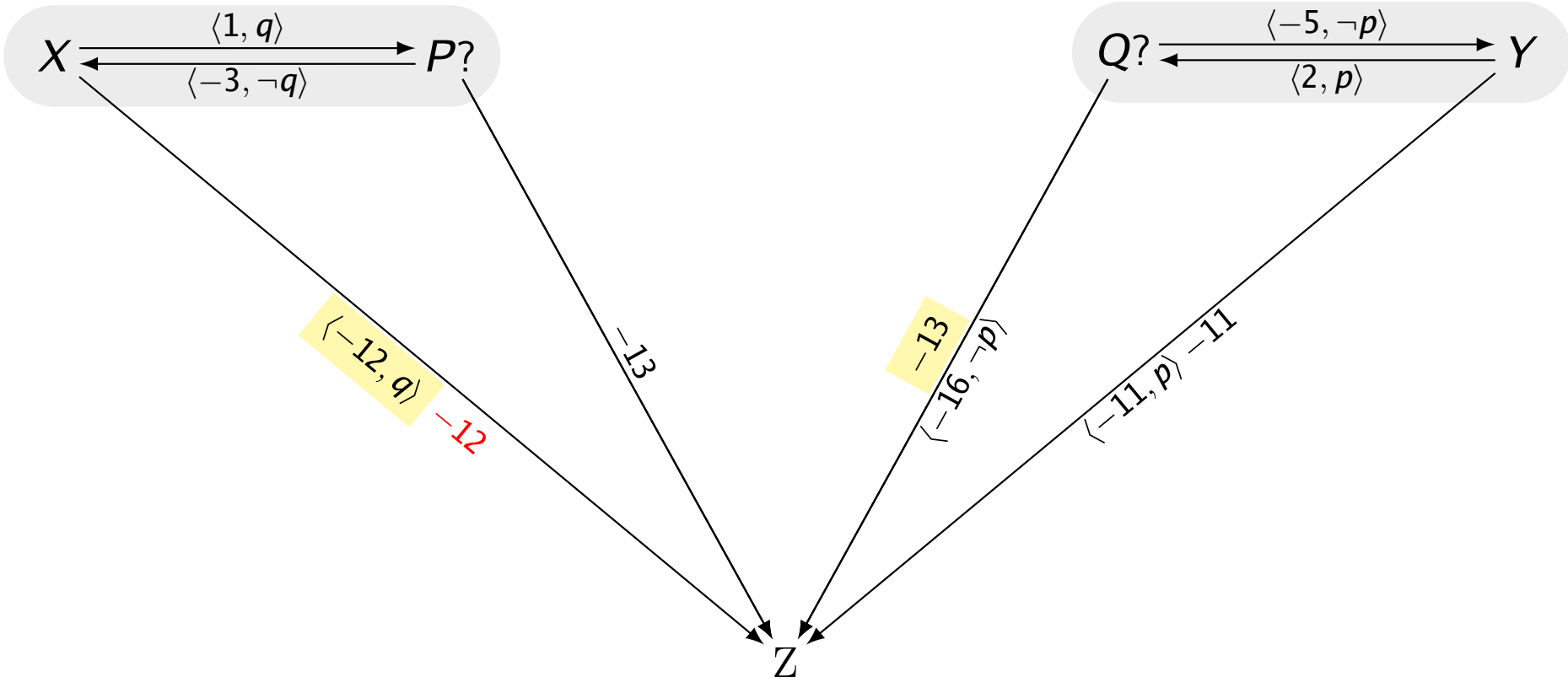
The HP_{18} Algorithm

Simulation



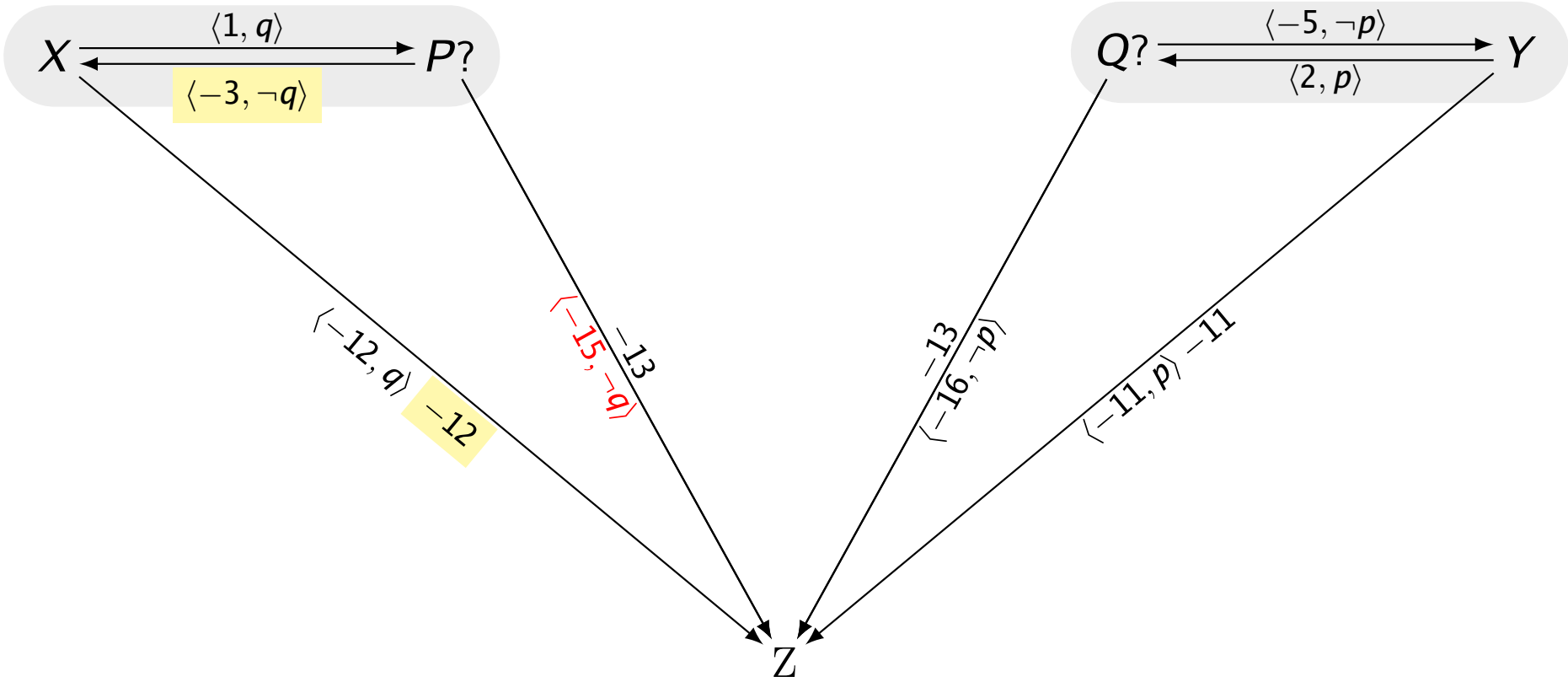
The HP_{18} Algorithm

Simulation



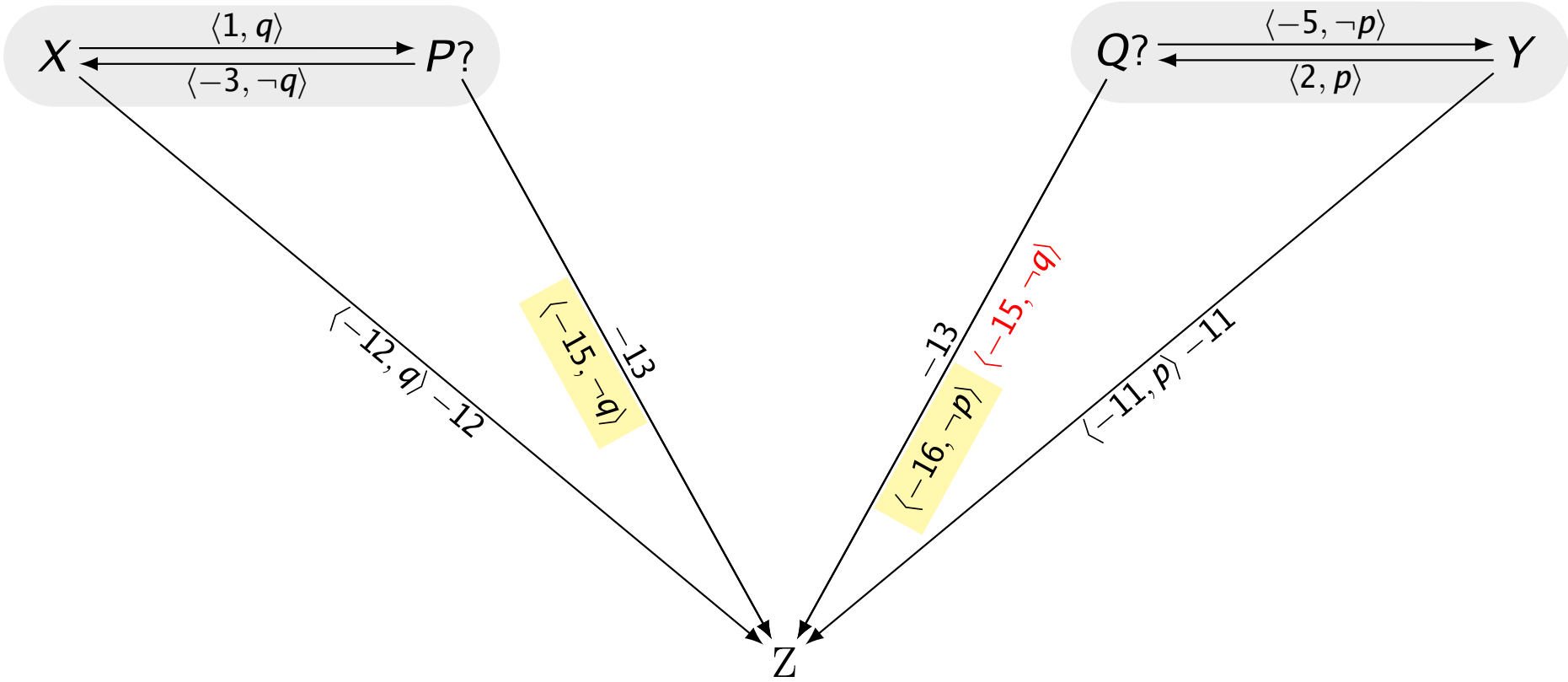
The HP_{18} Algorithm

Simulation



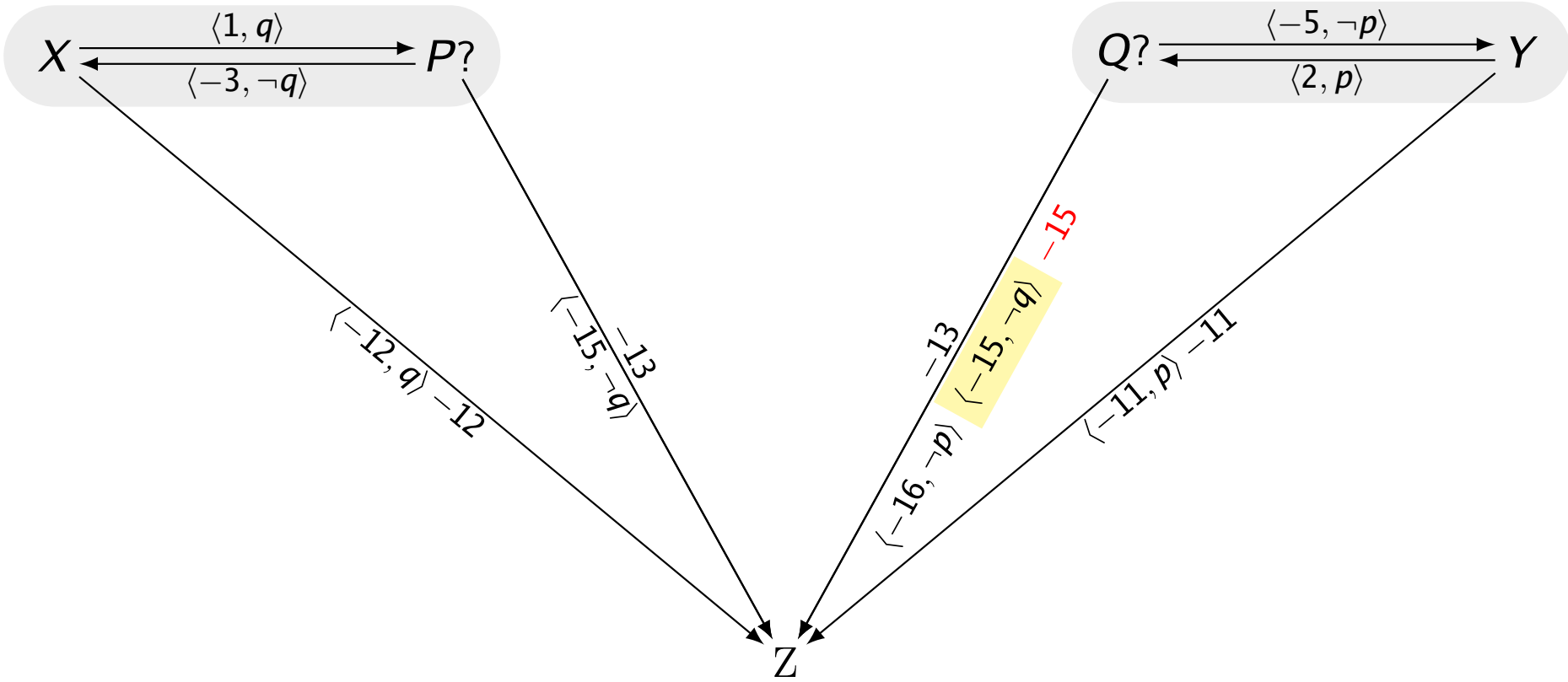
The $\mathtt{HP}_{18}$ Algorithm

Simulation



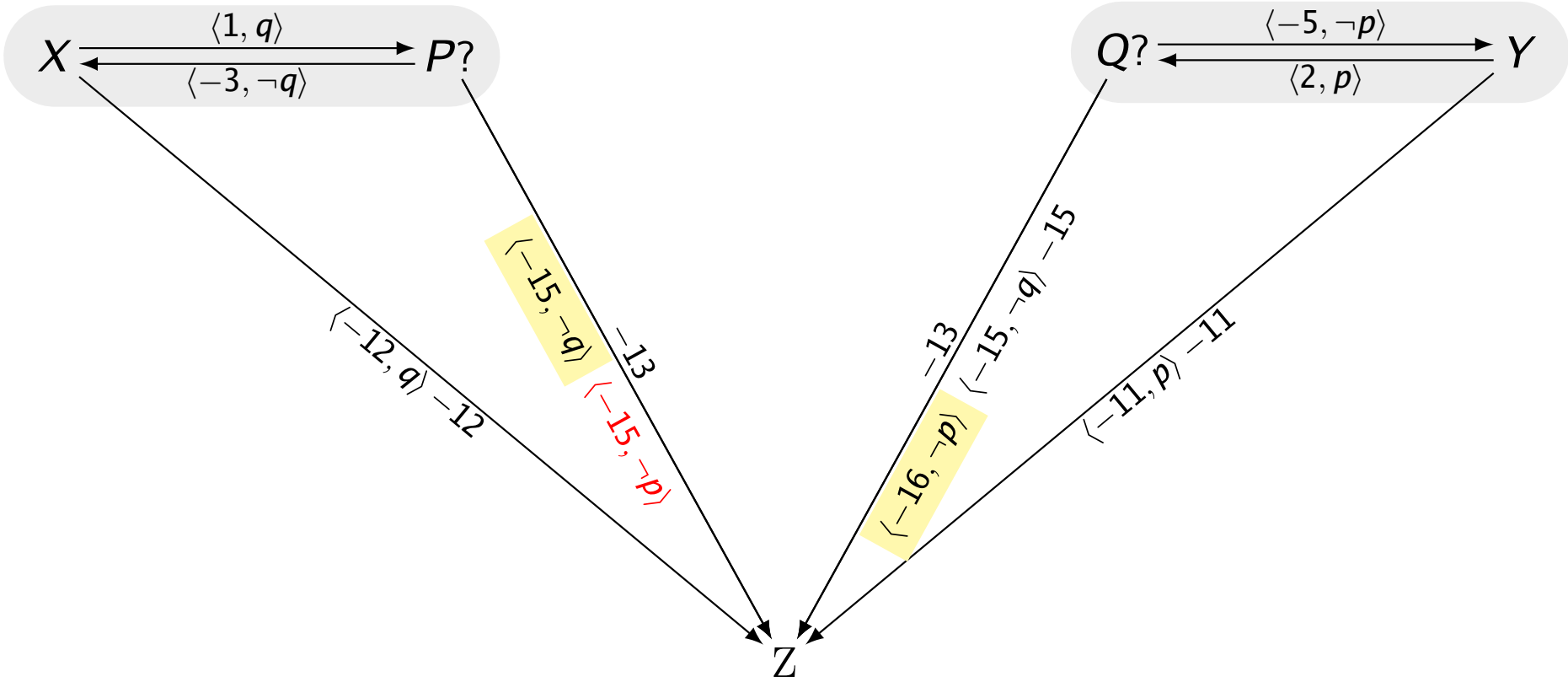
The HP_{18} Algorithm

Simulation



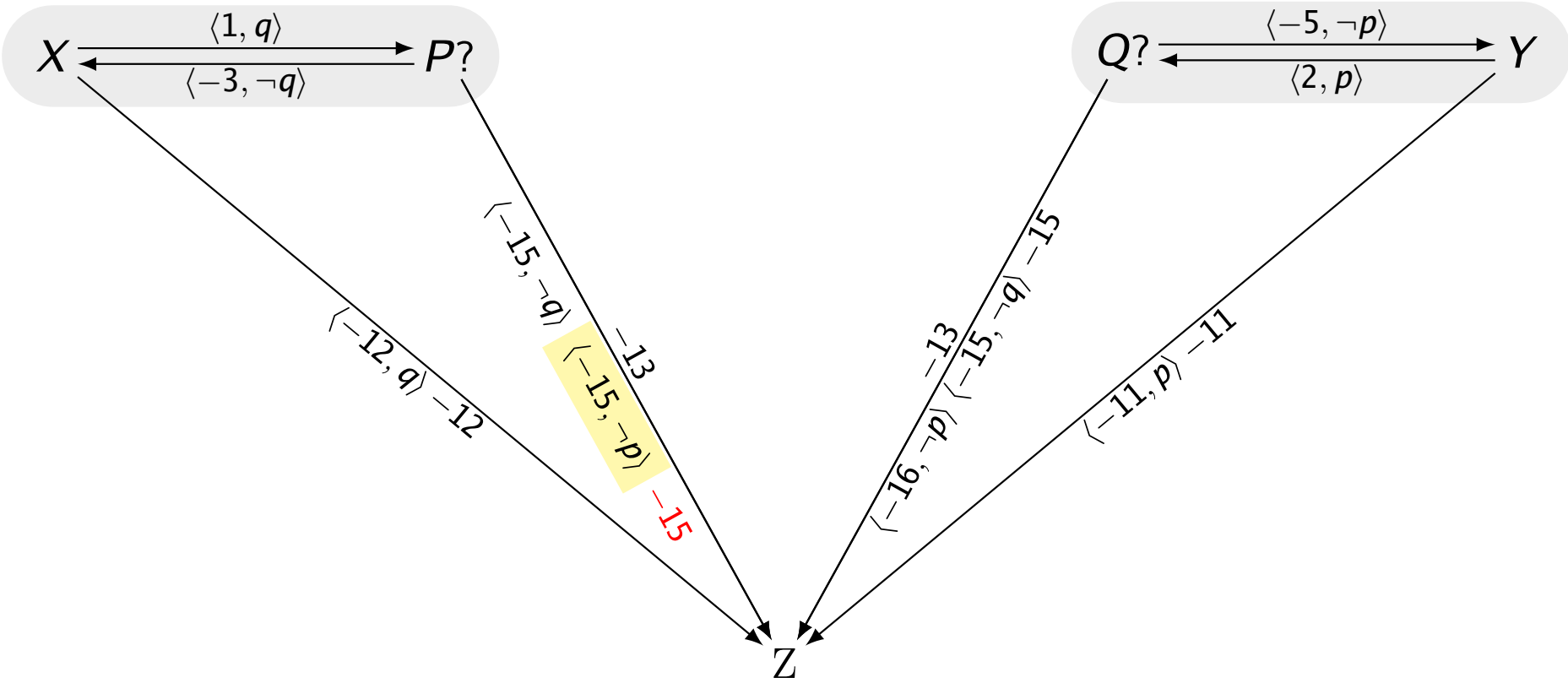
The HP_{18} Algorithm

Simulation



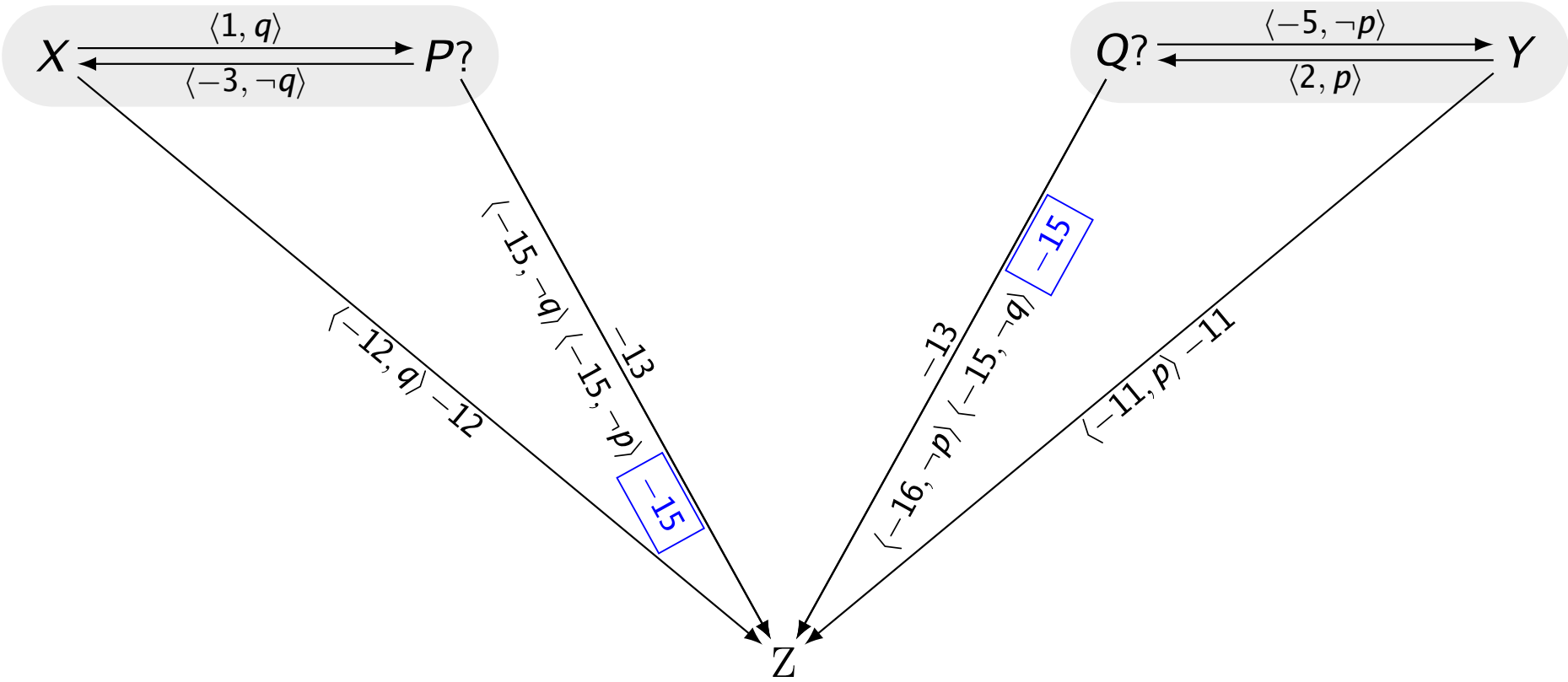
The HP_{18} Algorithm

Simulation



The $\mathtt{HP}_{18}$ Algorithm

Simulation



The HP_{19} Algorithm

Hunsberger & Posenato (2019) [4]

- Goal: Recognize negative q–loops.
- New rule leads to immediate value of $-\infty$ on certain self–loops

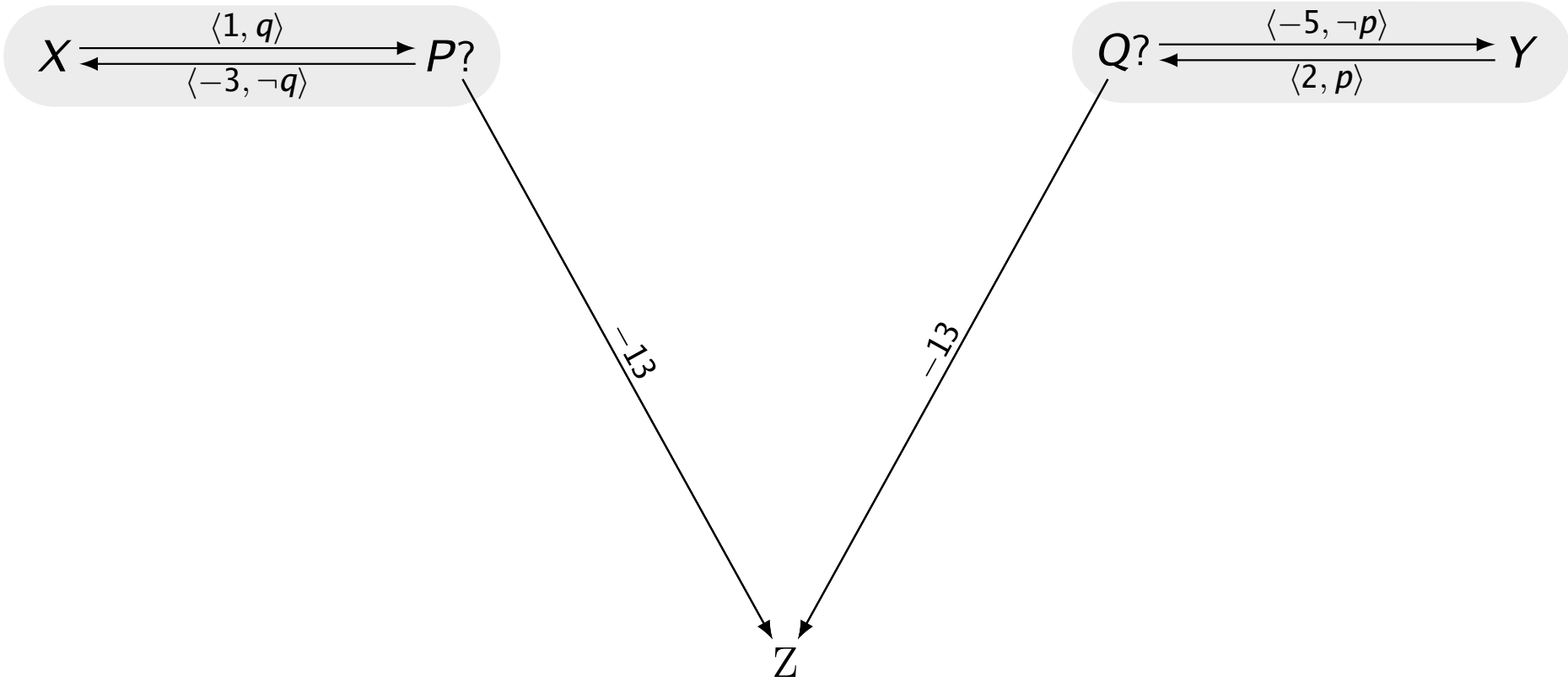
  Example: $\langle -\infty, p(?q)(?r)\rangle \circlearrowright X \underset{\langle 2, q(\neg r)\rangle}{\overset{\langle -8, p(\neg q)r\rangle}{\rightleftarrows}} W$ Meaning:

  X must not be executed as long as p *might* be true, and q and r are both unknown.
- Propagation rules extended to allow propagating $-\infty$ values.

  Example: $\langle -\infty, p(?q)\rangle \circlearrowright X \underset{\langle -\infty, (?p)(?q)\rangle}{\overset{\langle -3, \neg p\rangle}{\longrightarrow}} W$
- Propagation rules generate edges between any time–points —not just edges aimed at Z.
- HP_{19} not empirically evaluated (was not main focus of paper).

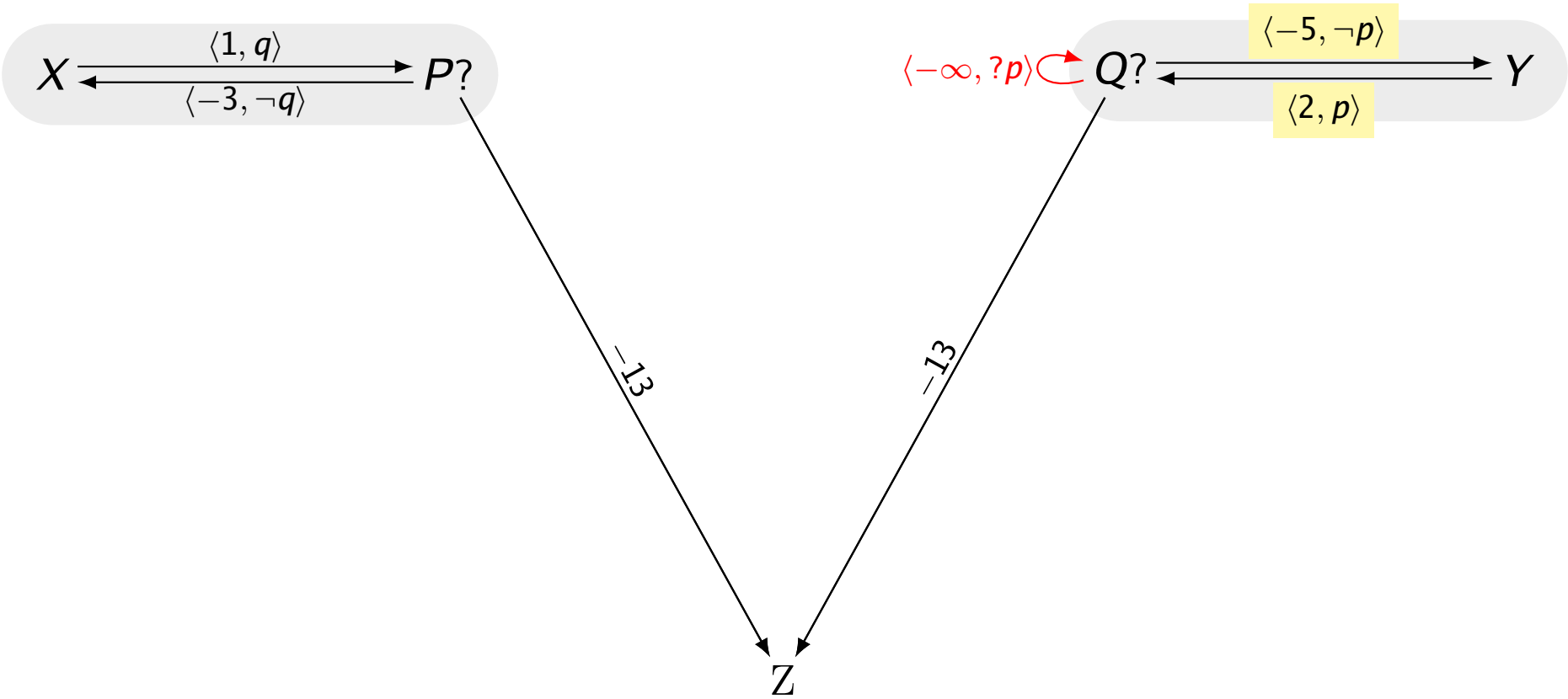
The $\mathtt{HP}_{19}$ Algorithm

Simulation



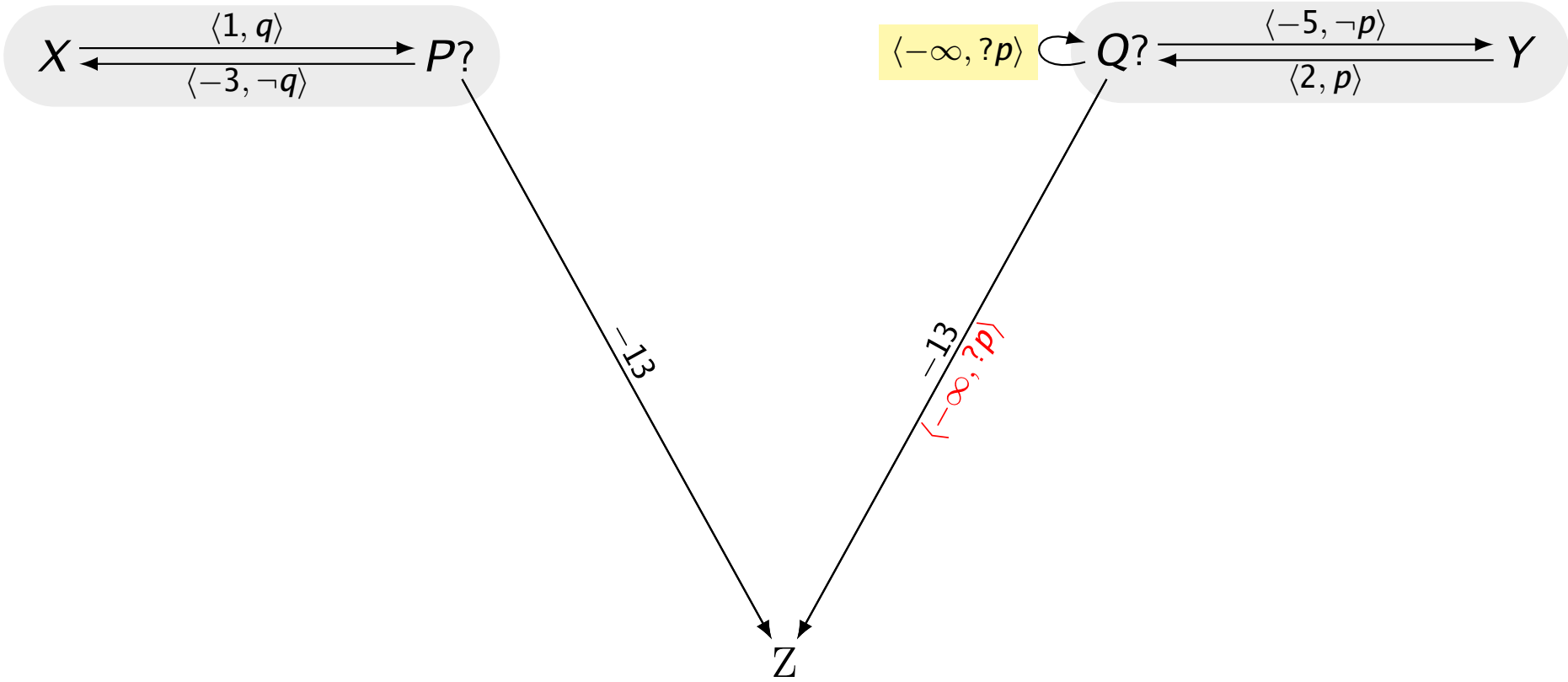
The HP_{19} Algorithm

Simulation



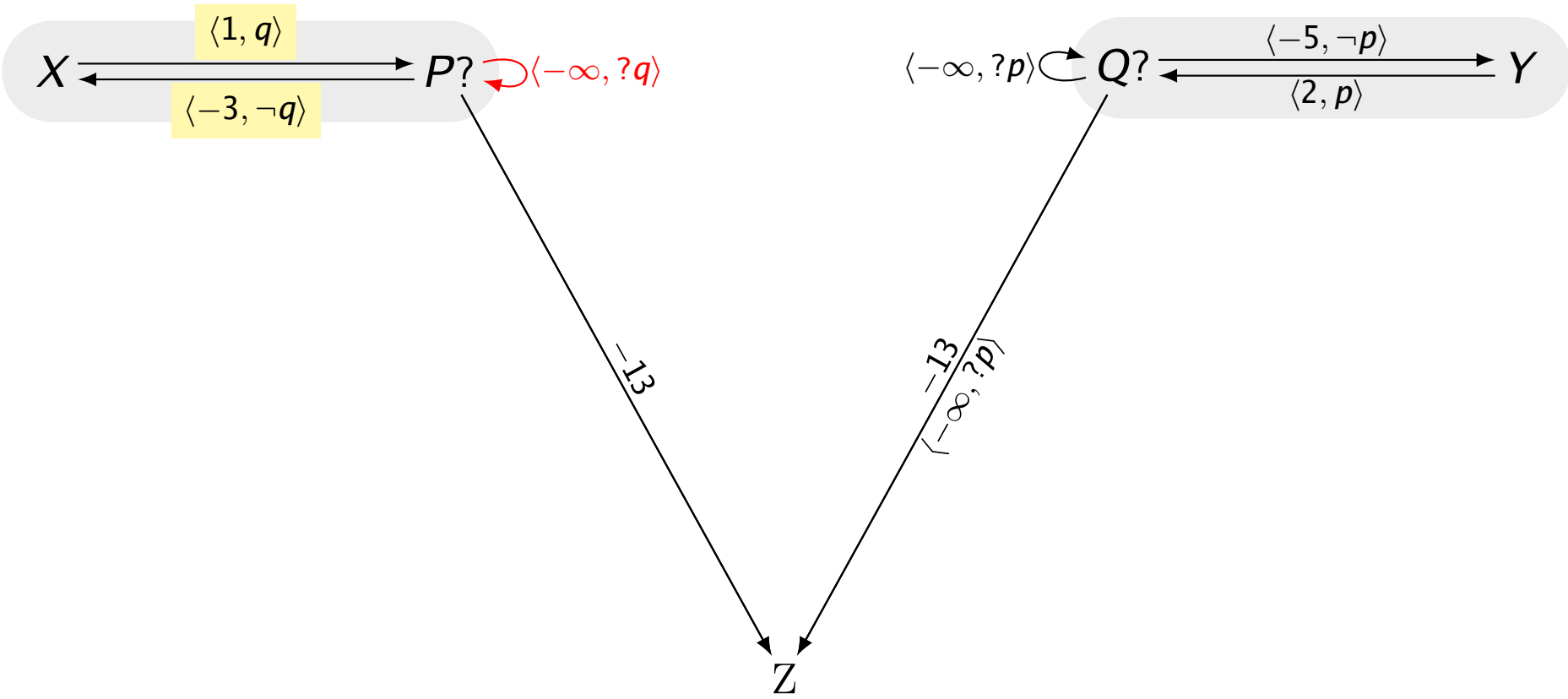
The HP_{19} Algorithm

Simulation



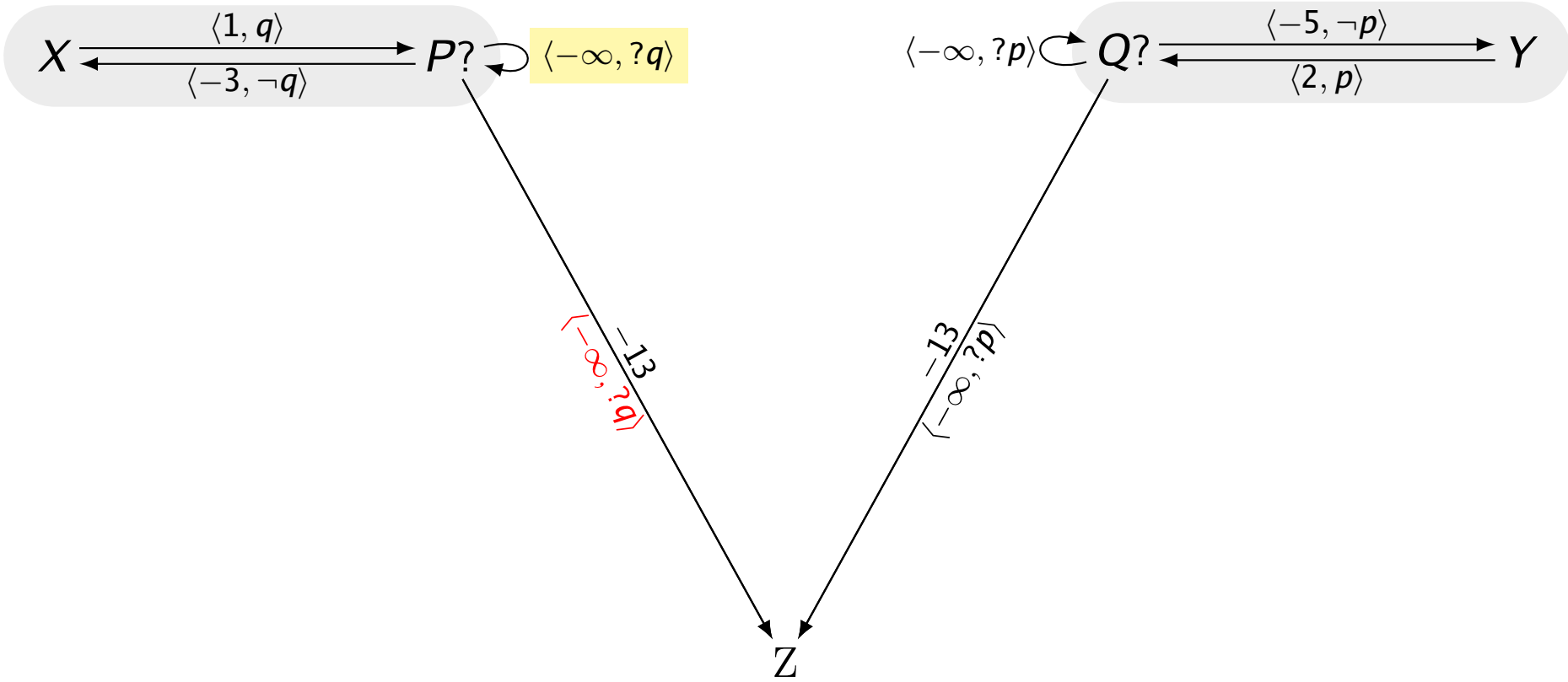
The HP_{19} Algorithm

Simulation



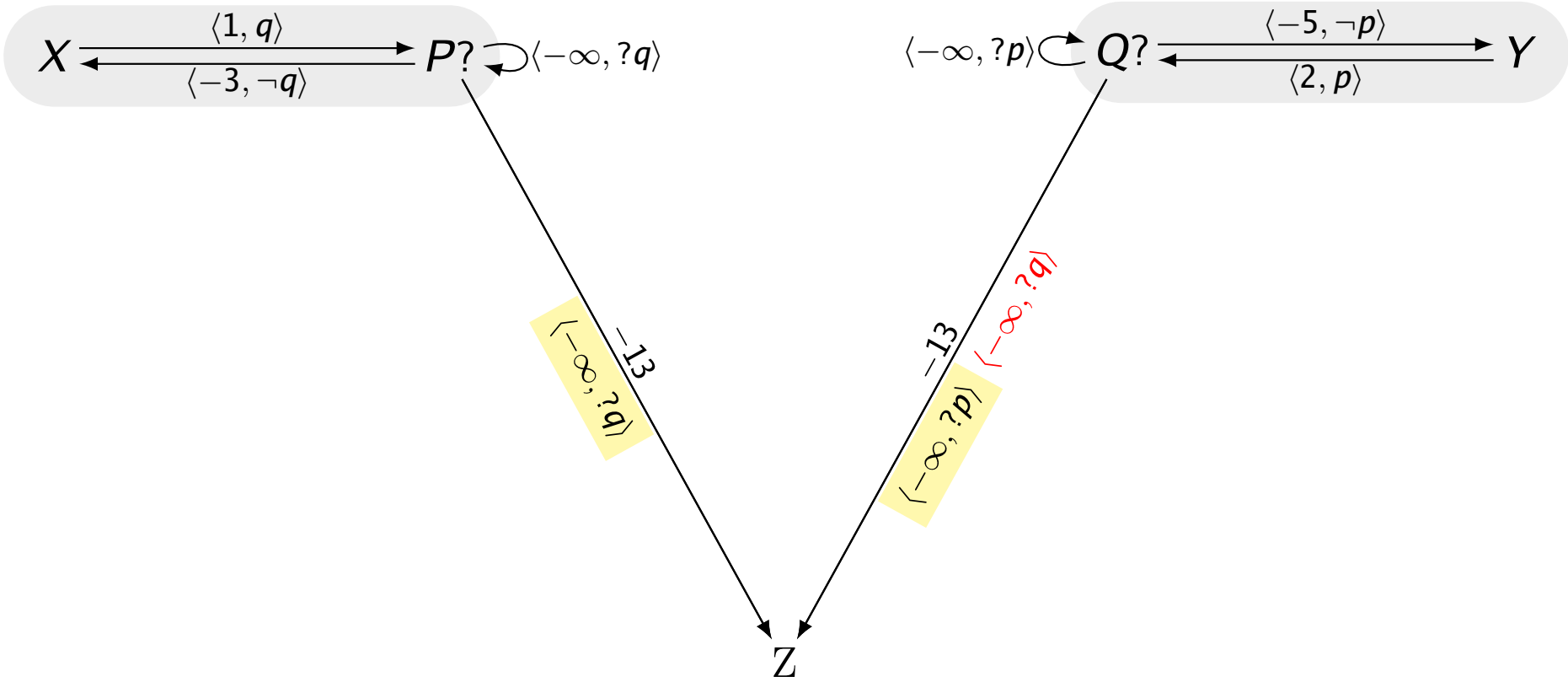
The $\mathtt{HP}_{19}$ Algorithm

Simulation



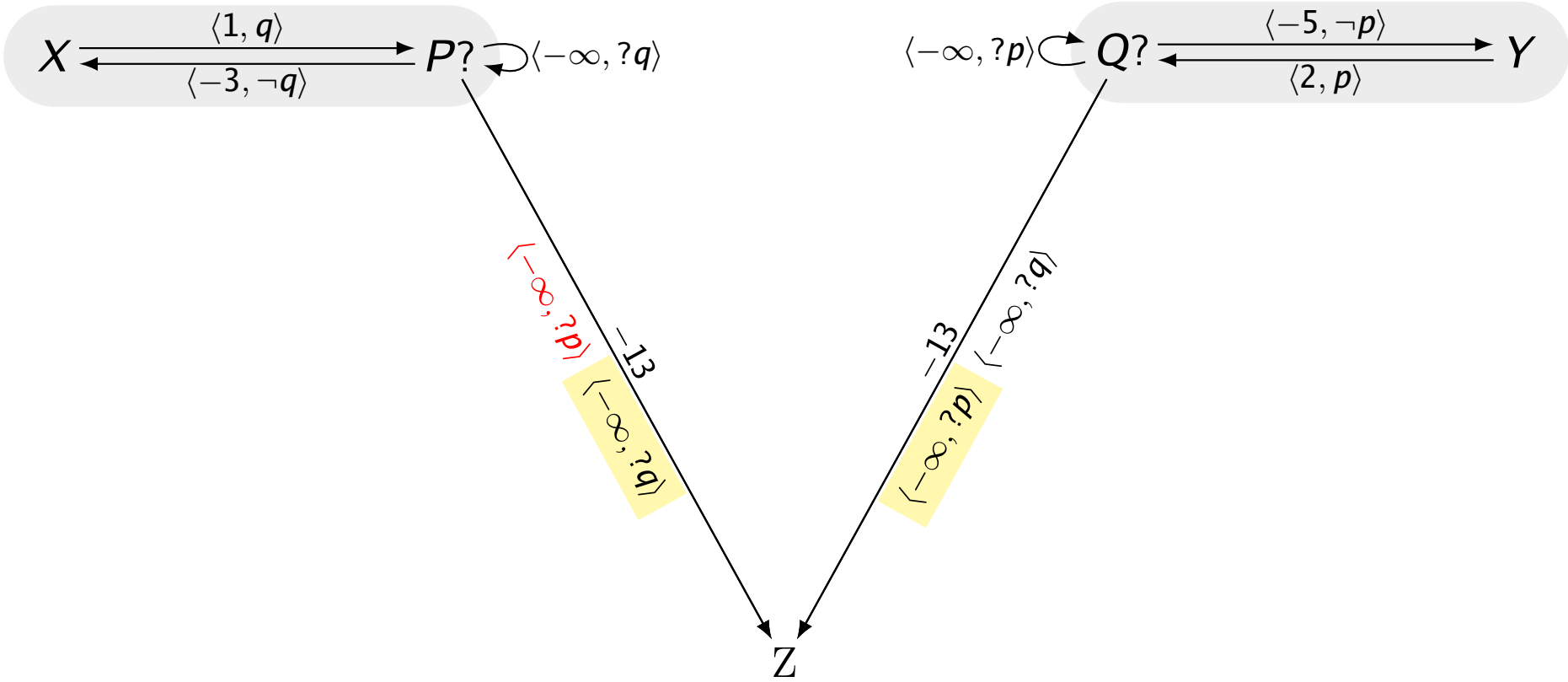
The $\mathtt{HP}_{19}$ Algorithm

Simulation



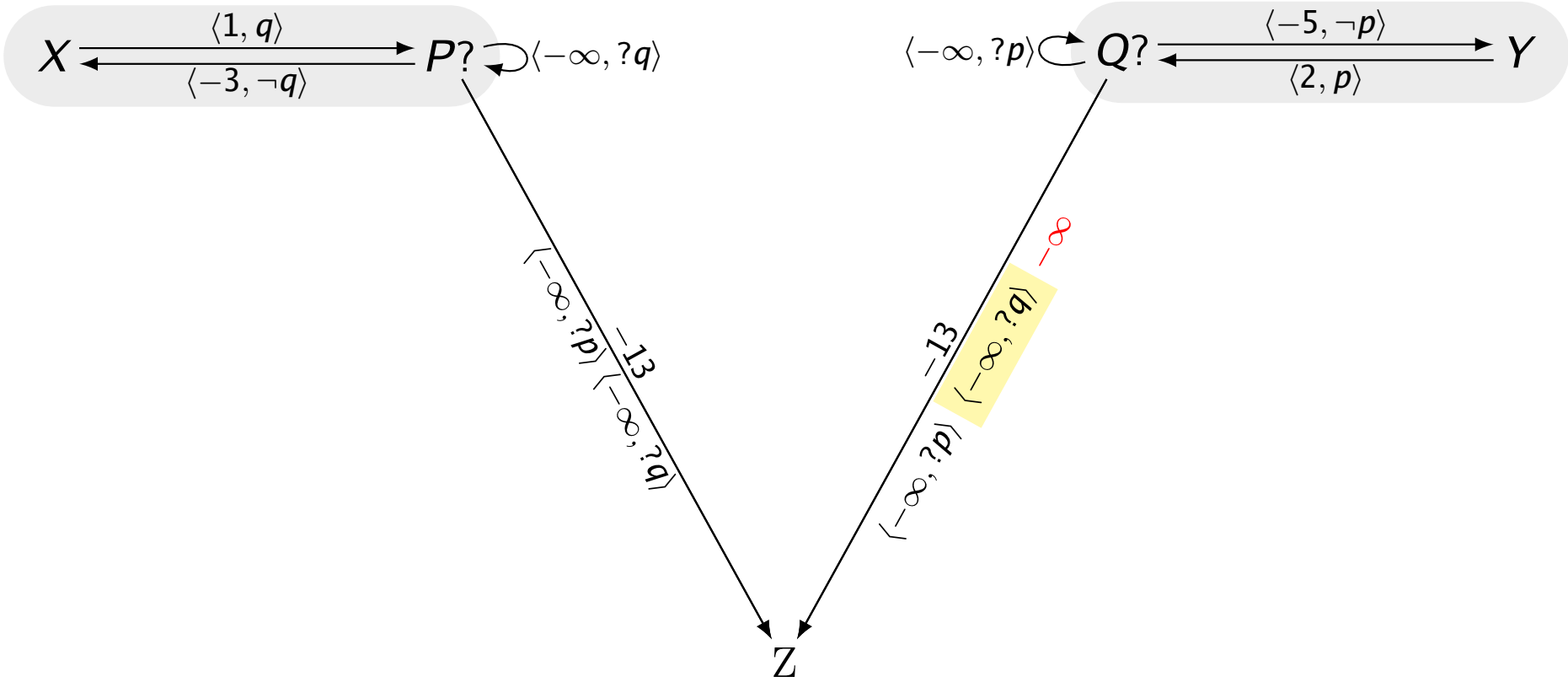
The $\mathtt{HP}_{19}$ Algorithm

Simulation



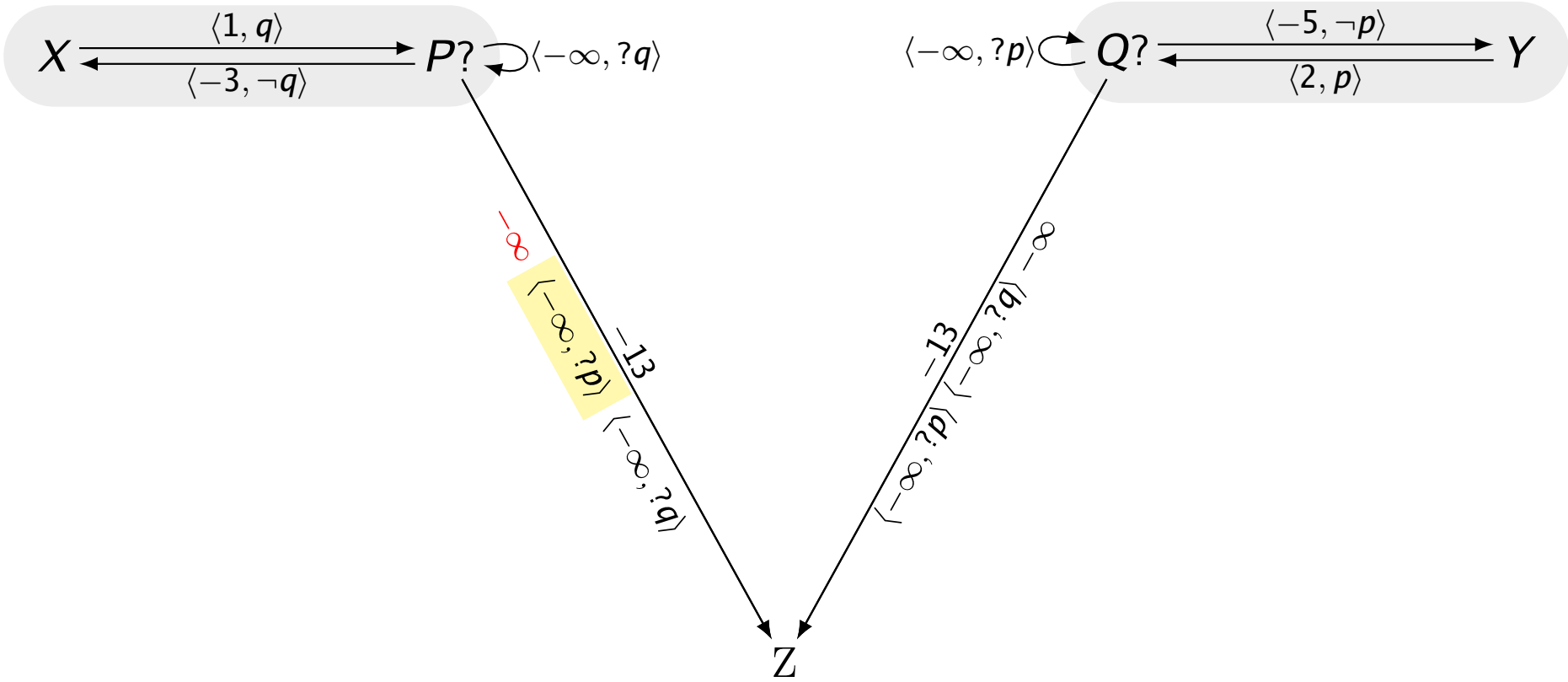
The HP_{19} Algorithm

Simulation



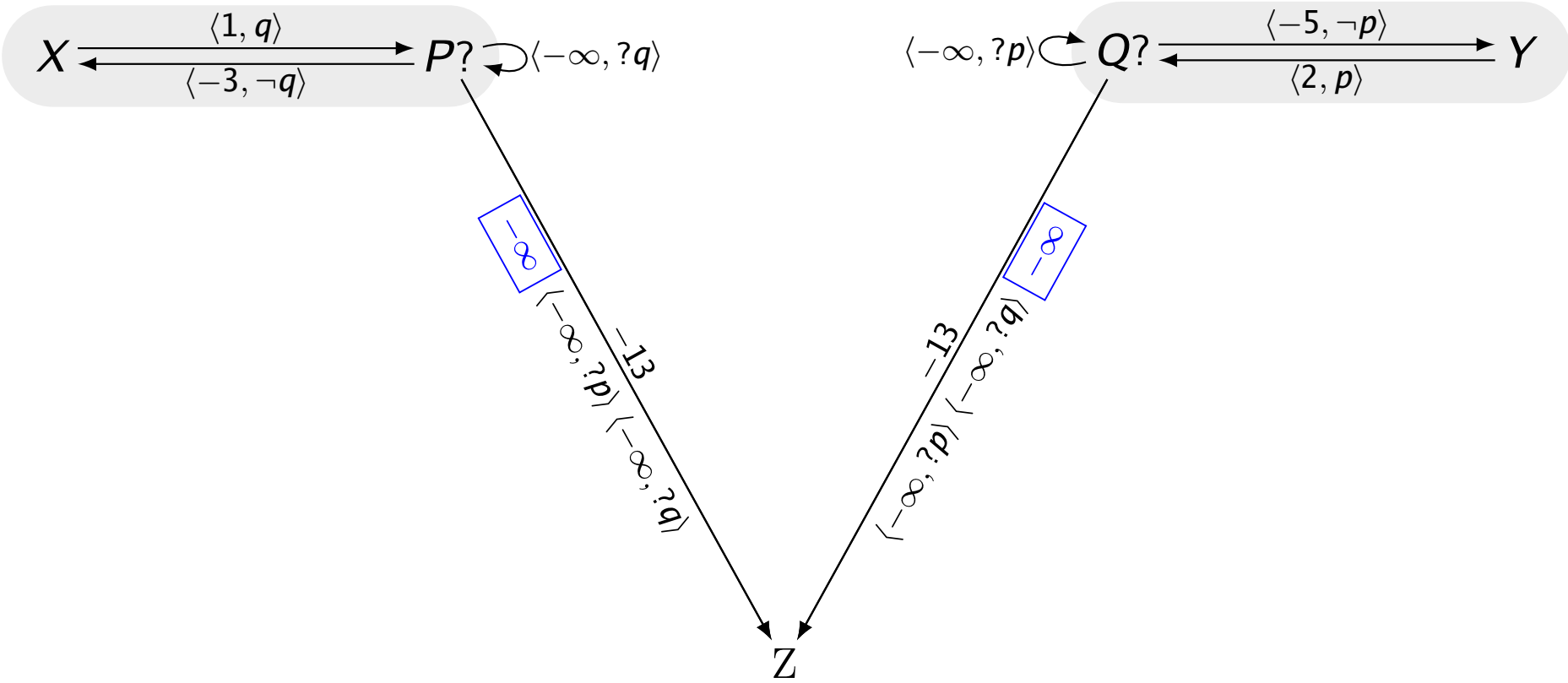
The HP_{19} Algorithm

Simulation



The $\mathtt{HP}_{19}$ Algorithm

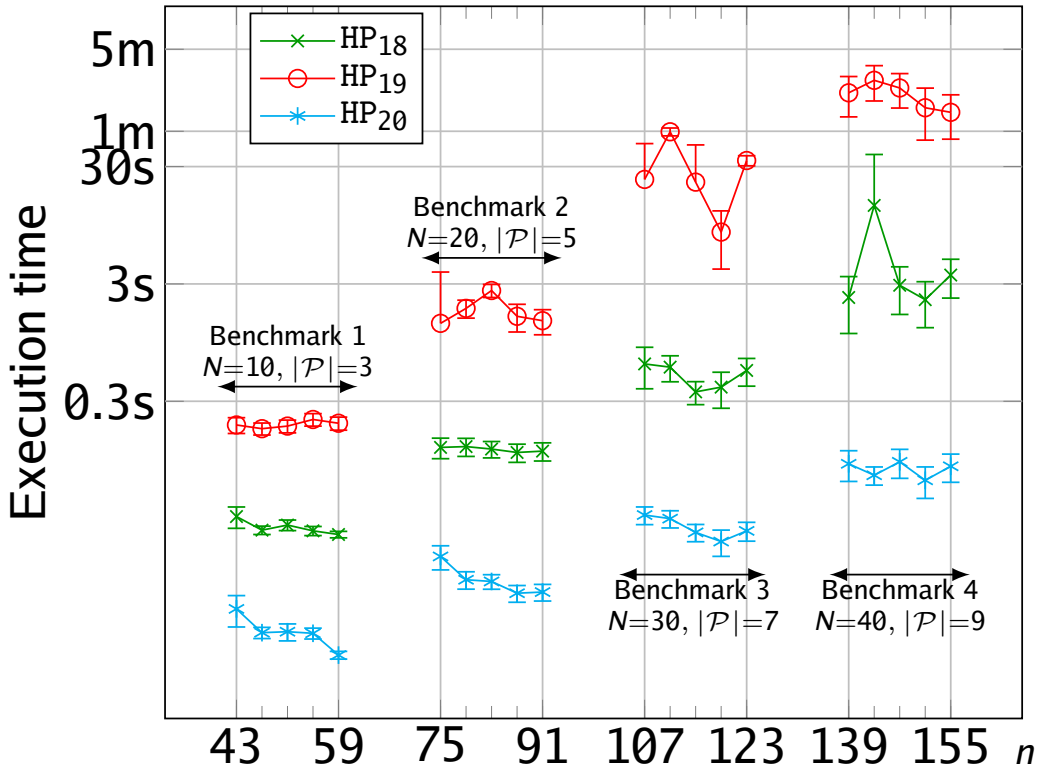
Simulation



The $\mathtt{HP}_{20}$ Algorithm

- Observation: Semantics of edge labeled by $\langle -\infty, \alpha \rangle$ depends only on source node, not on target node.
- Thus, much propagation of $\langle -\infty, \alpha \rangle$ by $\mathtt{HP}_{19}$ is redundant.
- Solution:
  - Associate $\langle -\infty, \alpha \rangle$ with node, not edge.
  - `NQLFinder`: pre–process that discovers some negative q–loops, generating $\langle -\infty, \alpha \rangle$ values for some nodes.
  - All remaining propagation generates edges aimed only at Z. Such edges can be viewed as providing potentials for nodes.
  - Main algorithm focuses exclusively on labeled values for nodes —not labeled values on edges.

Empirical Evaluation — Sample Results



- All algorithms implemented in Java, freely available.
- Each plotted point represents the average execution time over 250 DC–instances generated from random workflows [2].

Conclusions: DC–checking for CSTNs

- The $\texttt{HP}_{18}$ algorithm can get bogged down, cycling through negative q–loops.
- The $\texttt{HP}_{19}$ algorithm can avoid cycling through certain negative q–loops, but does lots of redundant propagation of $\langle -\infty, \alpha \rangle$ values.
- The new $\texttt{HP}_{20}$ algorithm is significantly faster than $\texttt{HP}_{18}$ and $\texttt{HP}_{19}$ across existing benchmarks and a new set of benchmarks.
  - $\texttt{HP}_{20}$ more efficiently identifies negative q–loops and more efficiently manages the propagation of labeled values of the form $\langle -\infty, \alpha \rangle$.
  - $\texttt{HP}_{20}$, unlike previous algorithms, only updates labeled values—whether finite or infinite—on nodes, not edges.

References I

[1] Massimo Cairo, Luke Hunsberger, Roberto Posenato, and Romeo Rizzi.
A Streamlined Model of Conditional Simple Temporal Networks – Semantics and Equivalence Results.
In *24th Int. Symp. on Temporal Representation and Reasoning (TIME–2017)*, volume 90 of *LIPIcs*, pages 10:1–10:19, 2017.

[2] 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.

[3] 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, 2018.

[4] Luke Hunsberger and Roberto Posenato.
Propagating Piecewise–Linear Weights in Temporal Networks.
In *29th International Conference on Automated Planning and Scheduling, ICAPS 2019*, volume 29, pages 223–231. AAAI Press, 2019.

References II

[5] 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.

[6] Ioannis Tsamardinos, Thierry Vidal, and Martha E. Pollack.
CTP: A new constraint–based formalism for conditional, temporal planning.
Constraints, 8:365–388, 2003.