Updated 1/25/2020

T1: Data models and query languages
L6: Datalog vs. Stable models

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp20)

https://northeastern-datalab.github.io/cs7240/sp20/

Version 1/24/2020

Outline: Datalog

- Datalog
  - Datalog rules
  - Recursion
  - Semantics
  - Datalog$^\neg$: Negation, stratification
  - Datalog$^\pm$
  - Stable model semantics (Answer set programming)
  - Datalog vs. RA
  - Naive and Semi-naive evaluation

Answer Set Programming

- Programming paradigm that can model AI problems (e.g, planning, combinatorics)
- Basic idea
  - Allow non-stratified negation and encode problem (specification & "instance") as logic program rules
  - Solutions are stable models of the program
- Semantics based on Possible Worlds and Stable Models
  - Given an answer set program, there can be multiple solutions (stable models, answer sets)
  - Each model: assignment of true/false value to propositions to make all formulas true.
  - Captures default reasoning, non-monotonic reasoning, constrained optimization, exceptions, weak exceptions, preferences, etc., in a natural way
- Finding stable models of answer set programs is not easy
  - Current systems CLASP, DLV, Smodels, etc., extremely sophisticated
  - Work by grounding the program, suitably transforming it to a propositional theory whose models are stable models of the original program
  - These models found using a SAT solver

Rules with Negation

- Closed world assumption
  - If a fact does not logically follow from a set of Datalog clauses, then we conclude that the negation of this fact is true.
- Problem: multiple minimal models ("Herbrand models")
  - boring(chess) :- not interesting(chess)
  - $H_a = \{interesting(chess)\},\ H_b = \{boring(chess)\}$

Semantics: Informally

- Informally, a stable model M of a ground program P is a set of ground atoms such that
  1. Every rule is satisfied:

     i.e., for any rule in P

     $$A \text{ :- } B_1, \dots, B_m, \text{not } C_1, \dots, \text{not } C_n.$$

     if each B_j is satisfied (B_i's are in M) and no C_i is satisfied (i.e. no C_i is is in M), then A is in M.
  2. Every A $\in$ M can be derived from a rule by a "non-circular reasoning" (informal for: we are looking for minimal models)

$? \{a\} > \{\}$ a :- a

Semantics: "non-circular" more formally

Idea: you guess a set of atoms. You then verify it is indeed exactly the set of atoms that "can be derived."

The reduct of P w.r.t M is:

$$P^M = \left\{ \boxed{h \text{ :- } b_1, \ldots, b_m.} \;\middle|\; \boxed{h \text{ :- } b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n.} \in P \wedge \text{no } C_i \in M \right\}$$

M is a stable model of P iff M is the least model of P^M

Examples

edge(?, a) :-

P1: `a :- a.`

~~M={a}~~ not a stable model (not minimal, derivation of a is based on a circular reasoning)

M={} stable model

P2: `a :- not b.`

M={a} only stable model

~~M={b}~~

P3: `a :- not a.`

{} ↔ {a}

has no stable model

{a}

For a normal program without negation ("positive program"), its least model is the set of its atomic consequences.

a ←

vs.

`a :-` {a}

`a :- a.` → {a}

Examples

P4:

```
a :- not b.
b :- not a.
```

M_1={a}

M_2={b}

two stable models

a :- ¬ ~~b~~

~~b :- not a~~

a ←

P5:

a :- ~~not b.~~

~~b :- not a.~~

~~a :- not a.~~

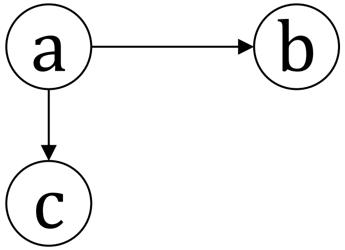
M={a} only stable model

a :- ¬ a

a :-

3-colorability

Q: For a graph (V, E) find an assignment of one of 3 colors to each vertex such that no adjacent vertices share a color.



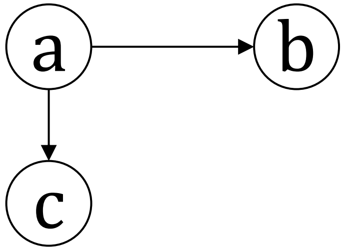
?

Convention in ASP: Capital letters are variables, small letters constants

Cp. edge(X,a)
vs. edge(x,'a')

3-colorability

Q: For a graph (V, E) find an assignment of one of 3 colors to each vertex such that no adjacent vertices share a color.




```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).
```

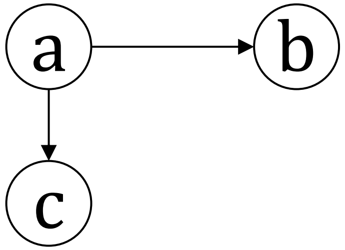
?

Convention in ASP: Capital letters are variables, small letters constants

Cp. edge(X,a)
vs. edge(x,'a')

3-colorability

Q: For a graph (V, E) find an assignment of one of 3 colors to each vertex such that no adjacent vertices share a color.




```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).


color(V,1) :- not color(V,2), not color(V,3), vertex(V).
color(V,2) :- not color(V,3), not color(V,1), vertex(V).
color(V,3) :- not color(V,1), not color(V,2), vertex(V).
```

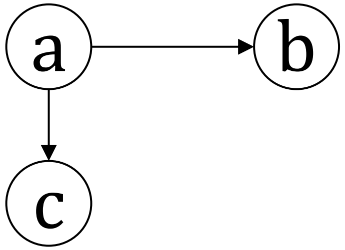
?

Convention in ASP: Capital letters are variables, small letters constants

Cp. edge(X,a)
vs. edge(x,'a')

3-colorability

Q: For a graph (V, E) find an assignment of one of 3 colors to each vertex such that no adjacent vertices share a color.



EDB

```
vertex(a). vertex(b). vertex(c). edge(a,b). edge(a,c).

color(V,1) :- not color(V,2), not color(V,3), vertex(V).
color(V,2) :- not color(V,3), not color(V,1), vertex(V).
color(V,3) :- not color(V,1), not color(V,2), vertex(V).

:- edge(V,U), color(V,C), color(U,C).
```

Convention in ASP: Capital letters are variables, small letters constants

Cp. edge(X,a) vs. edge(x,'a')

Data Conflict Resolution Using Trust Mappings

SIGMOD 2010

Paper: http://portal.acm.org/citation.cfm?id=1807167.1807193

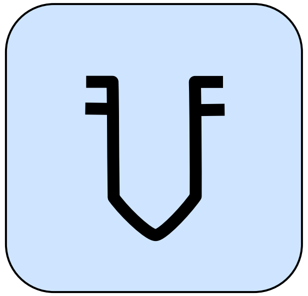
Full version with proofs: http://arxiv.org/pdf/1012.3320

Old Project web page: https://db.cs.washington.edu/projects/beliefdb/

Problem in social data: often no single ground truth

The Indus Script*

What is the origin
of this glyph?



: cow

Bob

: ship hull

Alice

: jar

Charlie

* Current state of knowledge on the Indus Script: Rao et al., Science 324(5931):1165, May 2009

Background: Conflicts & Trust in Community DBs

Conflicting beliefs

glyph	origin	
	ship hull	Alice
	cow	Bob
	jar	Charlie
	fish	Bob
	knot	Charlie
	arrow	Charlie

"Beliefs": annotated (key,value) pairs

Trust mappings

Alice ← Bob	(100)
Alice ← Charlie	(50)
Bob ← Alice	(80)

Priorities

glyph	origin
	cow
	fish
	arrow

100

80

"Explicit belief"

glyph	origin
	ship hull
	fish
	arrow

"Implicit belief"

50

glyph	origin
	jar
	knot
	arrow

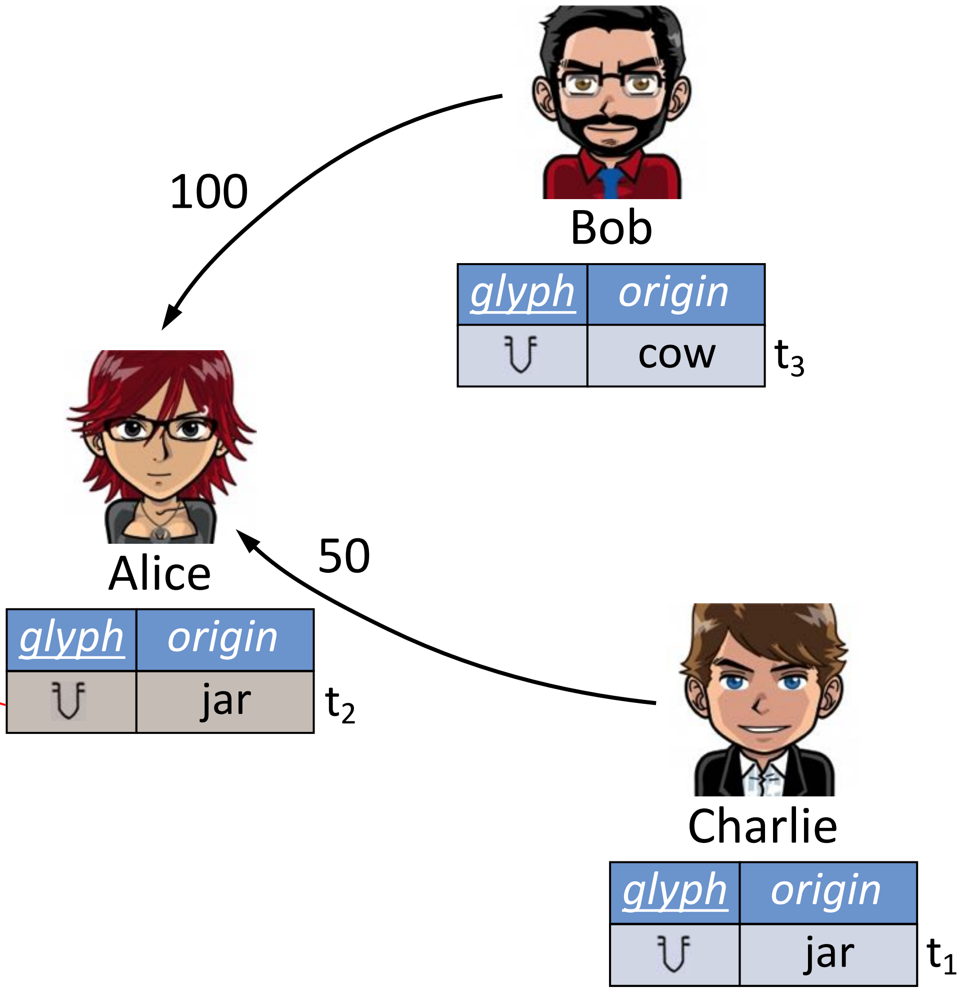
Recent work on community databases:

- Taylor & Ives [SIGMOD'06] ← Orchestra
- Green et al. [VLDB'07] ← Orchestra
- Kot & Koch [VLDB'09] ← Youtopia
- **GBKS** [VLDB'09] ← BeliefDB

Limitations of previous work: transient effects

1. Incorrect inserts
   – Value depends on order of inserts

Alice would have preferred Bob's value over Charlie's

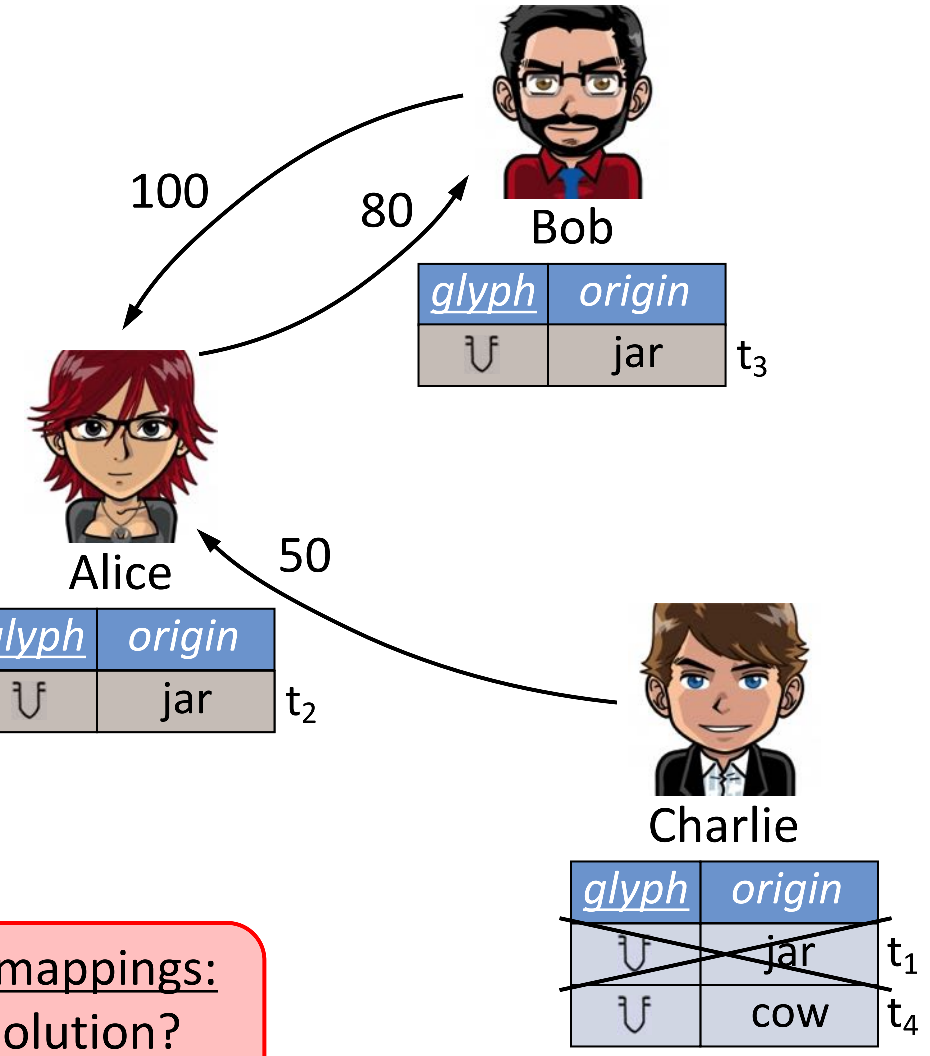


Limitations of previous work: transient effects

1. Incorrect inserts
   – Value depends on order of inserts

2. Incorrect updates
   – Mis-handling of revokes

Alice and Bob trust each other most, but have lost "justification" for their beliefs



<u>Automatic conflict resolution with trust mappings:</u>
1. How to define a globally consistent solution?
2. How to calculate it efficiently?

(3. Several extensions)

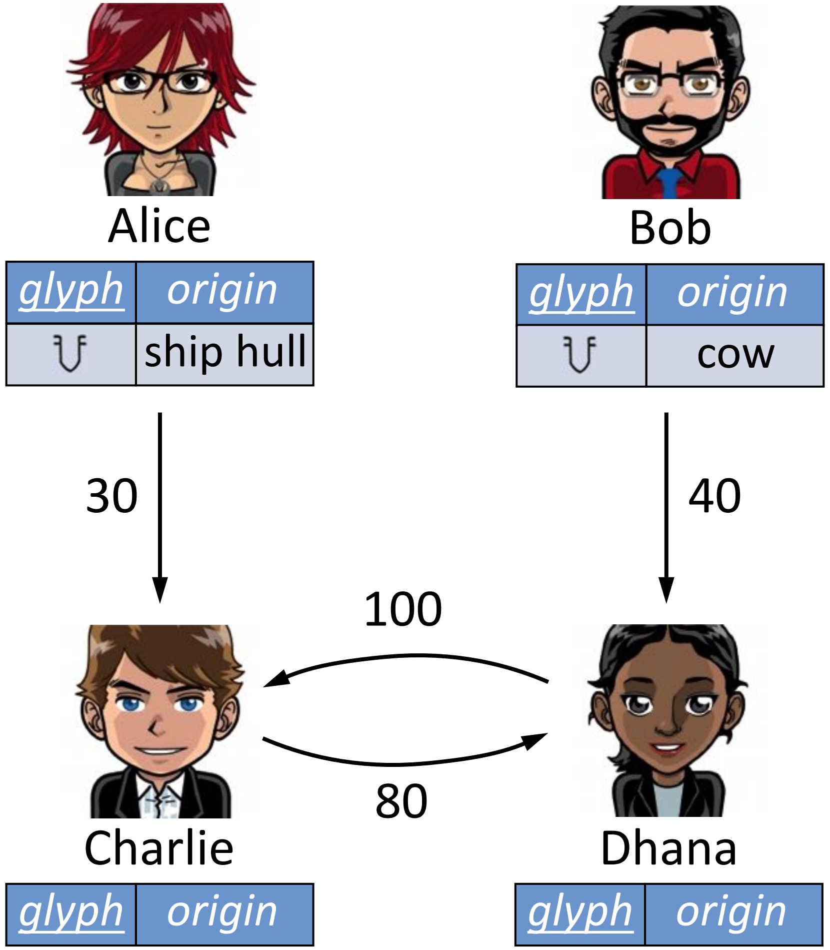
GS [Sigmod'10]

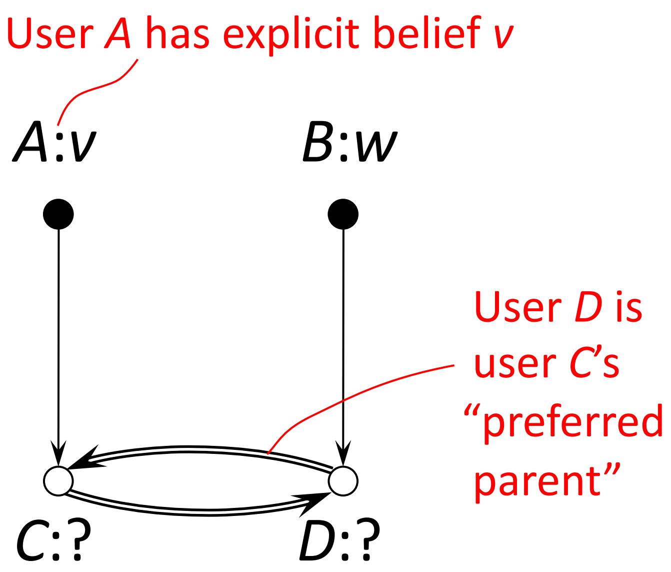
Agenda

1. Stable solutions
   - how to define a unique and consistent solution?
2. Resolution algorithm
   - how to calculate the solution efficiently?
3. Extensions
   - how to deal with “negative beliefs”?

Binary Trust Networks (BTNs)

To simplify presentation: focus on binary TNs

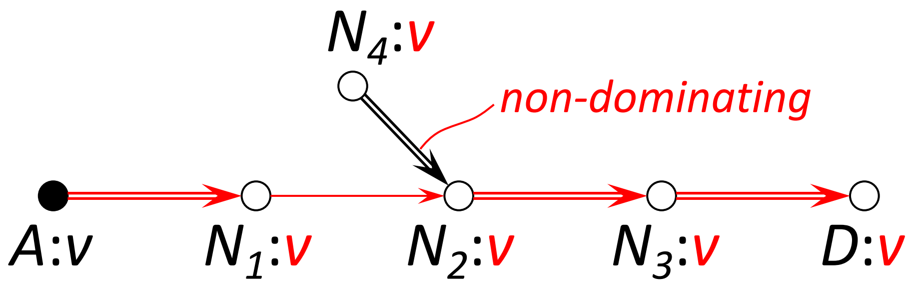


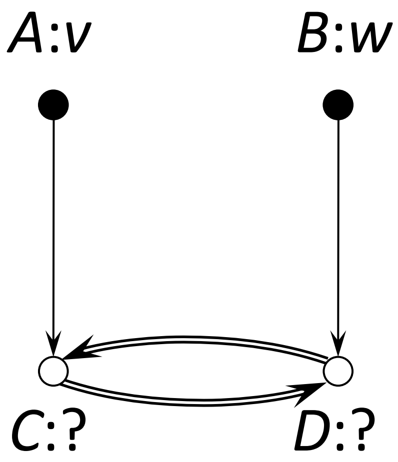


Focus on one single key
(we ignore the glyph)

The definition of a globally consistent solution

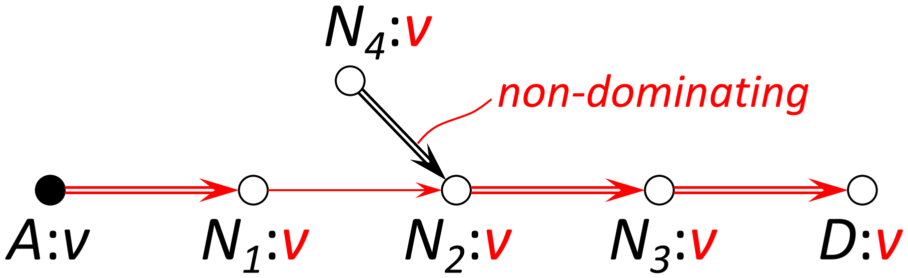
- Stable solution
  - assignment of values to each node, s.t. each belief has a "*non-dominated lineage*" to an explicit belief

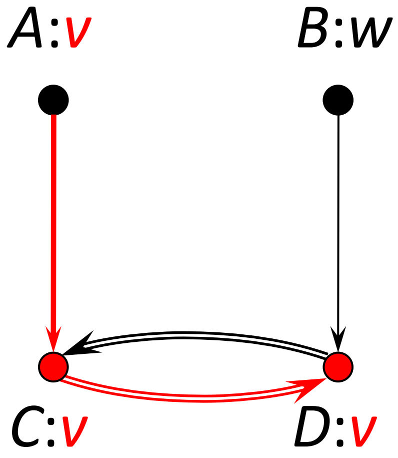




The definition of a globally consistent solution

- Stable solution
  - assignment of values to each node, s.t. each belief has a *"non-dominated lineage"* to an explicit belief

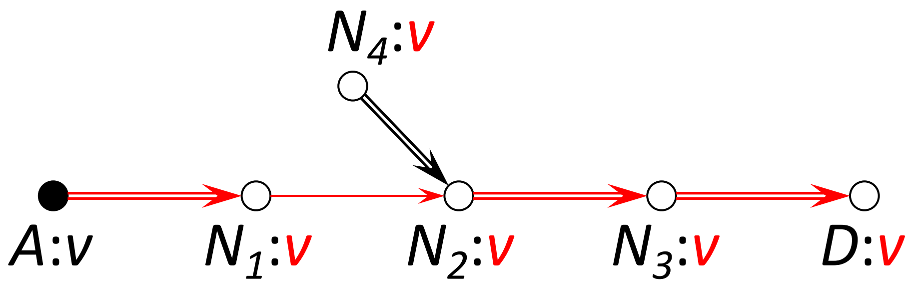


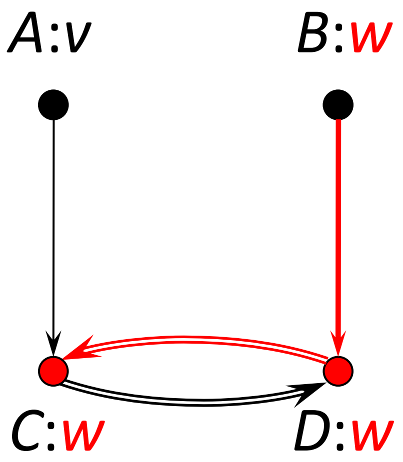


$SS_1=(A{:}v,\ B{:}w,\ C{:}v,\ D{:}v)$

The definition of a globally consistent solution

- Stable solution
  - assignment of values to each node, s.t. each belief has a *"non-dominated lineage"* to an explicit belief

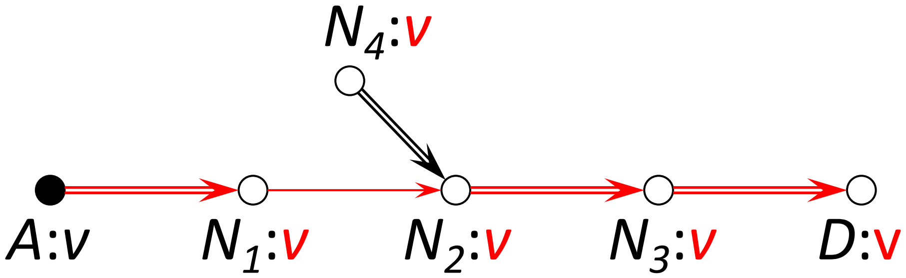


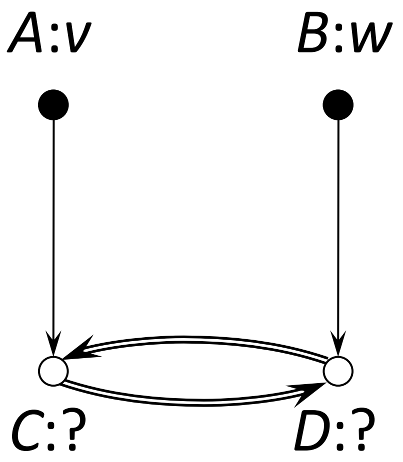


SS_1=(*A*:*v*, *B*:*w*, *C*:*v*, *D*:*v*)
SS_2=(*A*:*v*, *B*:*w*, *C*:*w*, *D*:*w*)

Possible and certain values from all stable solutions

- Stable solution
  - assignment of values to each node, s.t. each belief has a "*non-dominated lineage*" to an explicit belief





$SS_1 = (A{:}v,\ B{:}w,\ C{:}v,\ D{:}v)$
$SS_2 = (A{:}v,\ B{:}w,\ C{:}w,\ D{:}w)$

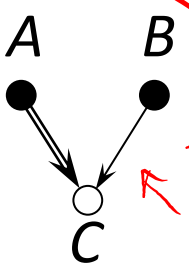
- Possible / Certain semantics
  - a stable solution determines, for each node, a possible value ("**poss**")
  - certain value ("**cert**") = intersection of all stable solutions, per user

X	**poss**(*X*)	**cert**(*X*)
A	{*v*}	{*v*}
B	{*w*}	{*w*}
C	{*v*,*w*}	∅
D	{*v*,*w*}	∅

poss	(v, w)
	A v
	C v
	C w

Logic programs (LP) with stable model semantics

- LPs can capture this semantics.



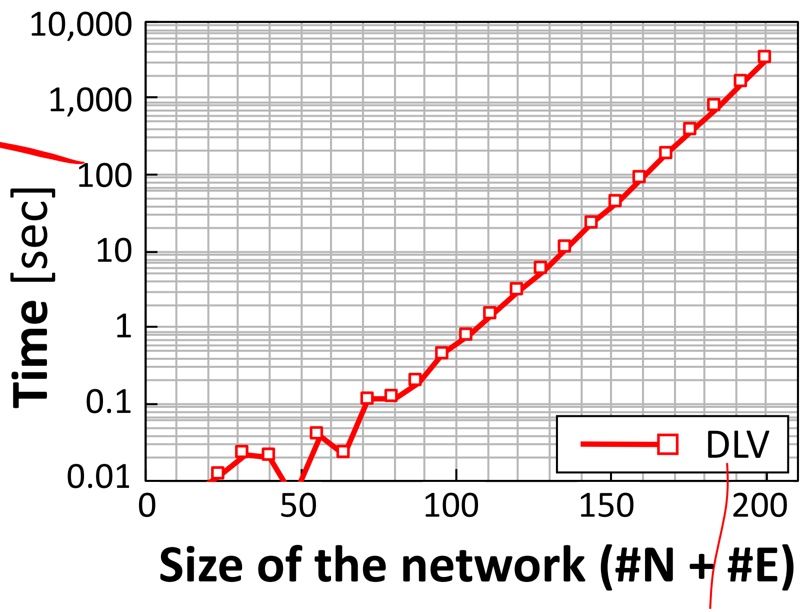

```
   P(C,x) :- P(A,x)
F(C,B,y) :- P(B,y), P(C,x), x≠y
   P(C,y) :- P(B,y), ¬F(C,B,y)
```

- There exist powerful and free LP solver available.

- Previous work on peer data exchange suggest using LPs.

  Greco et al. [TKDE'03]
  Arenas et al. [TLP'03]
  Barcelo, Bertossi [PADL'03]
  Bertossi, Bravo [LPAR'07]

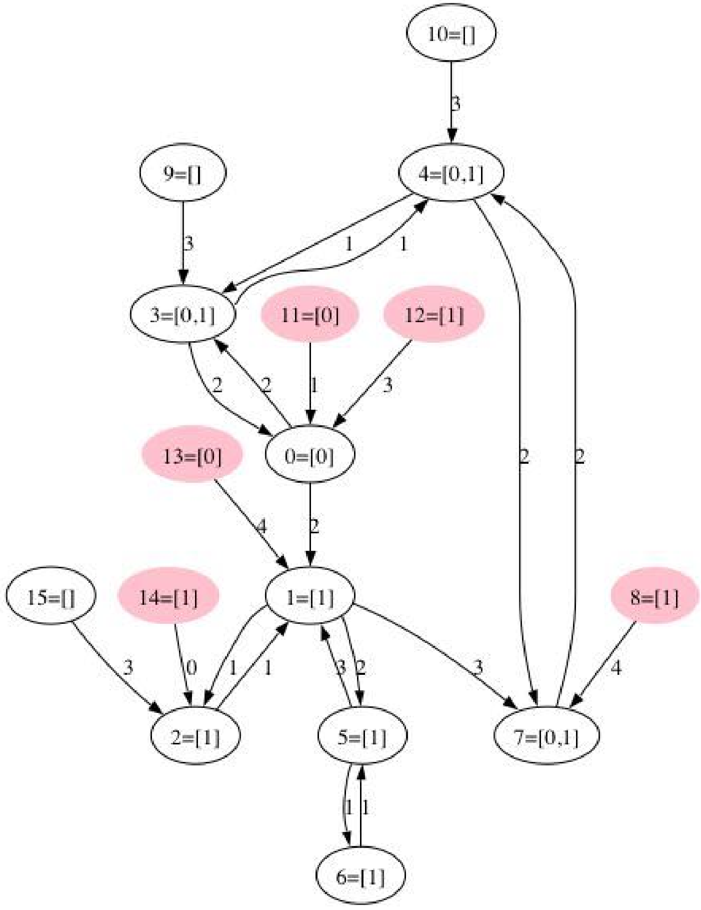
But solving LPs is hard ☹



State-of-the-art LP solver

Yet surprisingly, our problem allows a PTIME solution ☺

DLV example



Size: 38

input.txt

```
% --- Insert explicit beliefs ---
possH(h8_0,1).
possH(h11_0,0).
possH(h12_0,1).
possH(h13_0,0).
possH(h14_0,1).
% --- Node: 0 ---
possH(h0_1,X)       :- possH(h0_0,X).
block(h0_1,11,X)    :- poss(11,X), possH(h0_1,Y), Y!=X.
possH(h0_1,X)       :- poss(11,X), not block(h0_1,11,X).
possH(h0_2,X)       :- possH(h0_1,X).
block(h0_2,3,X)     :- poss(3,X), possH(h0_2,Y), Y!=X.
possH(h0_2,X)       :- poss(3,X), not block(h0_2,3,X).
possH(h0_3,X)       :- possH(h0_2,X).
block(h0_3,12,X)    :- poss(12,X), possH(h0_3,Y), Y!=X.
possH(h0_3,X)       :- poss(12,X), not block(h0_3,12,X).
poss(0,X)           :- possH(h0_3,X).
% --- Node: 1 ---
possH(h1_1,X)       :- possH(h1_0,X).
block(h1_1,2,X)     :- poss(2,X), possH(h1_1,Y), Y!=X.
possH(h1_1,X)       :- poss(2,X), not block(h1_1,2,X).
possH(h1_2,X)       :- possH(h1_1,X).
block(h1_2,0,X)     :- poss(0,X), possH(h1_2,Y), Y!=X.
possH(h1_2,X)       :- poss(0,X), not block(h1_2,0,X).
possH(h1_3,X)       :- possH(h1_2,X).
block(h1_3,5,X)     :- poss(5,X), possH(h1_3,Y), Y!=X.
possH(h1_3,X)       :- poss(5,X), not block(h1_3,5,X).
possH(h1_4,X)       :- possH(h1_3,X).
block(h1_4,13,X)    :- poss(13,X), possH(h1_4,Y), Y!=X.
possH(h1_4,X)       :- poss(13,X), not block(h1_4,13,X).
poss(1,X)           :- possH(h1_4,X).
% --- Node: 2 ---

. . . . . .

% --- Node: 13 ---
poss(13,X)          :- possH(h13_0,X).
% --- Node: 14 ---
poss(14,X)          :- possH(h14_0,X).
% --- Node: 15 ---
poss(15,X)          :- possH(h15_0,X).
```

query.txt

```
poss(X,U) ?
```

Executing program

```
./dlv.bin – brave
    input.txt. query-.txt
```

Result

```
Macintosh-2:DLV gat!
8, 1
11, 0
12, 1
13, 0
14, 1
0, 0
1, 1
2, 1
3, 0
3, 1
4, 0
4, 1
5, 1
6, 1
7, 0
7, 1
```

Agenda

1. Stable solutions
   – how to define a unique and consistent solution?

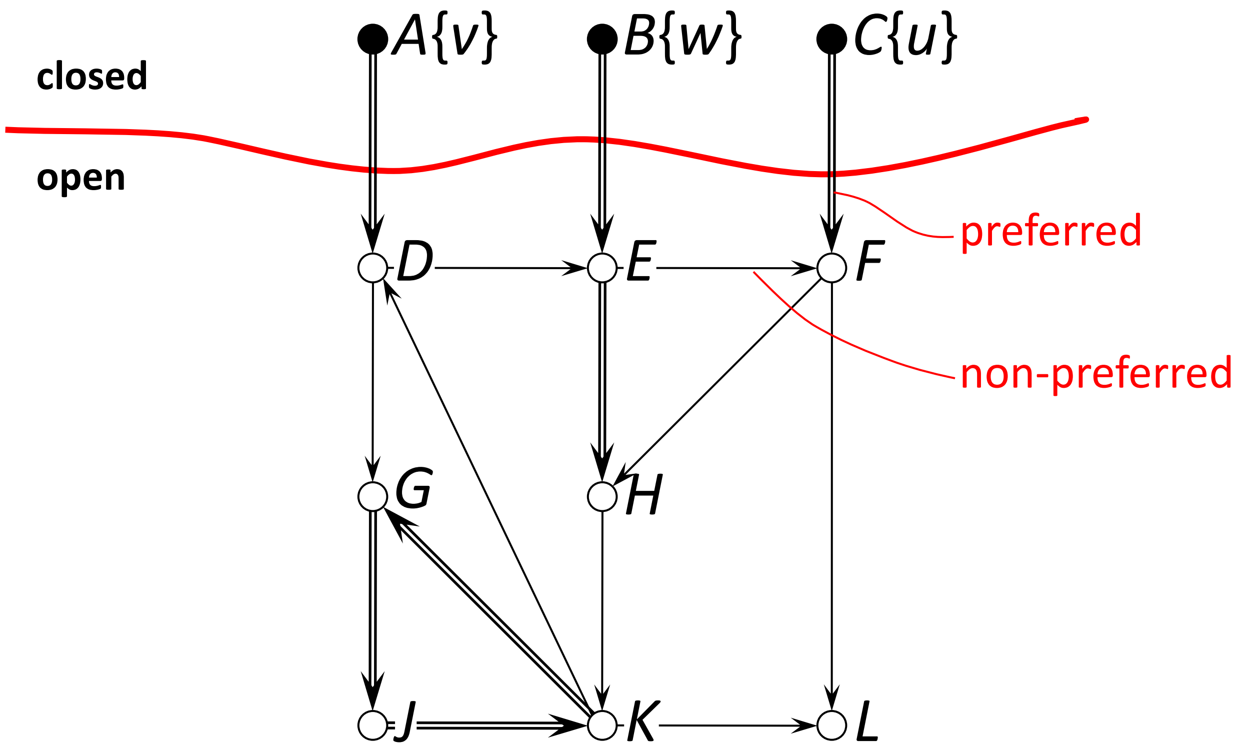
2. Resolution algorithm
   – how to calculate the solution efficiently?

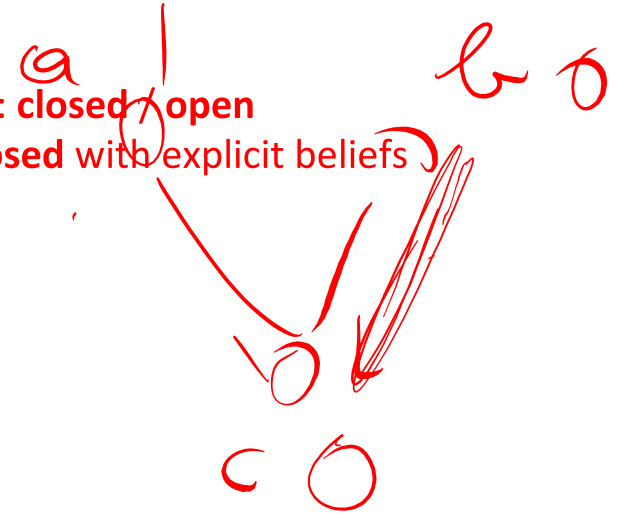
3. Extensions
   – how to deal with “negative beliefs”?

Resolution Algorithm

Focus on binary trust network

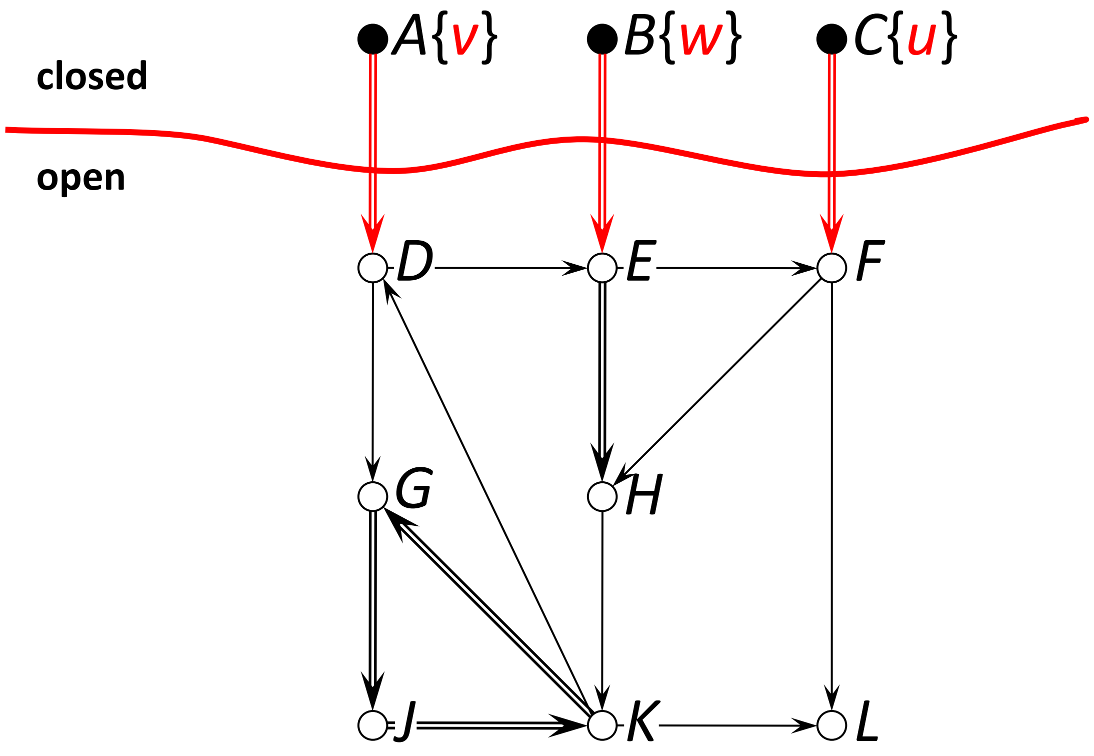
- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs





X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	?	?
E	?	?
F	?	?
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

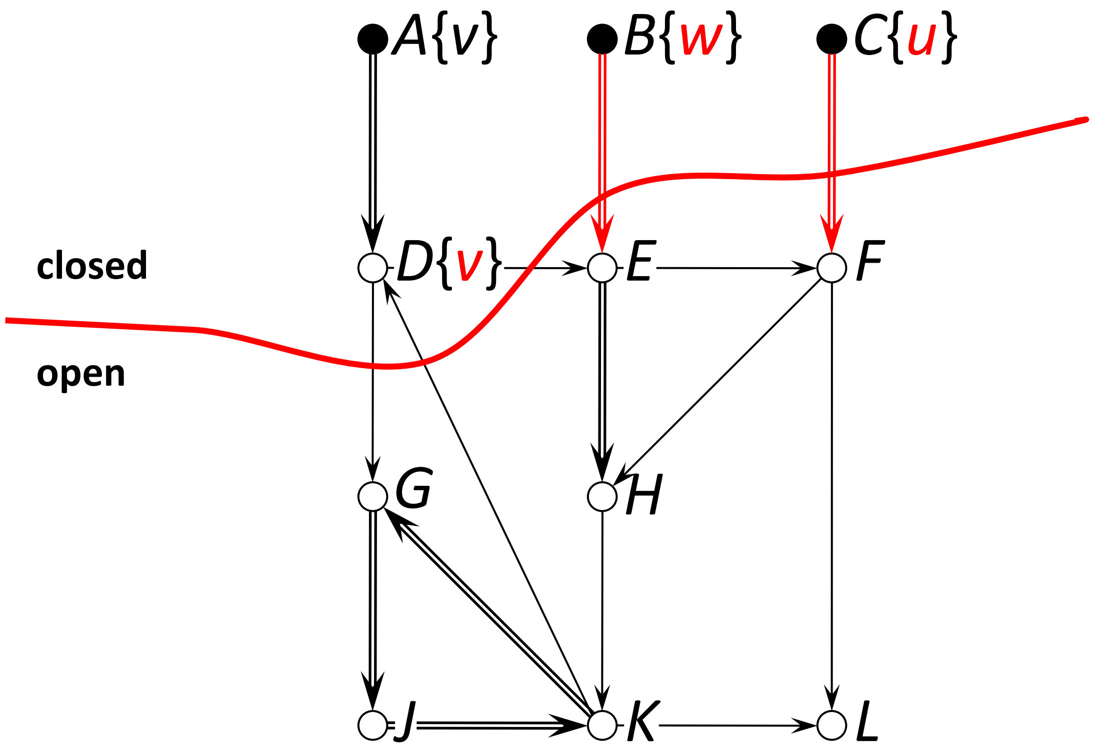
Resolution Algorithm



- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	?	?
E	?	?
F	?	?
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

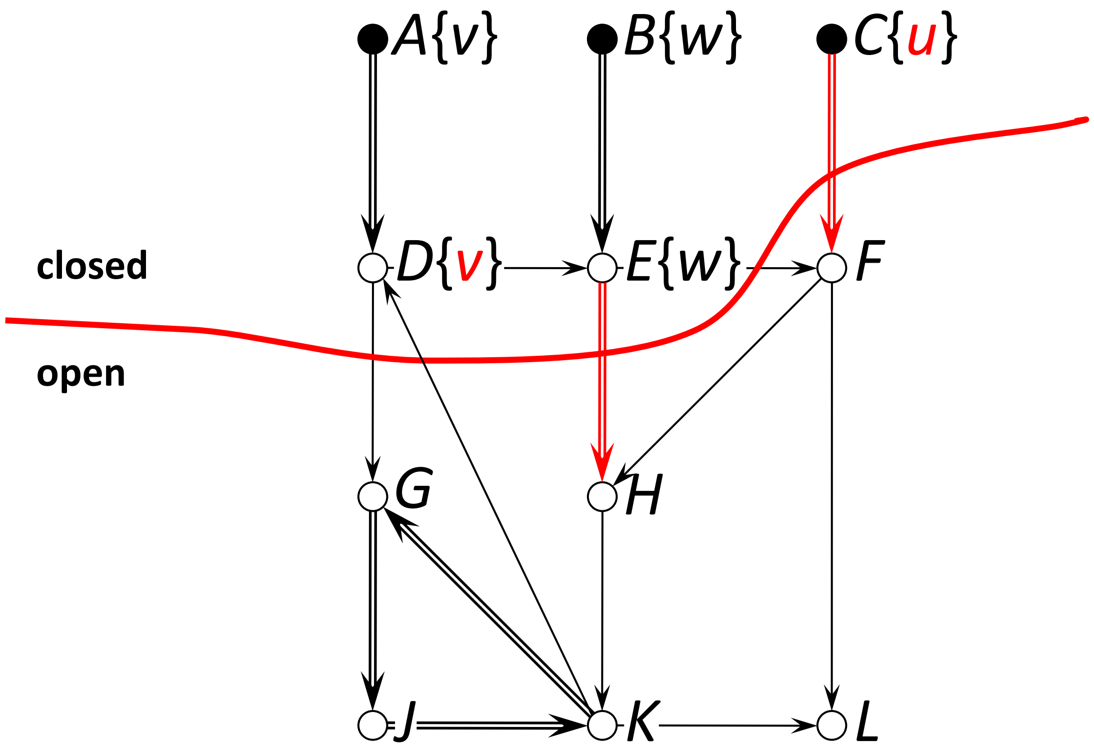
Resolution Algorithm



- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$\{v\}$	$\{v\}$
E	?	?
F	?	?
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

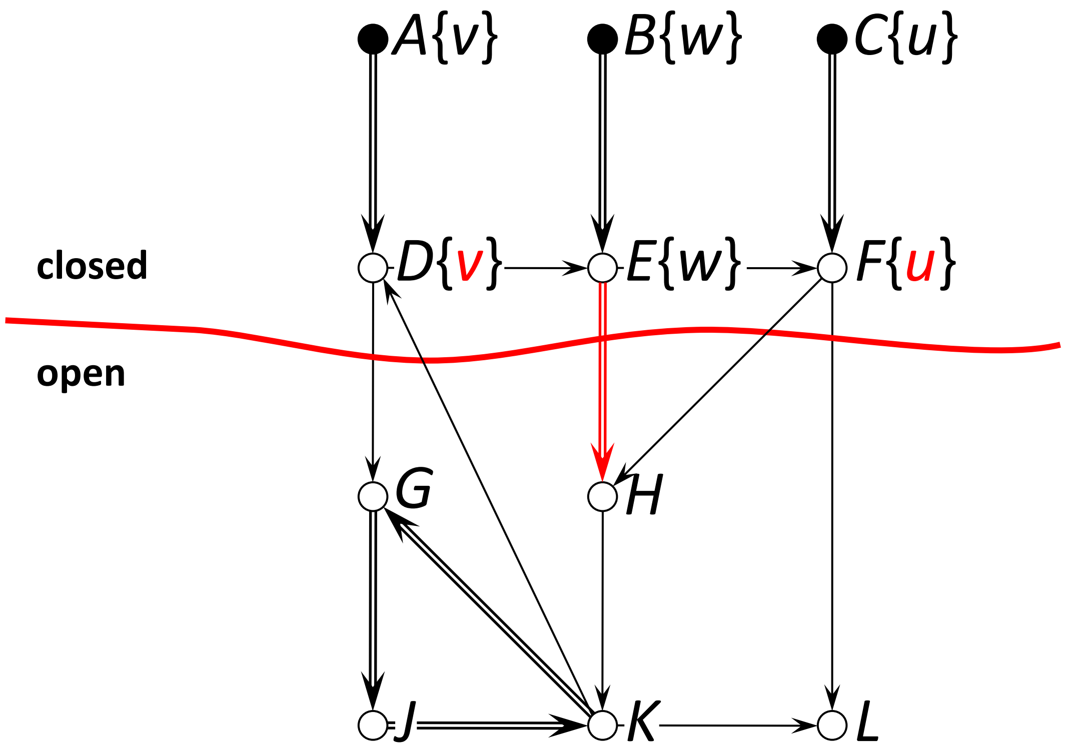
Resolution Algorithm



- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from **open** to **closed**
  → follow

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$\{v\}$	$\{v\}$
E	$\{w\}$	$\{w\}$
F	?	?
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

Resolution Algorithm

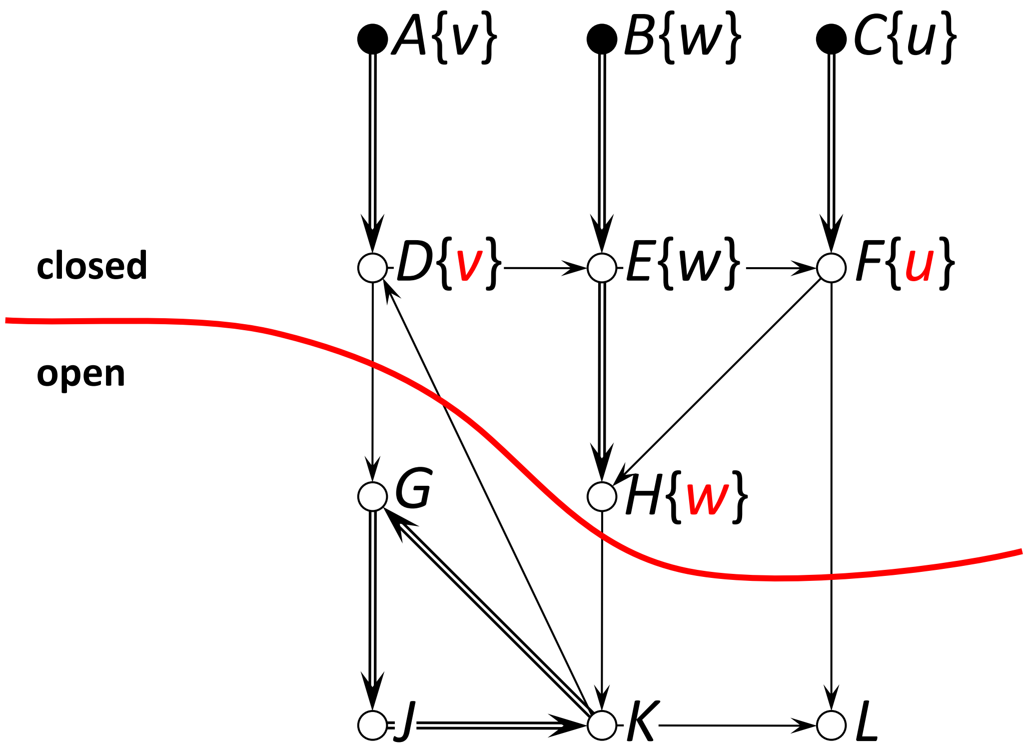


- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow

X	**poss**(X)	**cert**(X)
A	{v}	{v}
B	{w}	{w}
C	{u}	{u}
D	{v}	{v}
E	{w}	{w}
F	{u}	{u}
G	?	?
H	?	?
J	?	?
K	?	?
L	?	?

Resolution Algorithm

- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from **open** to **closed**
  → follow

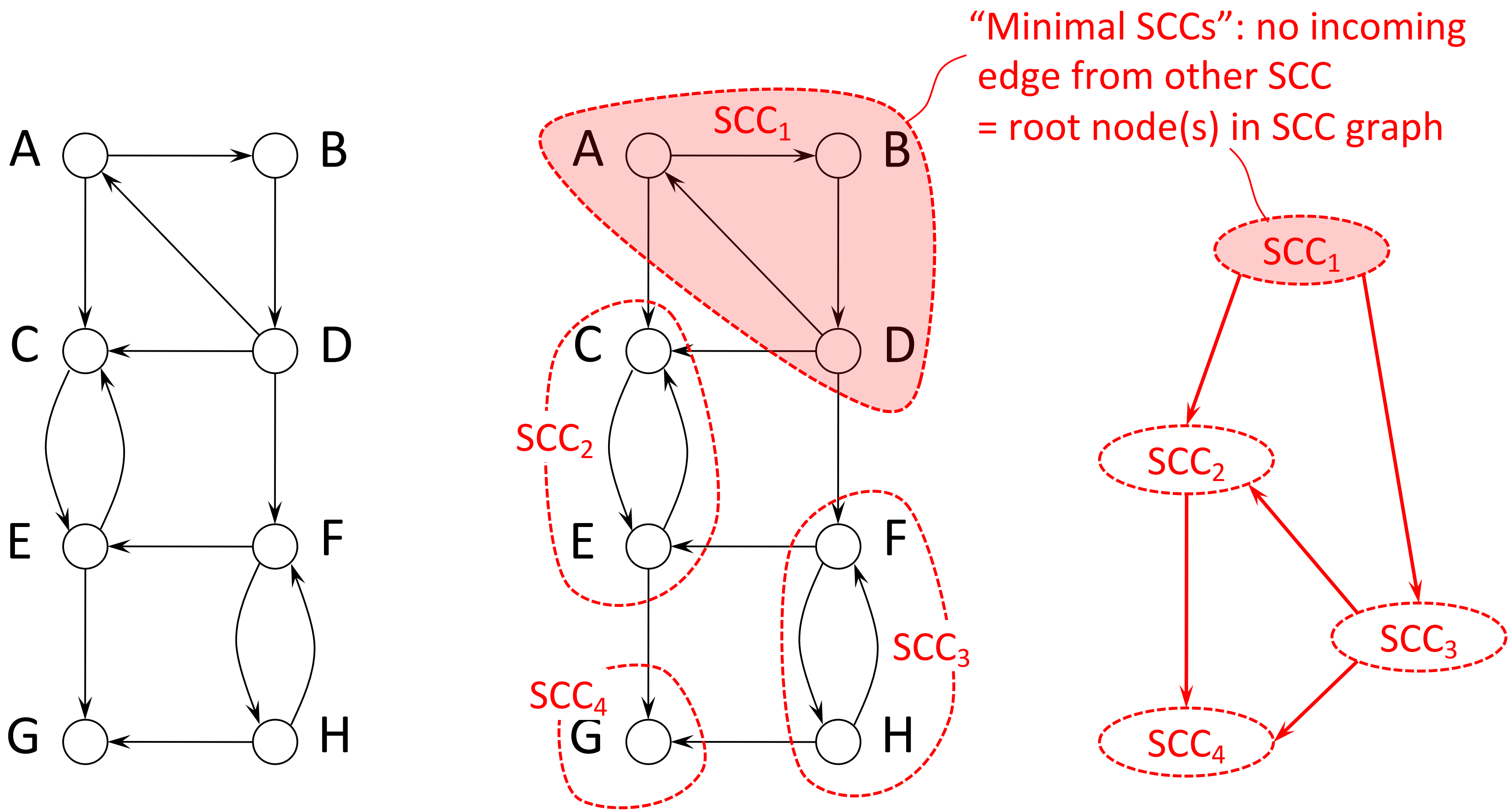


X	**poss**(X)	**cert**(X)
A	{v}	{v}
B	{w}	{w}
C	{u}	{u}
D	{v}	{v}
E	{w}	{w}
F	{u}	{u}
G	?	?
H	{w}	{w}
J	?	?
K	?	?
L	?	?

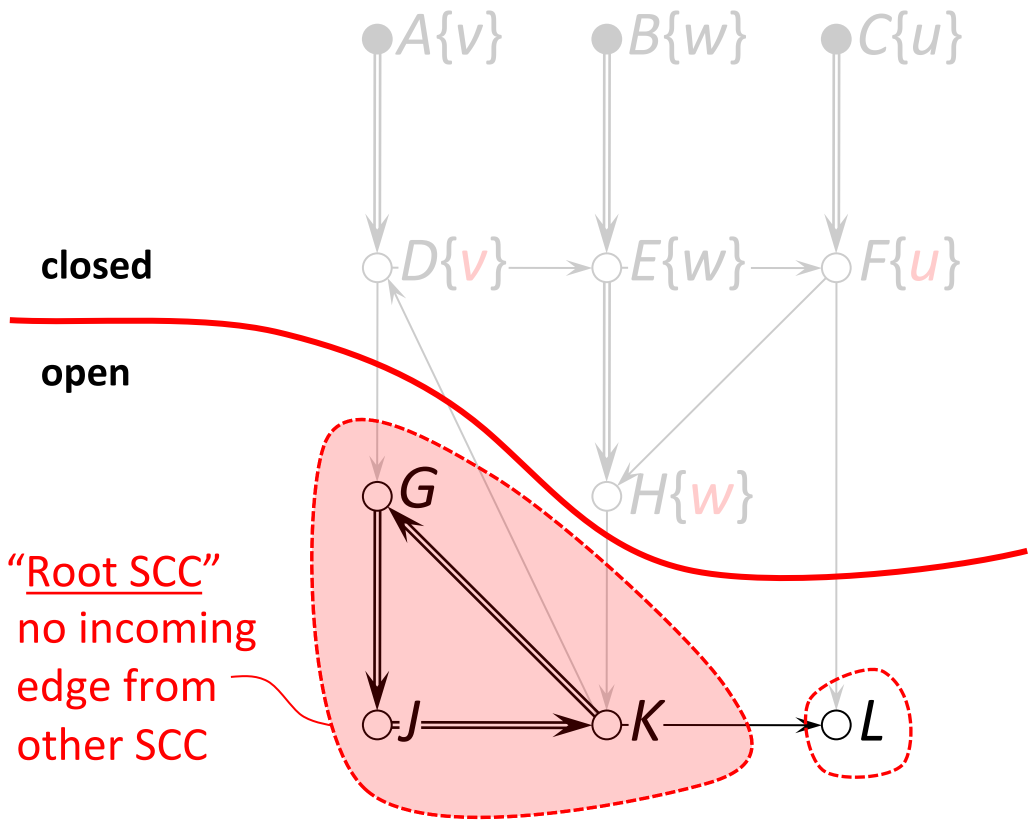
Now we are stuck!

Detail: Strongly Connected Components (SCCs)

For every cyclic or acyclic directed graph:
- The Strongly Connected Components graph is a DAG
- can be calculated in **O(*n*)** Tarjan [1972]



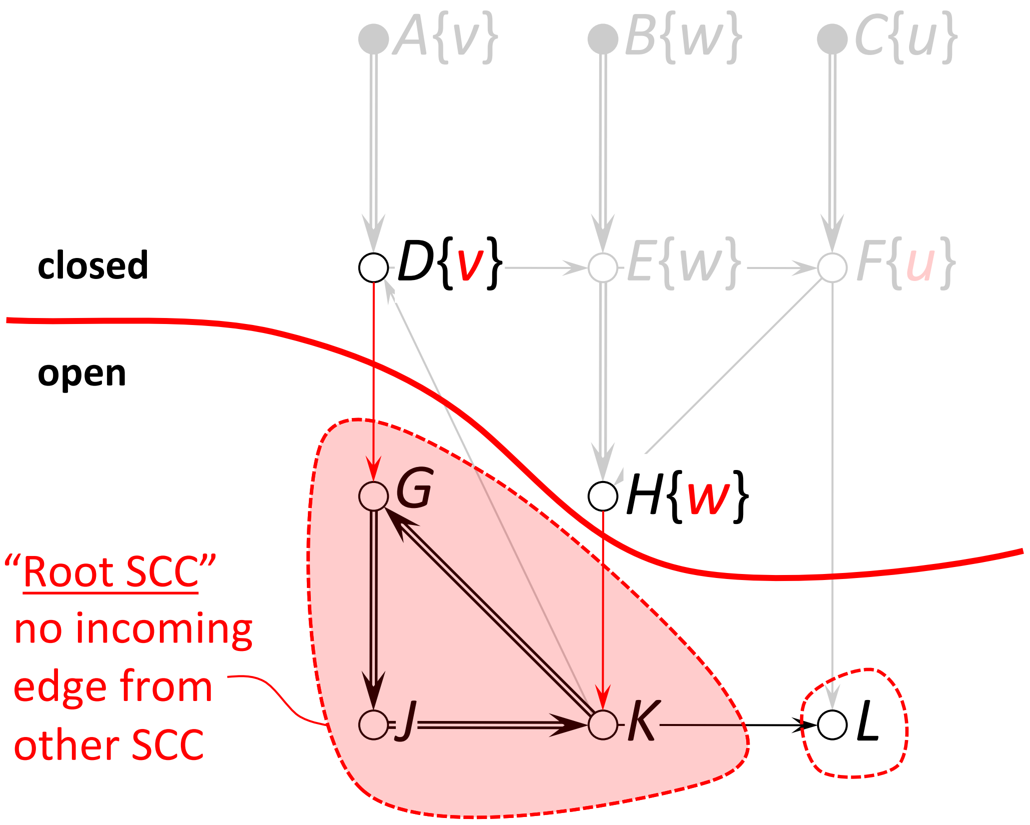
Resolution Algorithm



- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow
  Step 2: else
  → construct SCC graph of **open**

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$\{v\}$	$\{v\}$
E	$\{w\}$	$\{w\}$
F	$\{u\}$	$\{u\}$
G	?	?
H	$\{w\}$	$\{w\}$
J	?	?
K	?	?
L	?	?

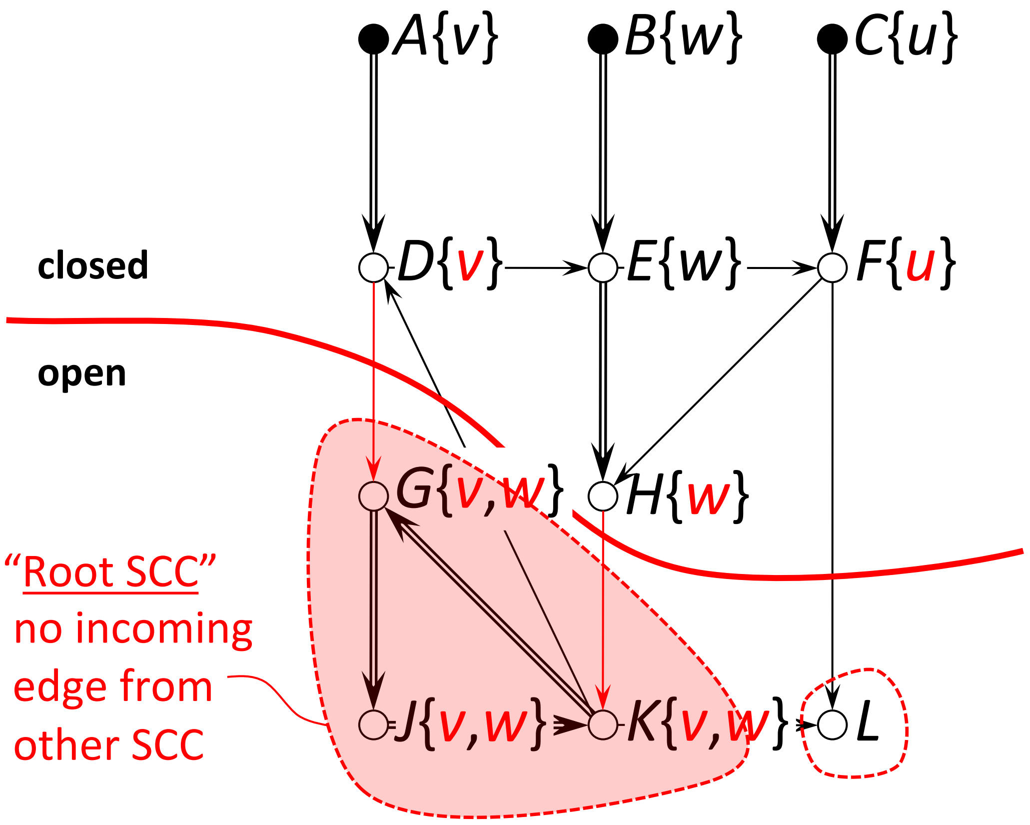
Resolution Algorithm



- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow
  Step 2: else
  → construct SCC graph of **open**

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$\{v\}$	$\{v\}$
E	$\{w\}$	$\{w\}$
F	$\{u\}$	$\{u\}$
G	?	?
H	$\{w\}$	$\{w\}$
J	?	?
K	?	?
L	?	?

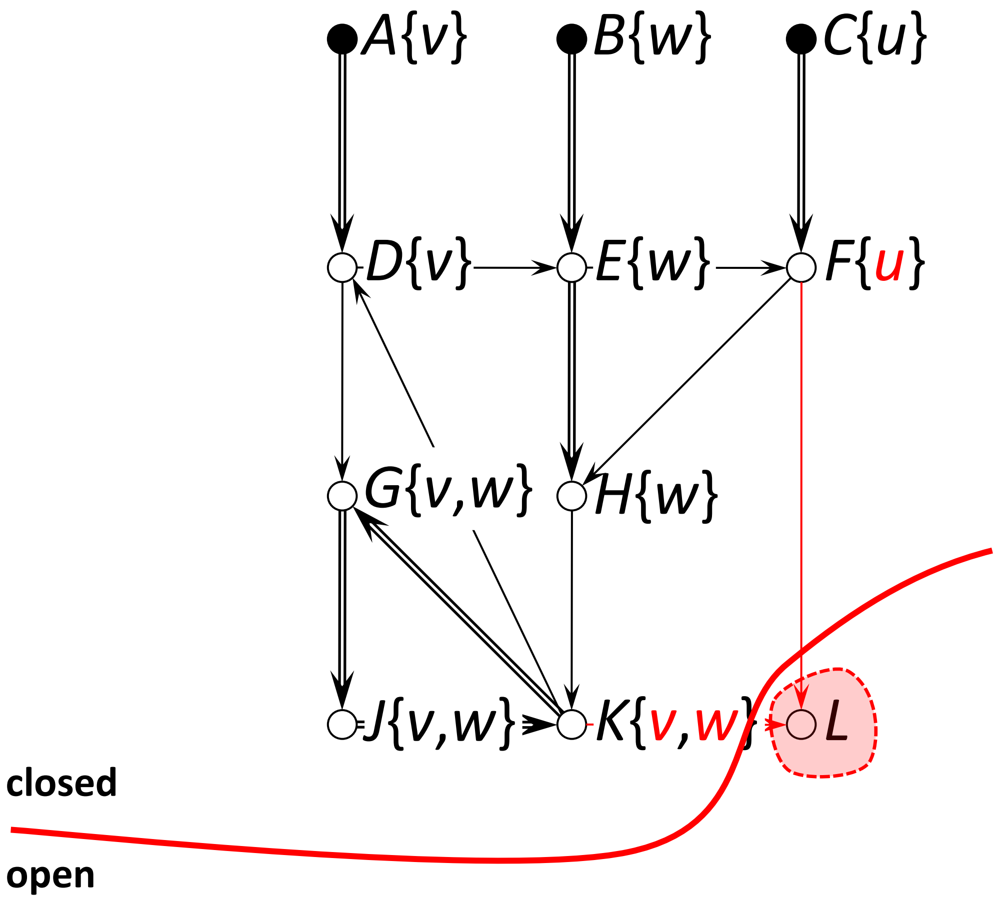
Resolution Algorithm



- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow
  Step 2: else
  → construct SCC graph of **open**
  → resolve minimum SCCs

X	**poss**(X)	**cert**(X)
A	$\{v\}$	$\{v\}$
B	$\{w\}$	$\{w\}$
C	$\{u\}$	$\{u\}$
D	$\{v\}$	$\{v\}$
E	$\{w\}$	$\{w\}$
F	$\{u\}$	$\{u\}$
G	$\{v,w\}$	$\varnothing$
H	$\{w\}$	$\{w\}$
J	$\{v,w\}$	$\varnothing$
K	$\{v,w\}$	$\varnothing$
L	?	?

Resolution Algorithm

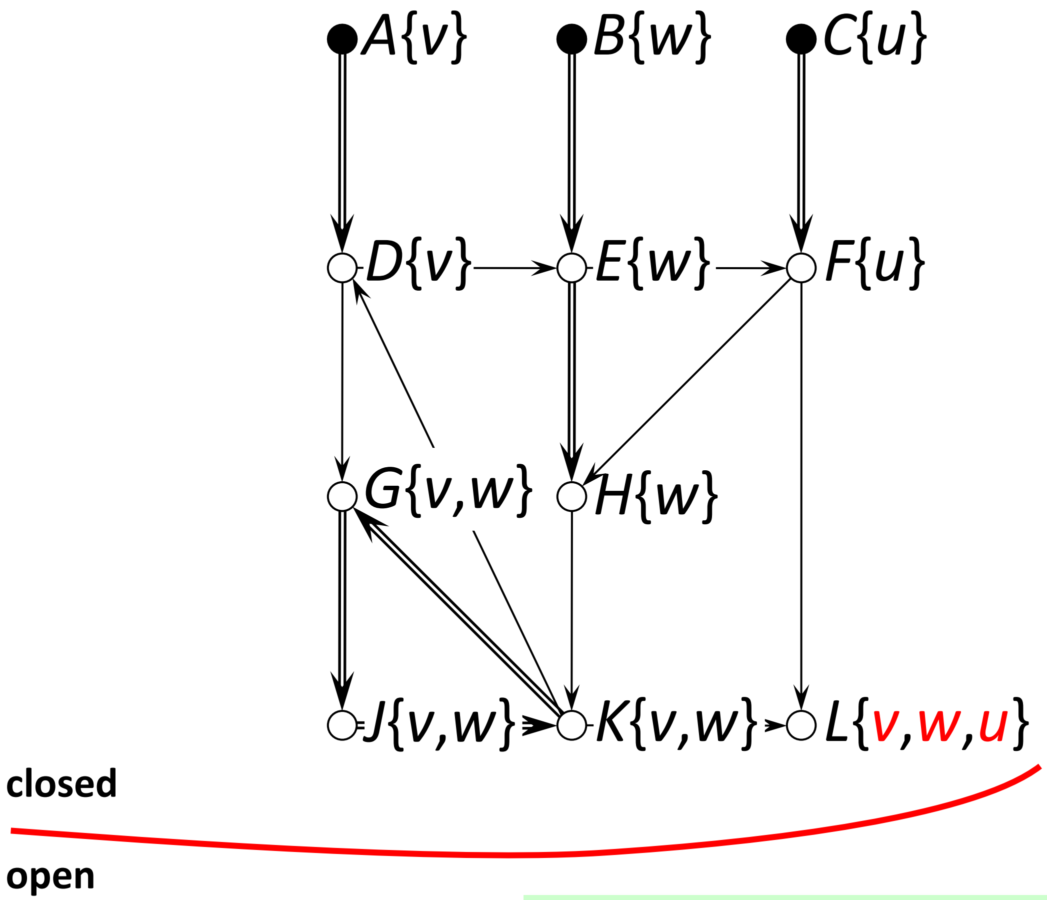


- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  Step 1: if ∃ preferred edges from
  open to **closed**
  → follow
  Step 2: else
  → construct SCC graph of **open**
  → resolve minimum SCCs

X	**poss**(X)	**cert**(X)
A	{v}	{v}
B	{w}	{w}
C	{u}	{u}
D	{v}	{v}
E	{w}	{w}
F	{u}	{u}
G	{v,w}	∅
H	{w}	{w}
J	{v,w}	∅
K	{v,w}	∅
L	?	?

Resolution Algorithm

- Keep 2 sets: **closed** / **open**
  Initialize **closed** with explicit beliefs
- MAIN
  - <u>Step 1</u>: if ∃ preferred edges from
    open to **closed**
    → follow
  - <u>Step 2</u>: else
    → construct SCC graph of **open**
    → resolve minimum SCCs



X	**poss**(*X*)	**cert**(*X*)
A	{*v*}	{*v*}
B	{*w*}	{*w*}
C	{*u*}	{*u*}
D	{*v*}	{*v*}
E	{*w*}	{*w*}
F	{*u*}	{*u*}
G	{*v*,*w*}	∅
H	{*w*}	{*w*}
J	{*v*,*w*}	∅
K	{*v*,*w*}	∅
L	{*v*,*w*,*u*}	∅

Can be implemented in current DBMS with transitive closure ☺

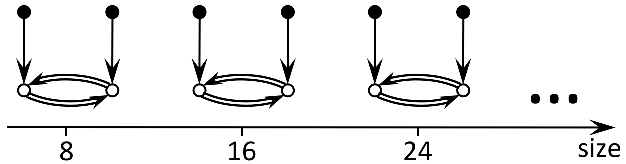
PTIME resolution algorithm
O(*n*²) worst case
O(*n*) on reasonable graphs

Experiments on large network data

Calculating **poss** / **cert** for fixed key

- **DLV**: State-of-the art logic programming solver
- **RA**: Resolution algorithm

Network 1: "Oscillators"

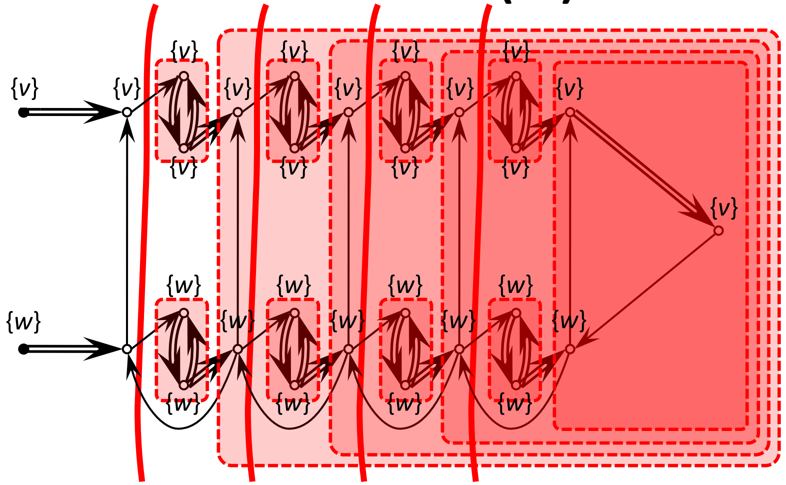


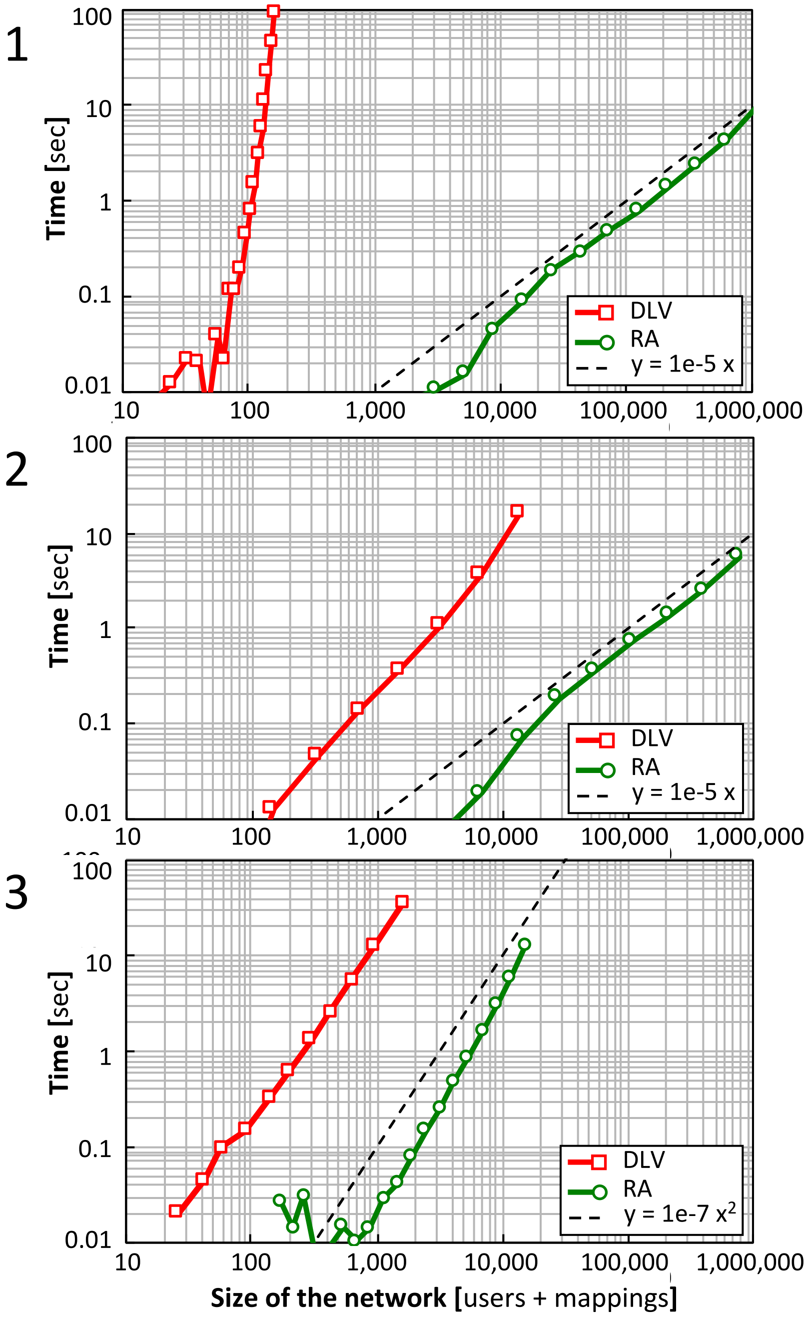
Network 2: "Web link data"

Web data set with 5.4m links between 270k domain names. Approach:

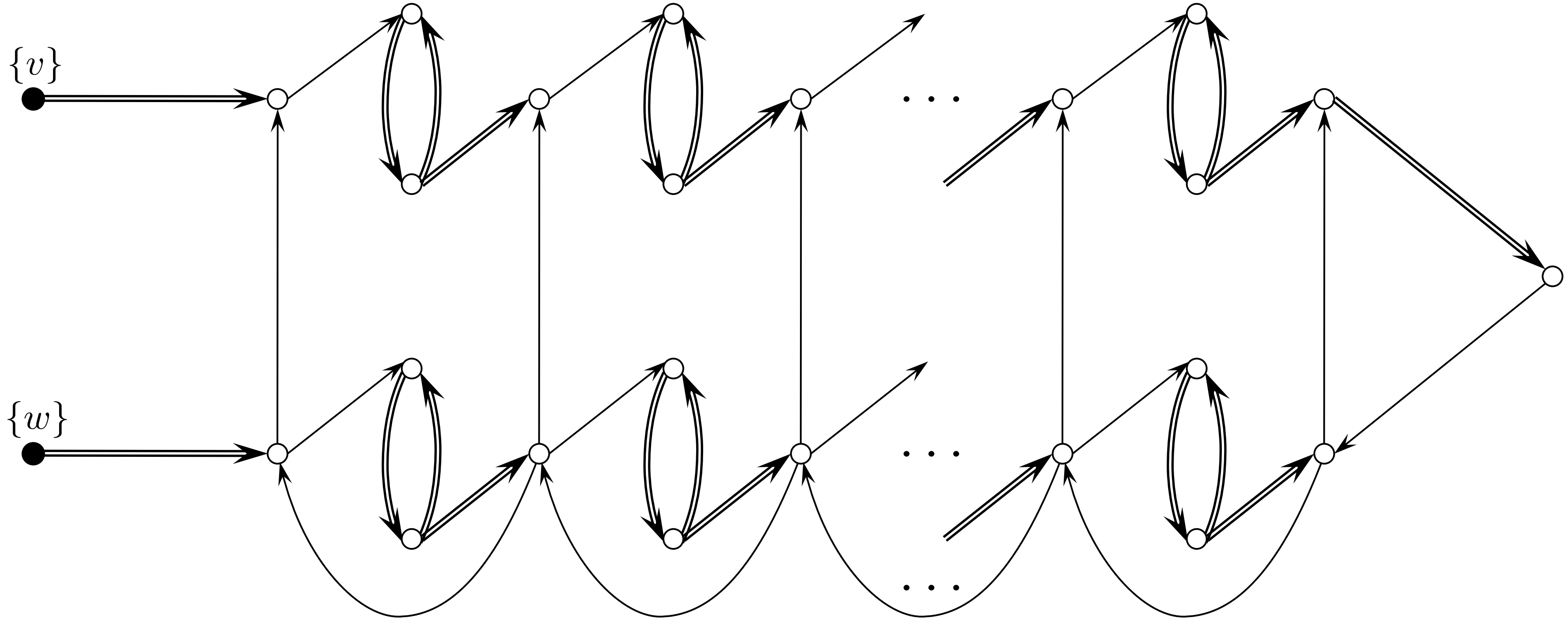
- Sample links with increasing ratio
- Include both nodes in sample
- Assign explicit beliefs randomly

Network 3: "Worst case" **O(n^2)**

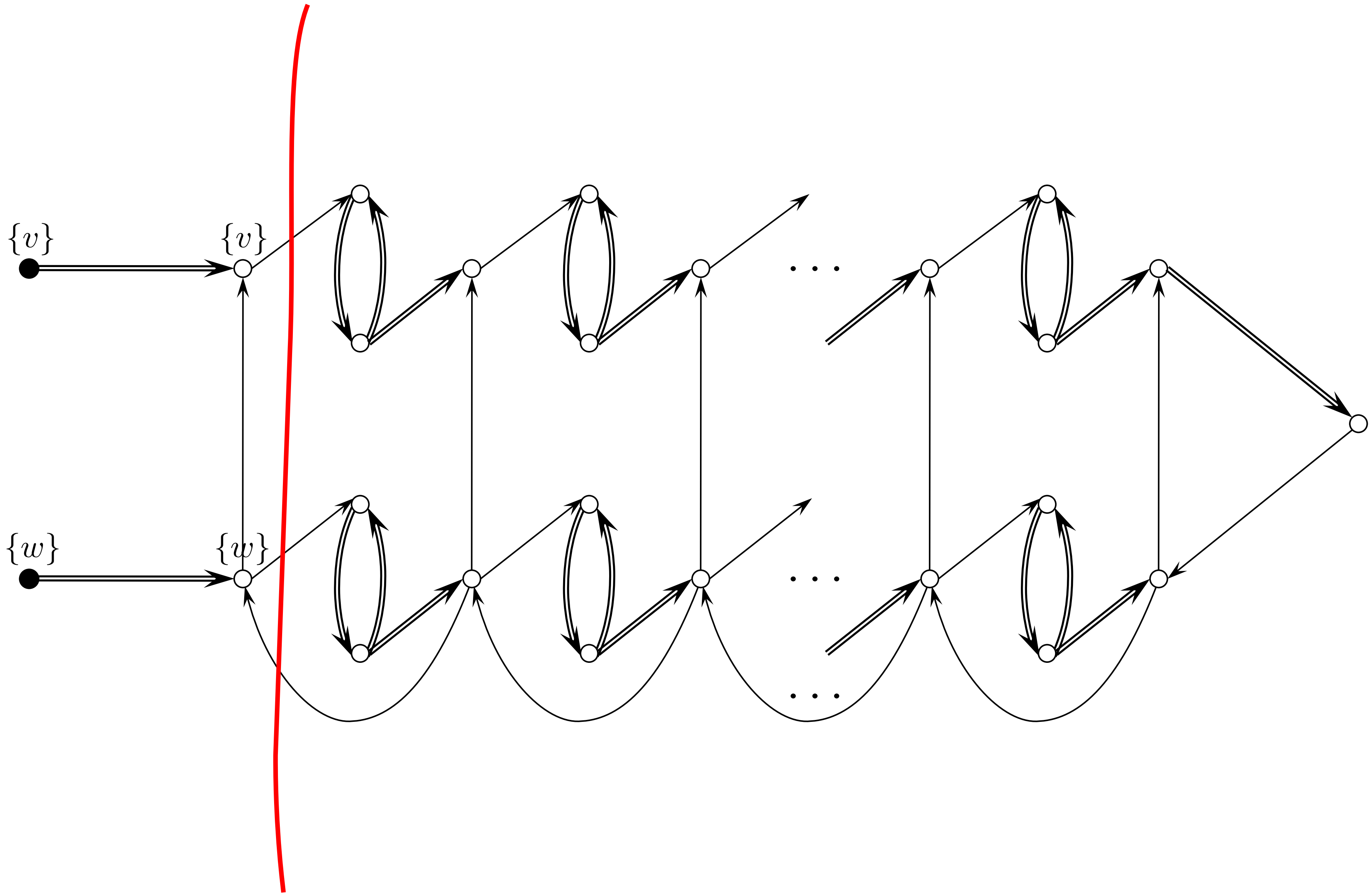




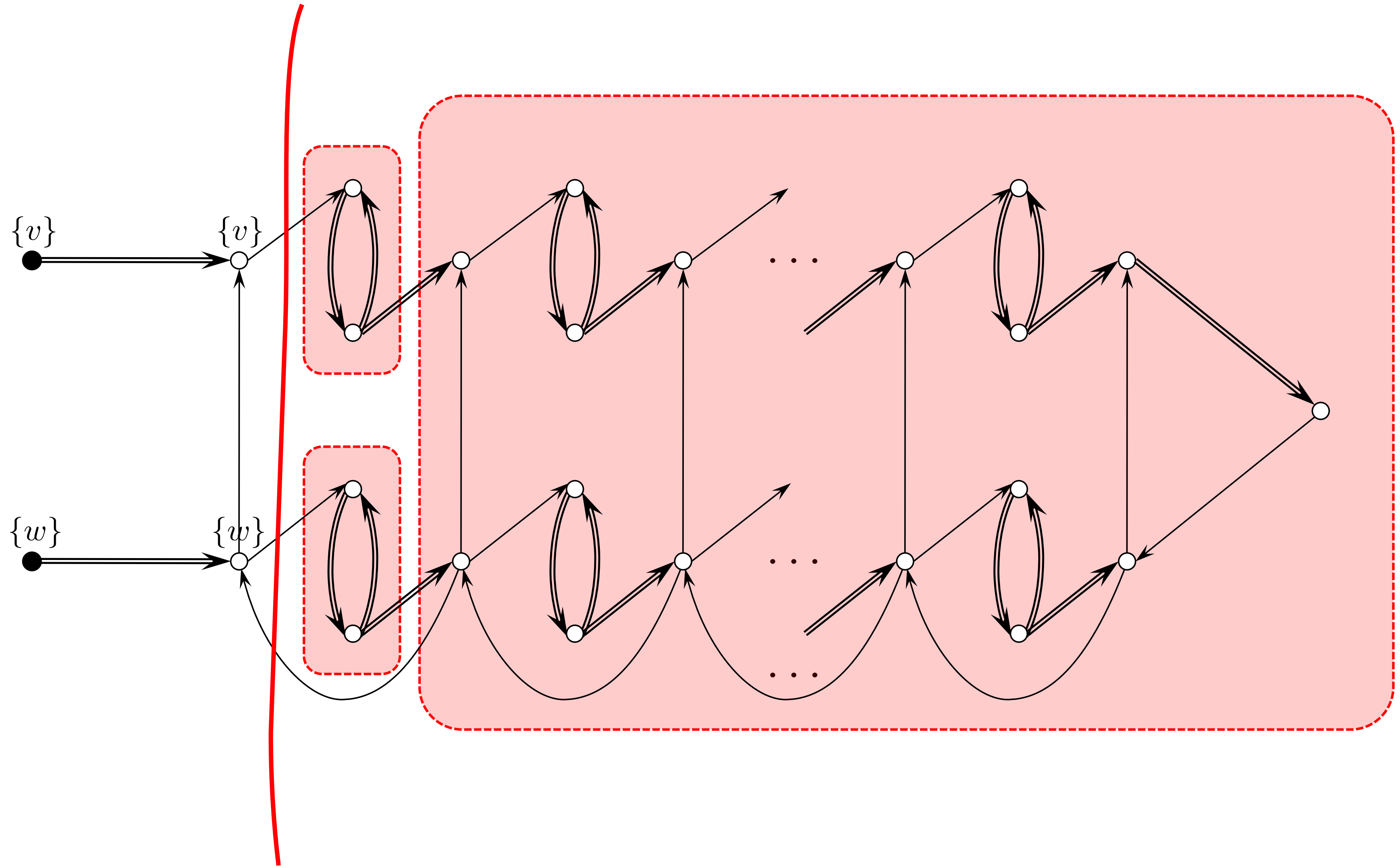
O(n2)-worst-case for Resolution Algorithm



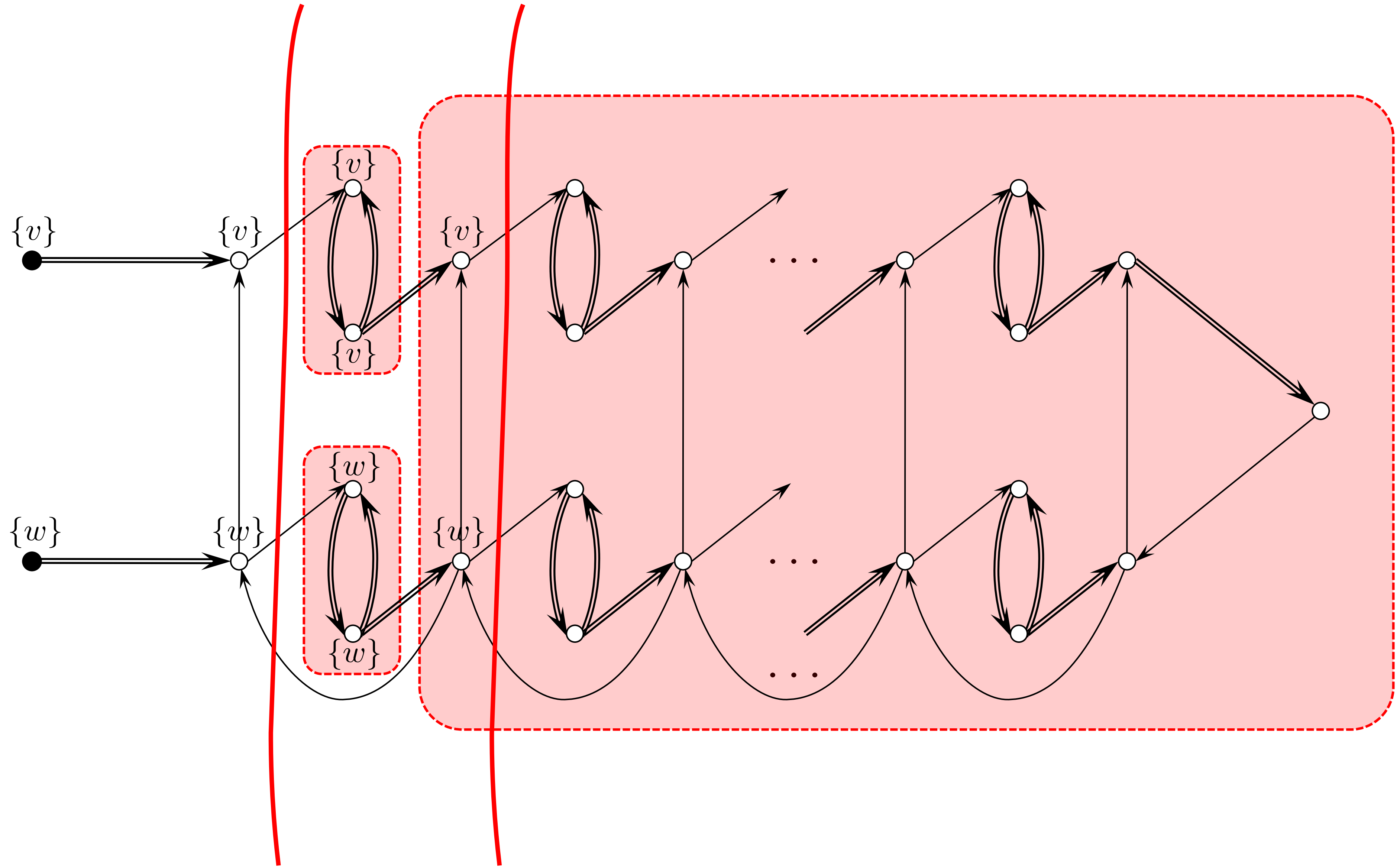
O(n2)-worst-case for Resolution Algorithm



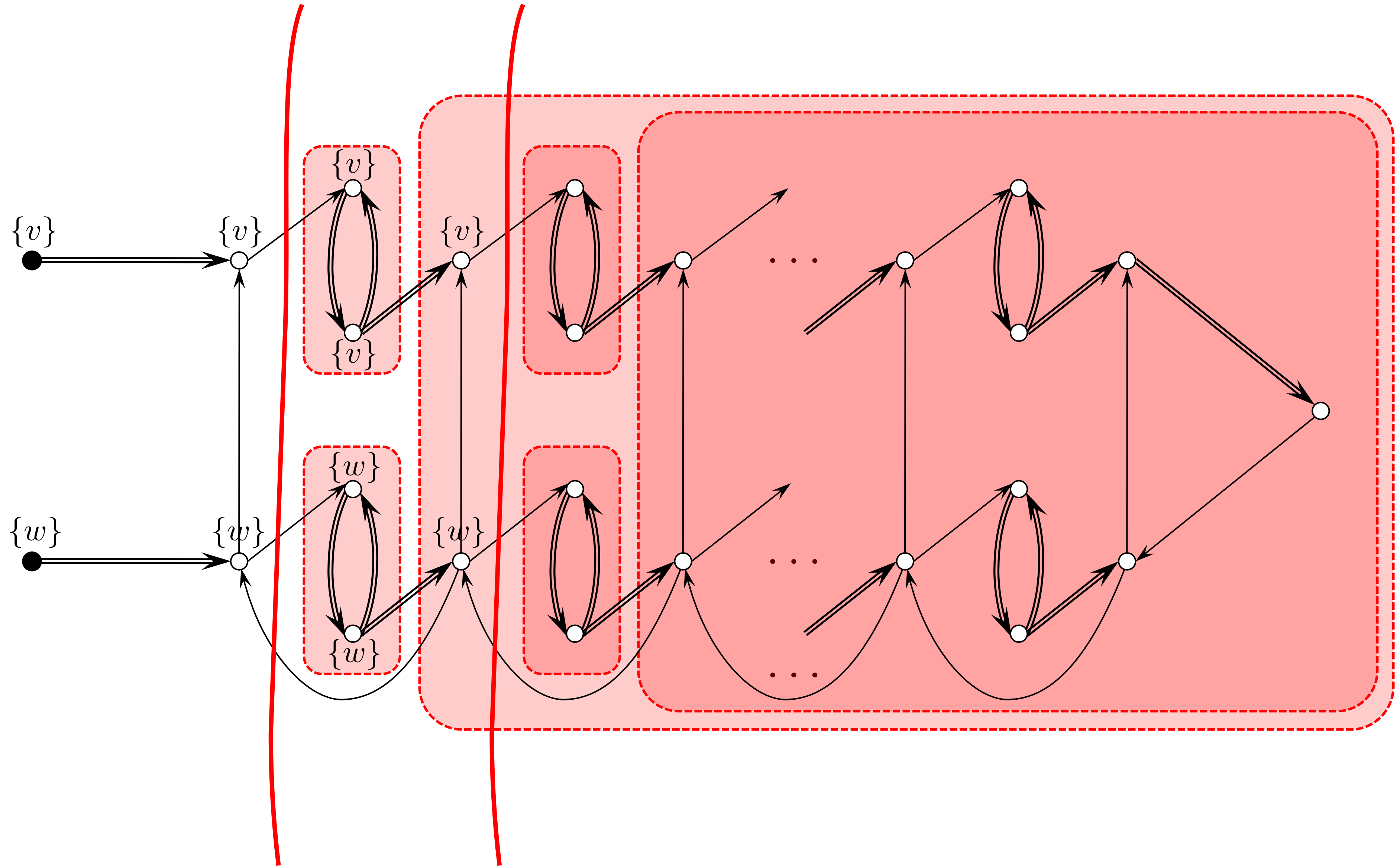
O(n2)-worst-case for Resolution Algorithm



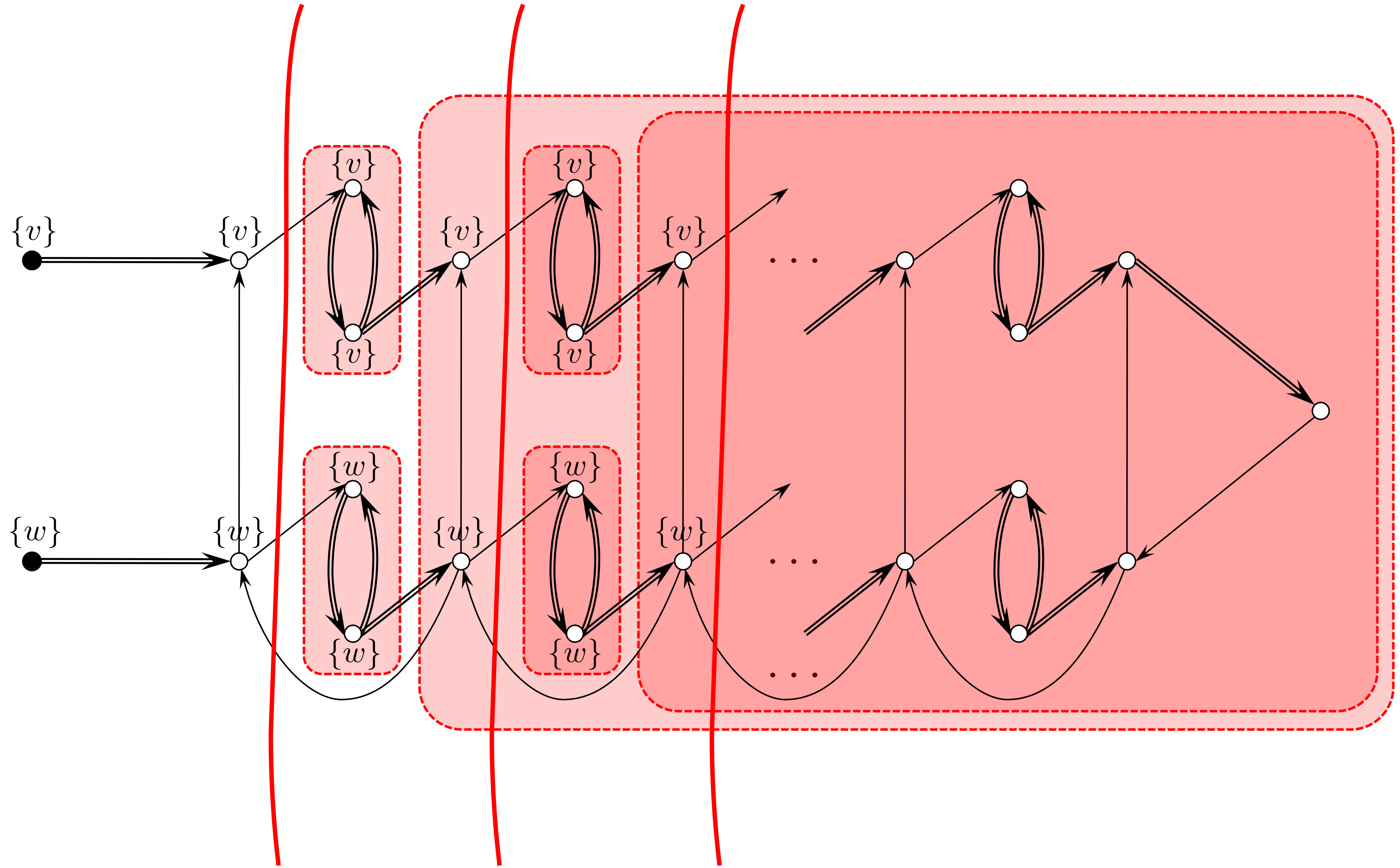
O(n2)-worst-case for Resolution Algorithm



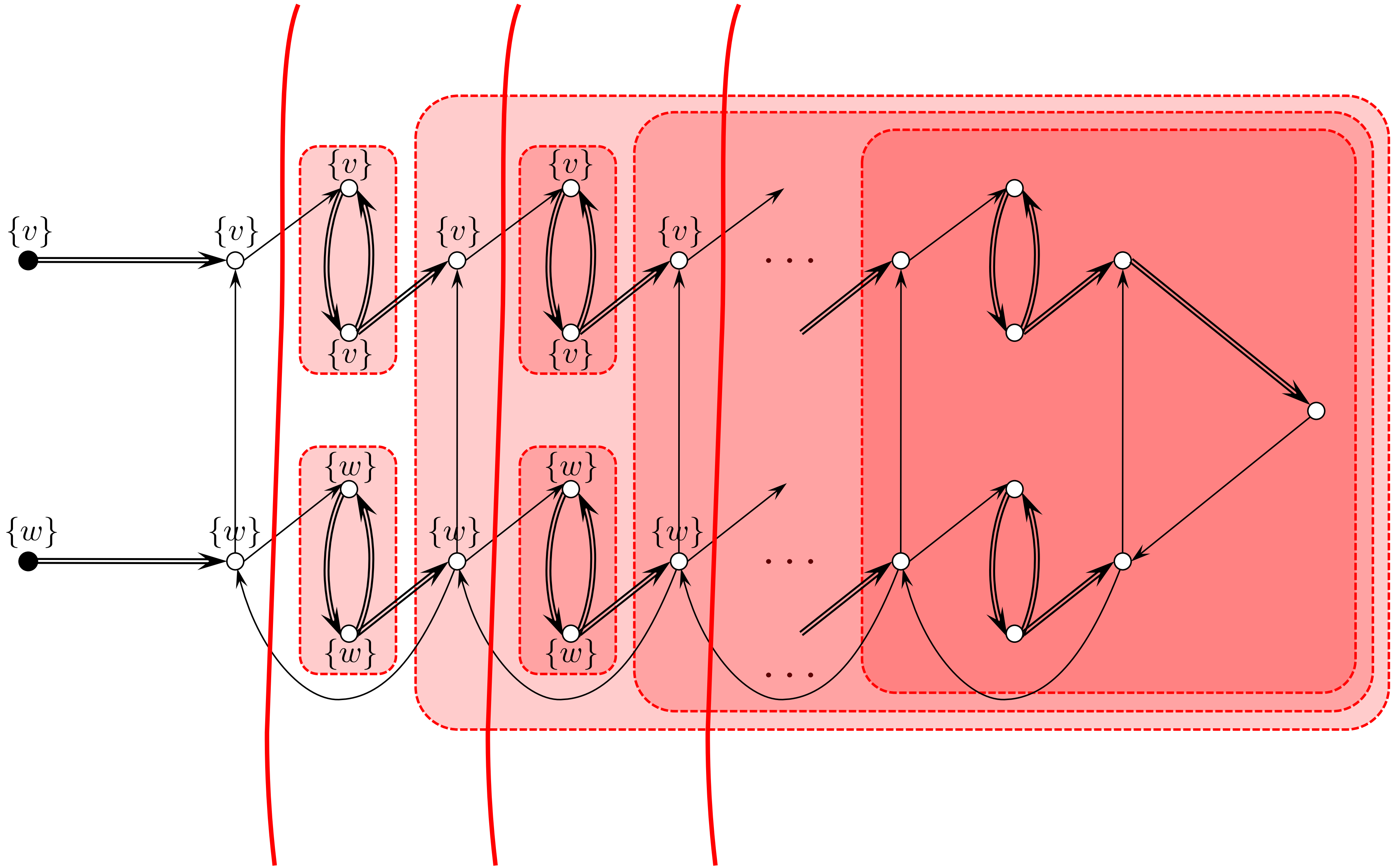
O(n2)-worst-case for Resolution Algorithm



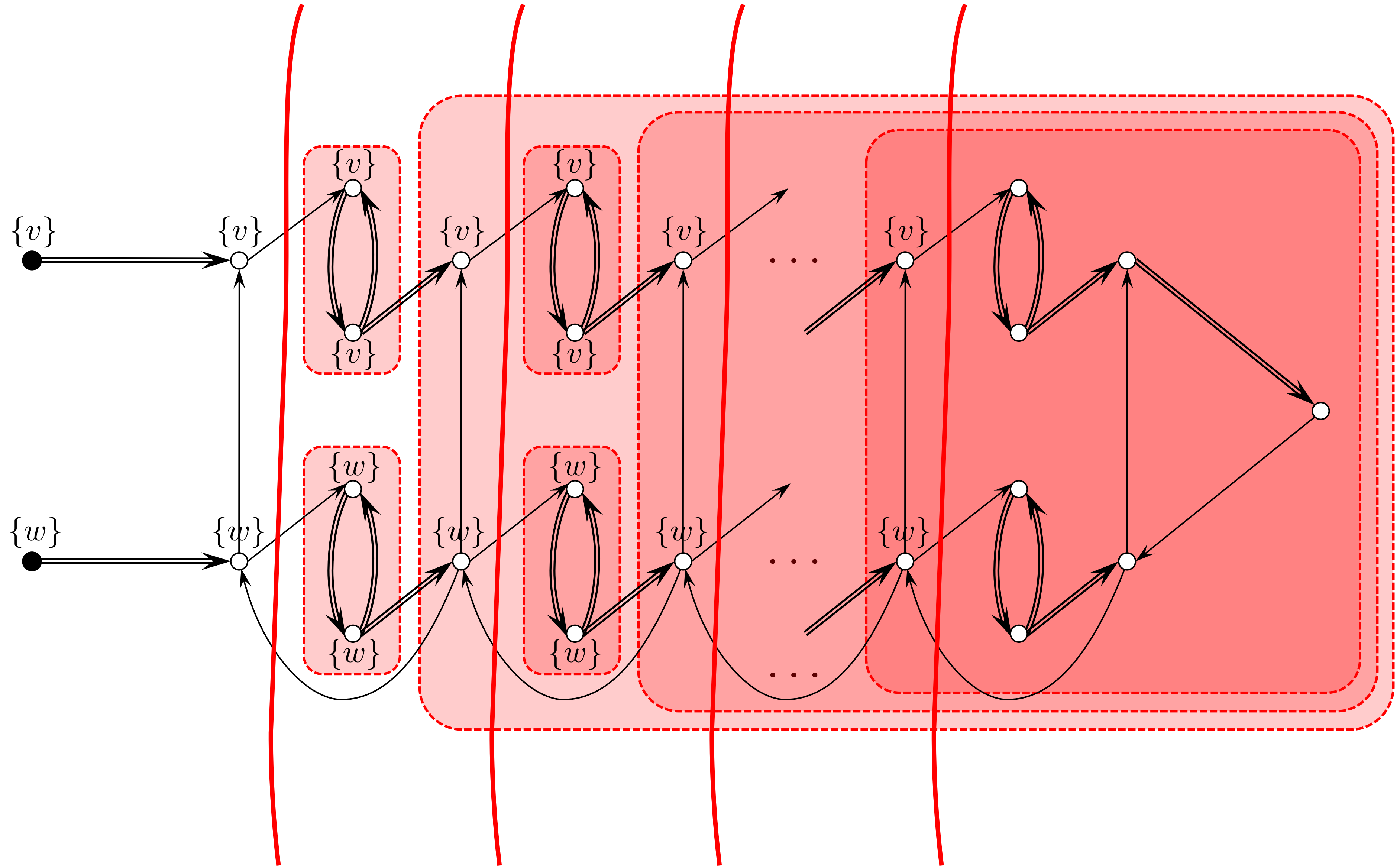
O(n2)-worst-case for Resolution Algorithm



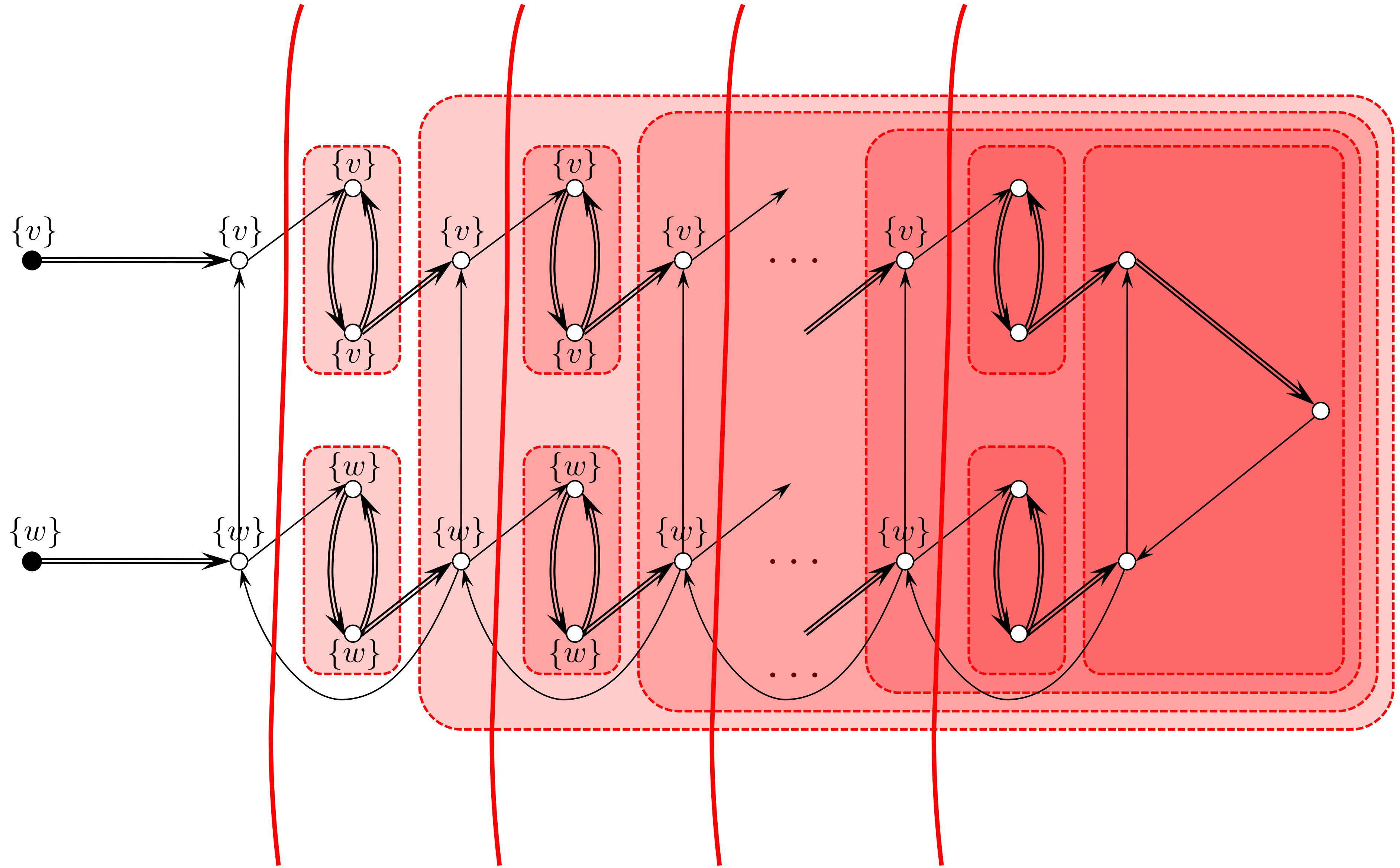
O(n2)-worst-case for Resolution Algorithm



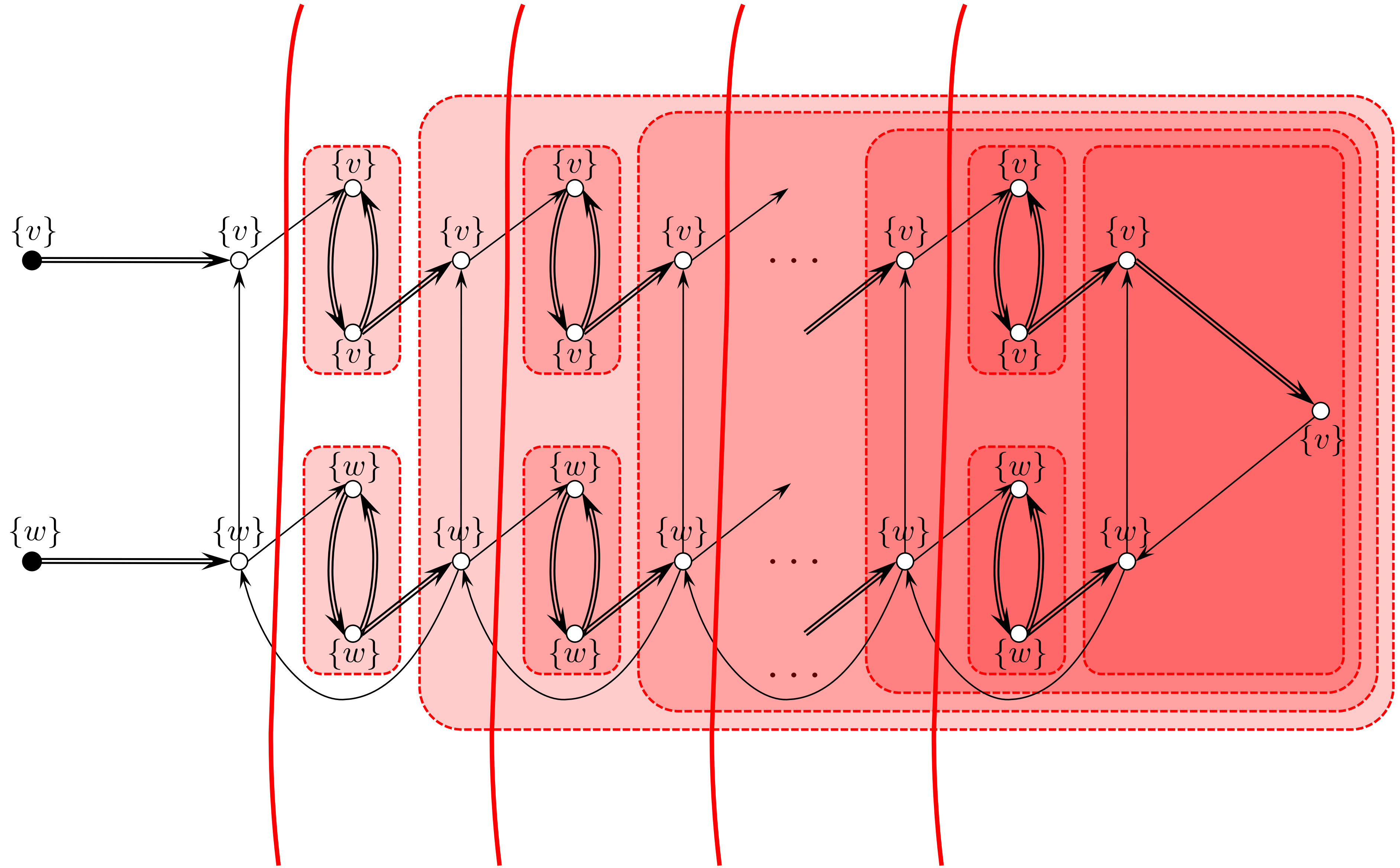
O(n2)-worst-case for Resolution Algorithm



O(n2)-worst-case for Resolution Algorithm



O(n2)-worst-case for Resolution Algorithm



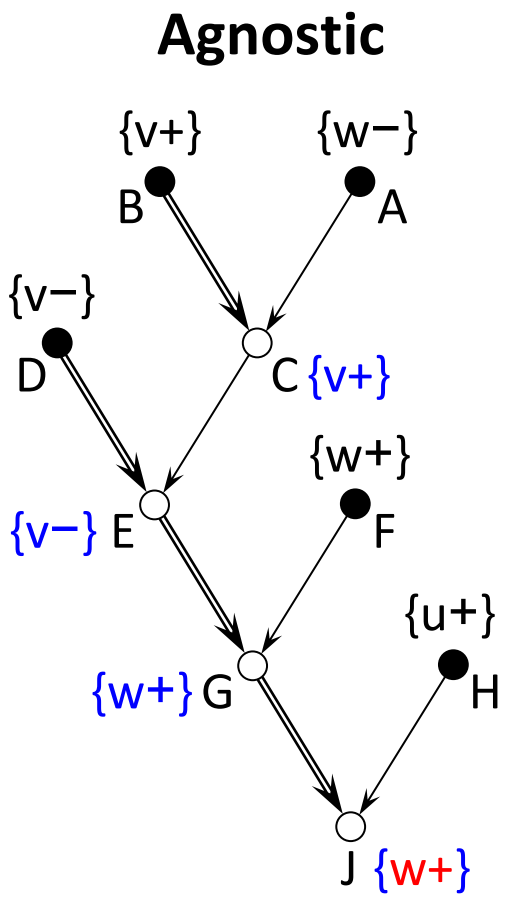
Agenda

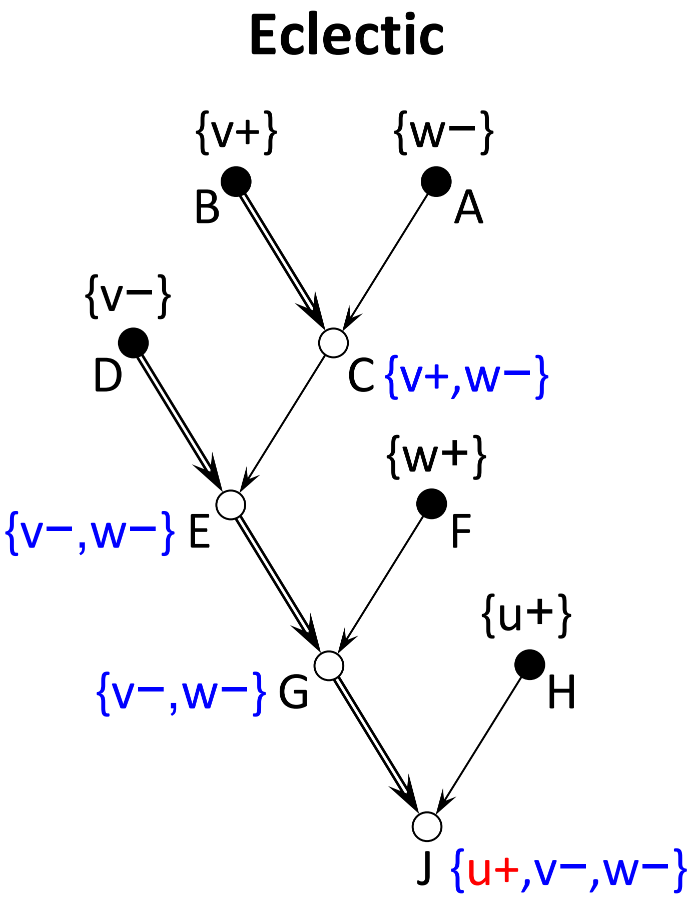
1. Stable solutions
   – how to define a unique and consistent solution?

2. Resolution algorithm
   – how to calculate the solution efficiently?

3. Extensions
   – how to deal with “negative beliefs”?

3 semantics for negative beliefs





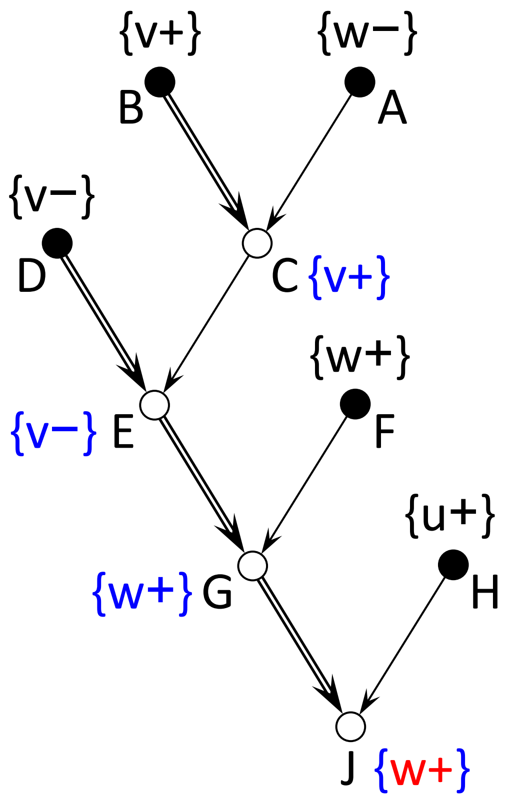
	Agnostic	Eclectic
w/o cycles*	**O(*n*)**	**O(*n*)**
w cycles	**NP-hard**	**NP-hard**

* assuming total order on parents for each node

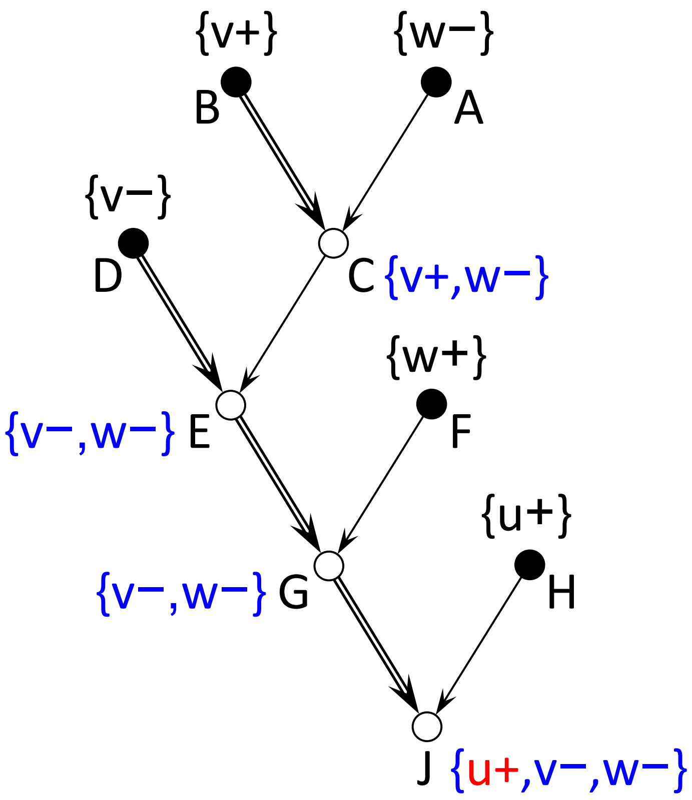
3 semantics for negative beliefs

Our recommendation

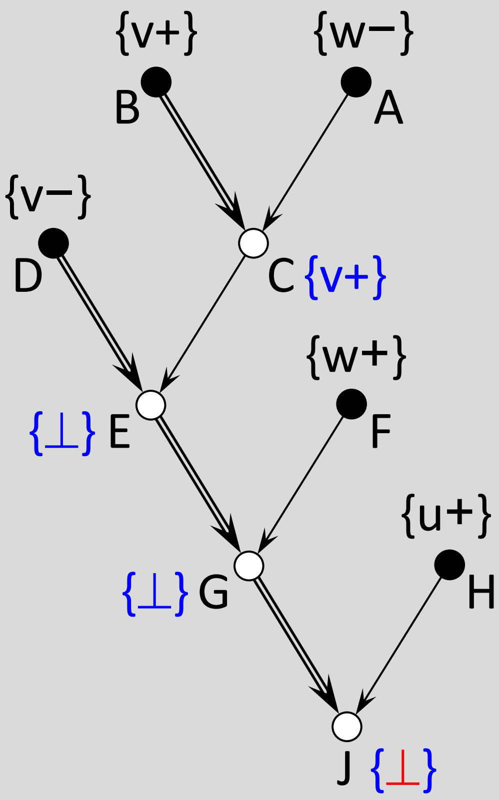
Agnostic



Eclectic



Skeptic



	Agnostic	Eclectic	Skeptic
w/o cycles*	**O(n)**	**O(n)**	**O(n)**
w cycles	**NP-hard**	**NP-hard**	**O(n^2)**

with a variation of resolution algorithm

* assuming total order on parents for each node

Take-aways automatic conflict resolution

Problem

- Given explicit beliefs & trust mappings, how to assign consistent value assignment to users?

Our solution

- Stable solutions with possible/certain value semantics
- PTIME algorithm [**$O(n^2)$** worst case, **$O(n)$** experiments]
- Several extensions
  - negative beliefs: 3 semantics, two hard, one **$O(n^2)$**
  - bulk inserts
  - agreement checking
  - consensus value
  - lineage computation

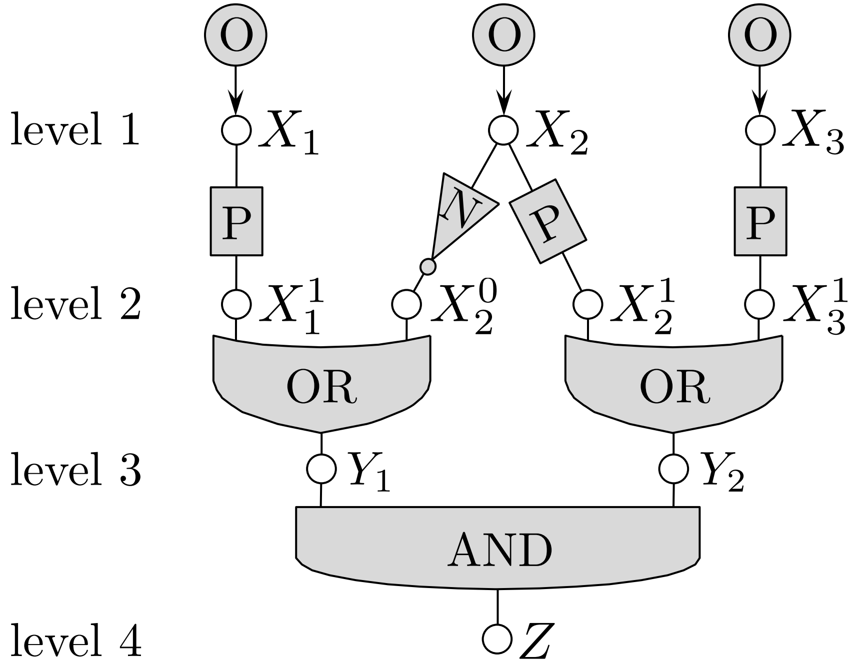
in the paper & TR

https://db.cs.washington.edu/projects/beliefdb/

details

Fig_ComplexityExampleLong

8-17-2010



Encoding

(0/1) = (a+/b+)

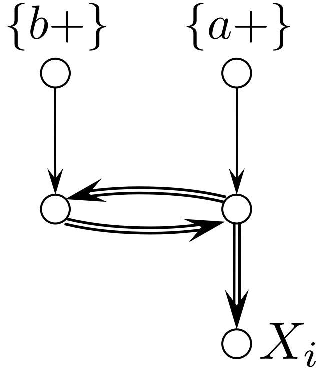
(0/1) = (c+/d+)

(0/1) = (e+/d+)

(0/1) = (e+/f+)

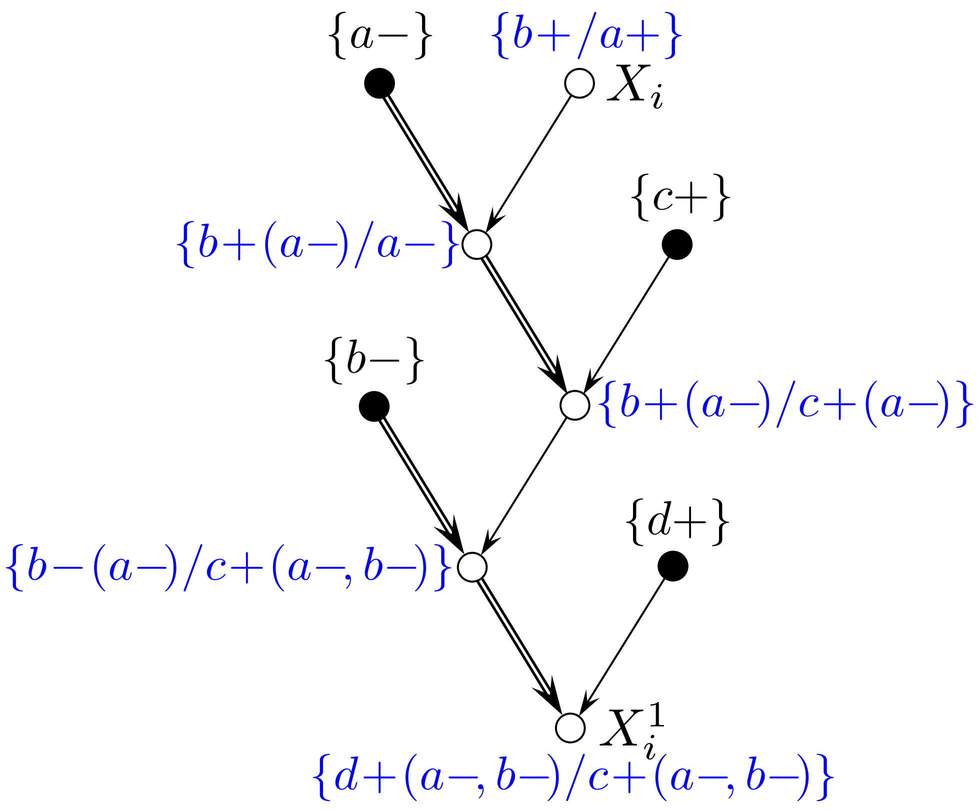
Fig_ComplexityOscillator

8-16-2010



Fig_ComplexityPassLong

8-17-2010



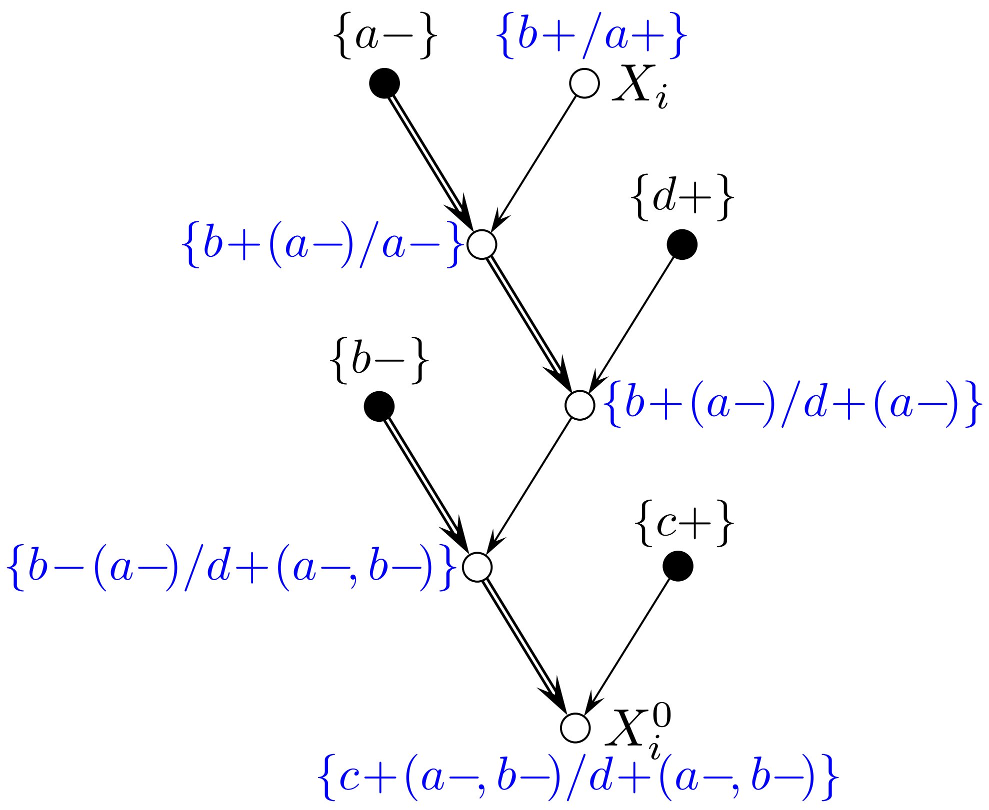
Encoding

(0/1) = (a+/b+)

(0/1) = (c+/d+)

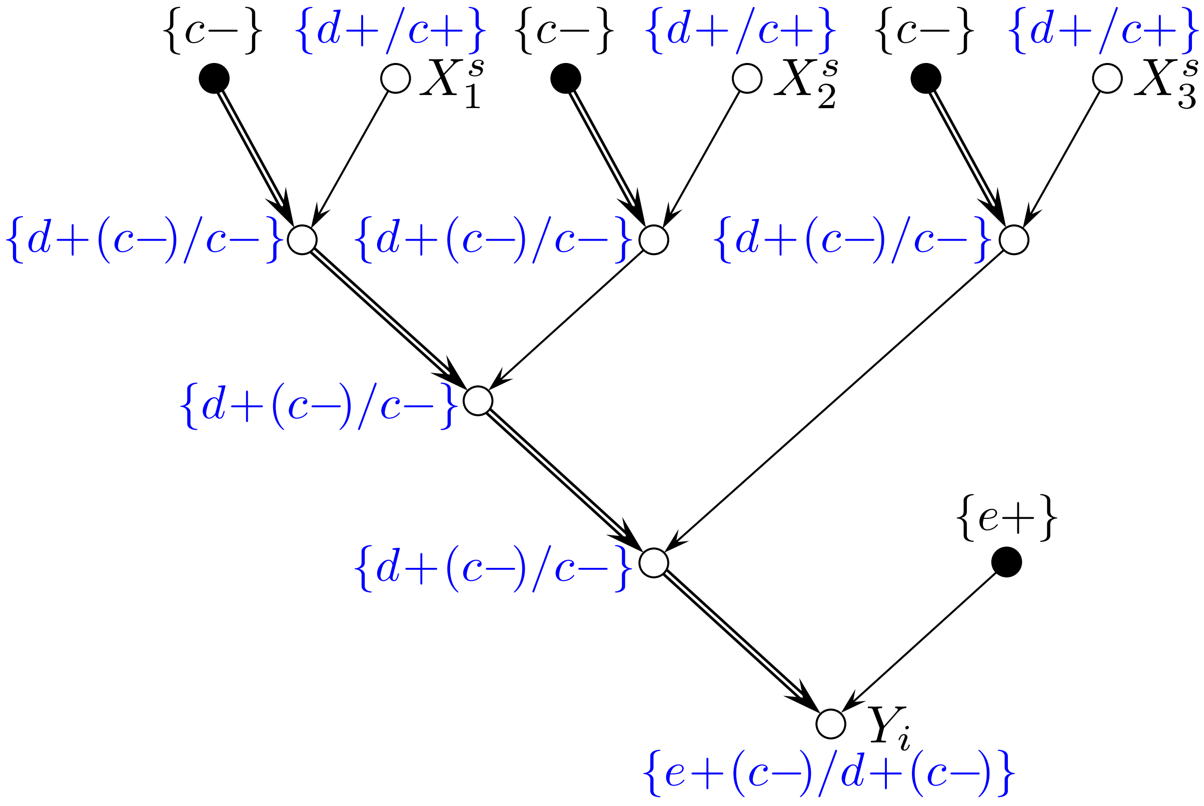
Fig_ComplexityNotLong

8-17-2010



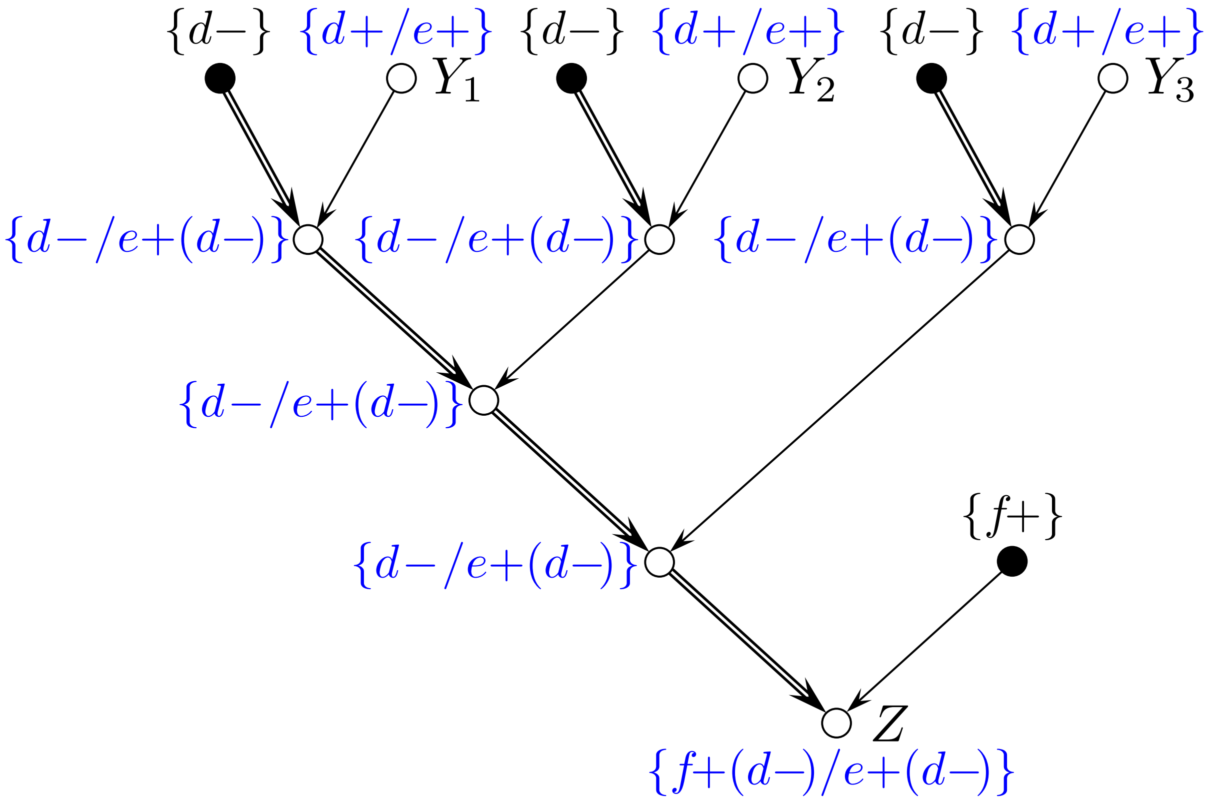
Fig_ComplexityOrLong

8-17-2010



Fig_ComplexityAndLong

8-17-2010



DEFINITION 3.1 (CONSISTENCY). *Two beliefs b_1, b_2 are conflicting* ($b_1 \not\leftrightarrow b_2$) *if they are either distinct positive beliefs* $v+, w+$, *or one is* $v+$ *and the other is* $v-$. *Otherwise,* b_1, b_2 *are consistent* ($b_1 \leftrightarrow b_2$). *A set of beliefs B is called consistent if any two beliefs $b_1, b_2 \in B$ are consistent.*

DEFINITION 3.2 (PREFERRED UNION). *Given two consistent sets of beliefs B_1, B_2, their preferred union is:*

$$B_1 \vec{\cup} B_2 = B_1 \cup \{b_2 \mid b_2 \in B_2 . \big(\forall b_1 \in B_1 . b_1 \leftrightarrow b_2\big)\}$$

be a consistent set of positive and/or negative beliefs. For each paradigm $\sigma \in \{\texttt{Agnostic}, \texttt{Eclectic}, \texttt{Skeptic}\}$ (abbreviated by $\{\texttt{A}, \texttt{E}, \texttt{S}\}$), the *normal form* $Norm_\sigma(B)$ is:

$$Norm_{\texttt{A}}(B) = \begin{cases} \{v+\} & \text{if } \exists v+ \in B \\ B & \text{otherwise} \end{cases}$$

$$Norm_{\texttt{E}}(B) = B$$

$$Norm_{\texttt{S}}(B) = \begin{cases} \{v+\} \cup (\bot - \{v-\}) & \text{if } \exists v+ \in B \\ B & \text{otherwise} \end{cases}$$

The *preferred union specialized to the paradigm* σ is:

$$B_1 \vec{\cup}_\sigma B_2 = Norm_\sigma \left(Norm_\sigma(B_1) \vec{\cup} Norm_\sigma(B_2) \right) \qquad (1)$$

For example:

$$\{a-\} \vec{\cup}_{\texttt{A}} \{b+\} = \{b+\}$$

$$\{a-\} \vec{\cup}_{\texttt{E}} \{b+\} = \{b+, a-\}$$

$$\{a-\} \vec{\cup}_{\texttt{S}} \{b+\} = \{b+, a-, c-, d-, \ldots\}$$

$$\{b-\} \vec{\cup}_{\texttt{S}} \{b+\} = \bot$$

$a^- \; (a^+ \;\; b^+)$

$(a^- \;\; a^+) \;\; b^+$

A puzzling question is why is the `Skeptic` paradigm in PTIME, while the other two are hard. It is easy to see that the Boolean gates in Fig. 7 no longer work under `Skeptic`, but we do not consider this a satisfactory explanation. While we cannot give an ultimate cause, we point out one interesting difference. The preferred union for `Skeptic` is *associative*, while it is not associative for either `Agnostic` nor `Eclectic`. For example, consider the two expressions $B_1 = \{a-\} \vec{\cup}_\sigma \left(\{a+\} \vec{\cup}_\sigma \{b+\}\right)$, $B_2 = \left(\{a-\} \vec{\cup}_\sigma \{a+\}\right) \vec{\cup}_\sigma \{b+\}$. For `Agnostic`, we have $B_2 = \{b+\}$, for `Eclectic` $B_2 = \{a-, b+\}$, while for both $B_1 = \{a-\}$. By contrast, one can show that $\vec{\cup}_{\mathsf{S}}$ is associative. Associativity as a desirable property during data merging was pointed out in [14].

The issue of associativity

null appears in a join column. No matter what choice is taken, $⟕$ is not associative. Consider the relations

$q(A$	$B)$
1	2

$r(B$	$C)$
2	3

$s(A$	$C)$
1	4

Computing $(q ⟕ r) ⟕ s$ we get

$q'(A$	B	$C)$
1	2	3
1	$\perp$	4

while $q ⟕ (r ⟕ s)$ gives

$q''(A$	B	$C)$
1	2	4
$\perp$	2	3

$$\{a^-\} \vec{\cup}_a (\{a\} \vec{\cup}_a \{b\}) = \{a^-\}$$

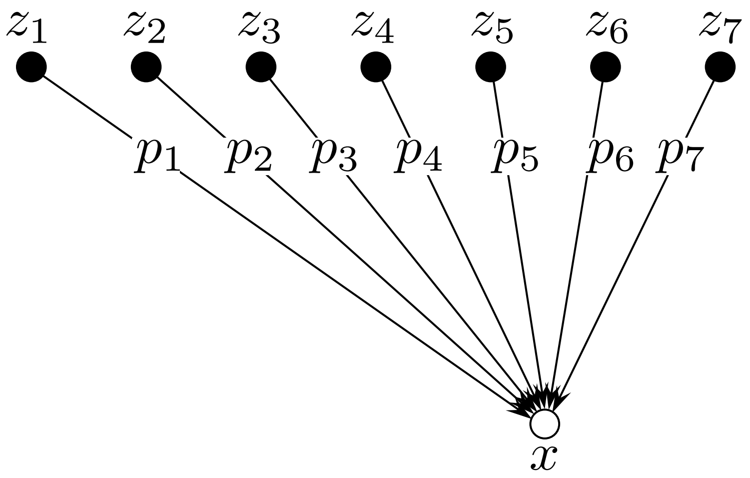
$$(\{a^-\} \vec{\cup}_a \{a\}) \vec{\cup}_a \{b\} = \{b\}$$

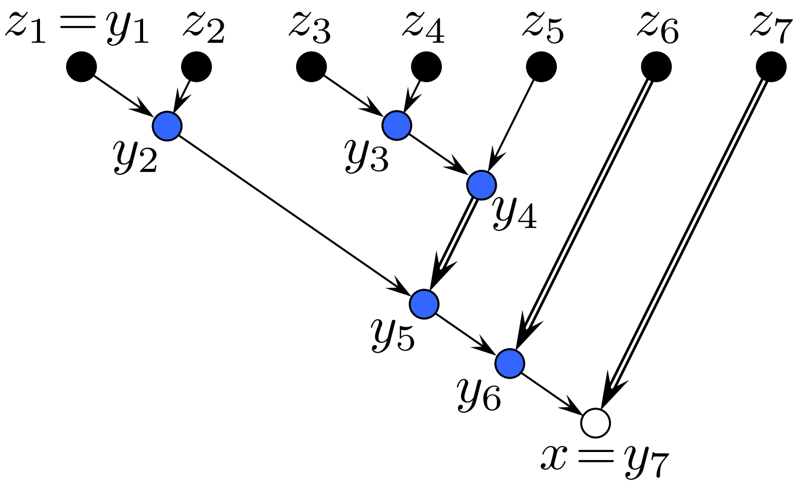
left outer join example from p392 in "Maier. The theory of relational databases, 1983."

right preferred union example from "Gatterbauer, Suciu. Conflict resolution using trust mapping. SIGMOD 2010.

backup

Binarization example





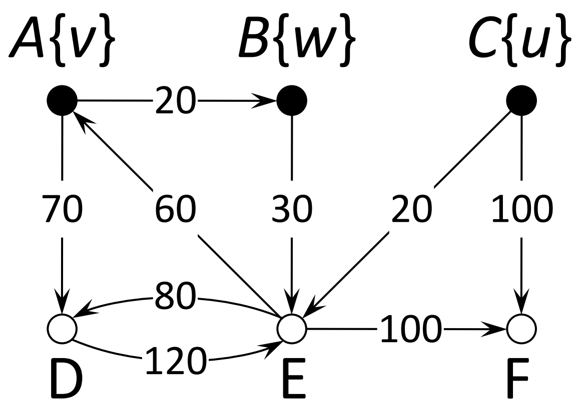
$p_1 = p_2 < p_3 = p_4 = p_5 < p_6 < p_7$

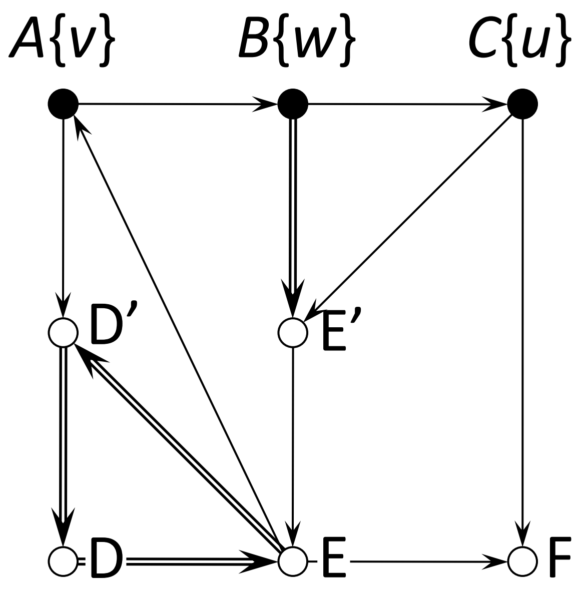
Binarization for Resolution Algorithm*

Example Trust Network (TN)
6 nodes, 9 arcs (size 15)
3 explicit beliefs: A:v, B:w, C:u

Corresponding Binary TN (BTN)
8 nodes, 12 arcs (size 20)

Size increase (N+E): ≤ 3

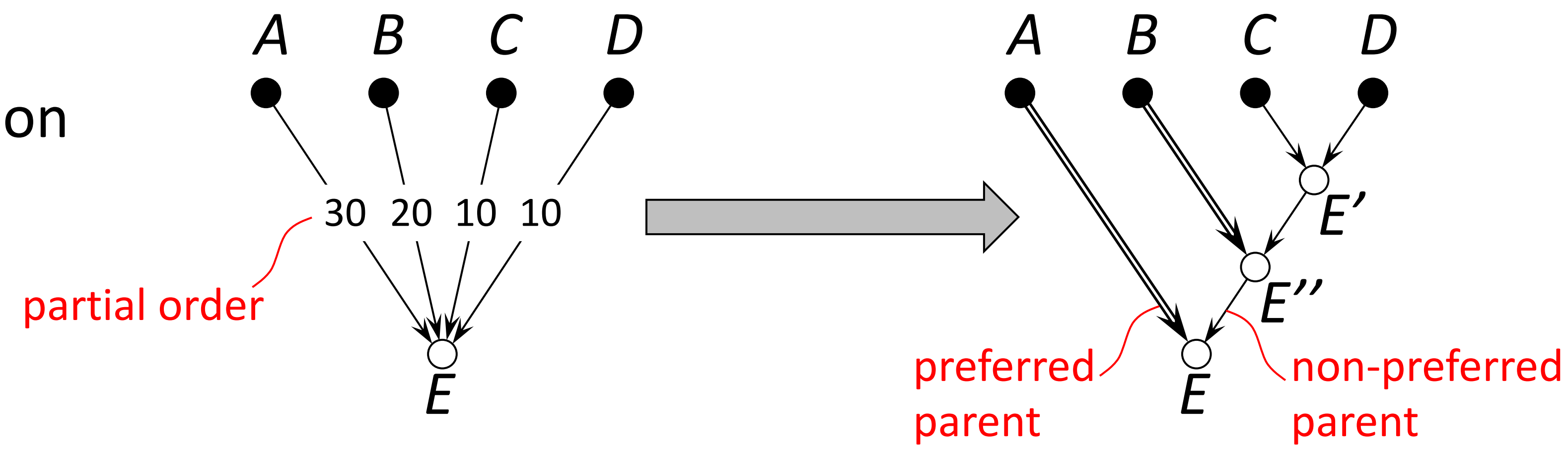




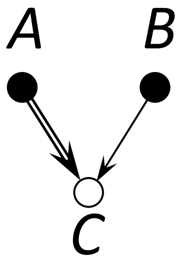
* Note that binarization is not necessary, but greatly simplifies the presentation

Logic programs with stable model semantics

Step 1:
Binarization

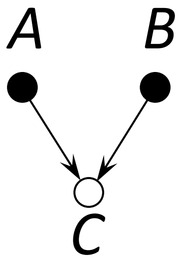


Step 2:
Logic program



1: accept all **poss** of preferred parent

P(C,x) :- P(A,x)
F(C,B,y) :- P(B,y), P(C,x), x≠y
P(C,y) :- P(B,y), ¬F(C,B,y



F(C,A,y) :- P(A,y), P(C,x), x≠y
P(C,y) :- P(A,y), ¬F(C,A,y)
F(C,B,y) :- P(B,y), P(C,x), x≠y
P(C,y) :- P(B,y), ¬F(C,B,y)

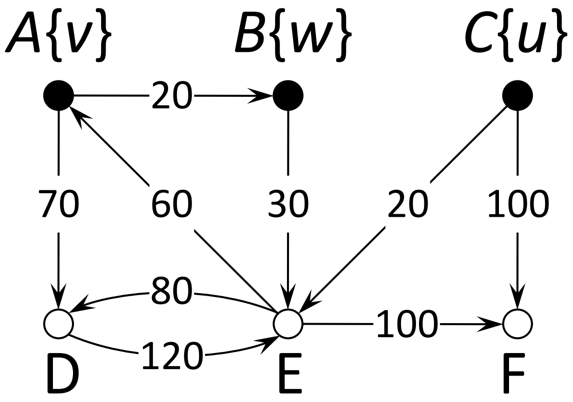
2: accept **poss** from non-preferred parent, that are not conflicting with an existing value

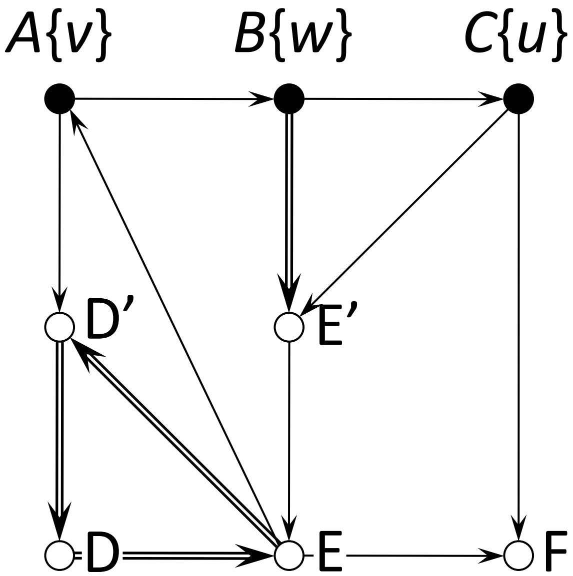
Binarization for Resolution Algorithm*

Example Trust Network (TN)
6 nodes, 9 arcs (size 15)
3 explicit beliefs: A:v, B:w, C:u

Corresponding Binary TN (BTN)
8 nodes, 12 arcs (size 20)

Size increase : ≤ 3

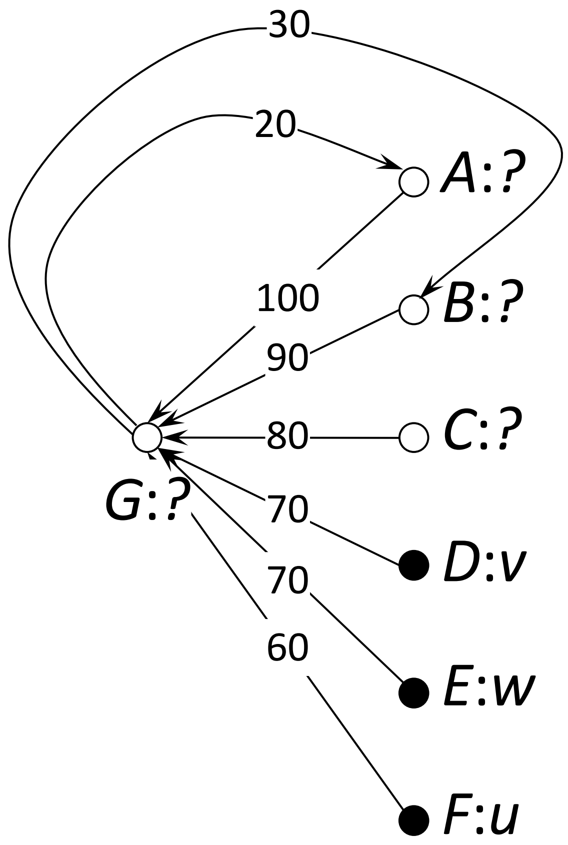




* Note that binarization is not necessary, but greatly simplifies the presentation

Stable solutions: example 2

- Priority trust network (TN)
  - assume a fixed key
  - users (nodes): *A*, *B*, *C*
  - values (beliefs): *v*, *w*, *u*
  - trust mappings (arcs) from "parents"
- Stable solution
  - assignment of values to each node*, s.t. each belief has a "*non-dominated lineage*" to an explicit belief
- Certain values
  - all stable solution determine, for each node, a possible value ("poss")
  - certain value ("cert") = intersection of all stable solutions



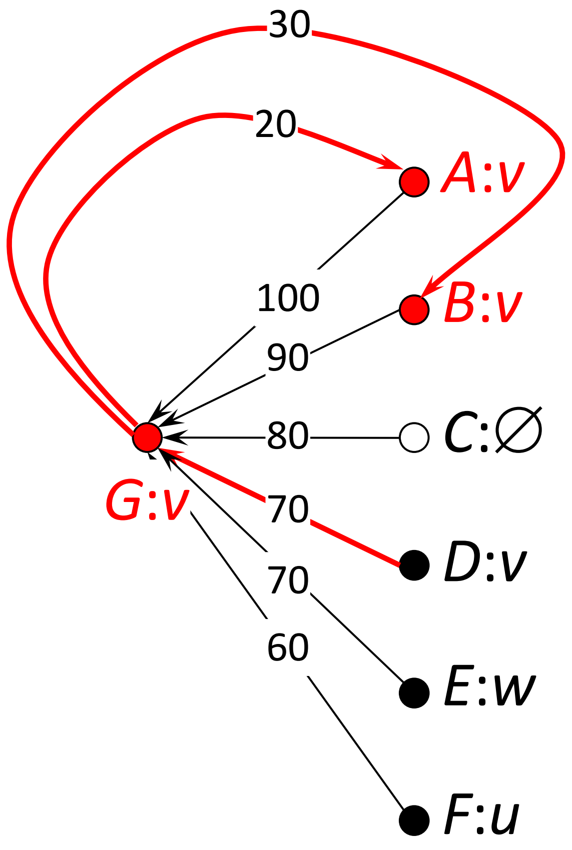
* each node with at least one ancestor with explicit belief

Stable solutions: example 2

- Priority trust network (TN)
  - assume a fixed key
  - users (nodes): *A*, *B*, *C*
  - values (beliefs): *v*, *w*, *u*
  - trust mappings (arcs) from "parents"

- Stable solution
  - assignment of values to each node*, s.t. each belief has a "*non-dominated lineage*" to an explicit belief

- Certain values
  - all stable solution determine, for each node, a possible value ("poss")
  - certain value ("cert") = intersection of all stable solutions



poss(*G*) = {*v*,...}

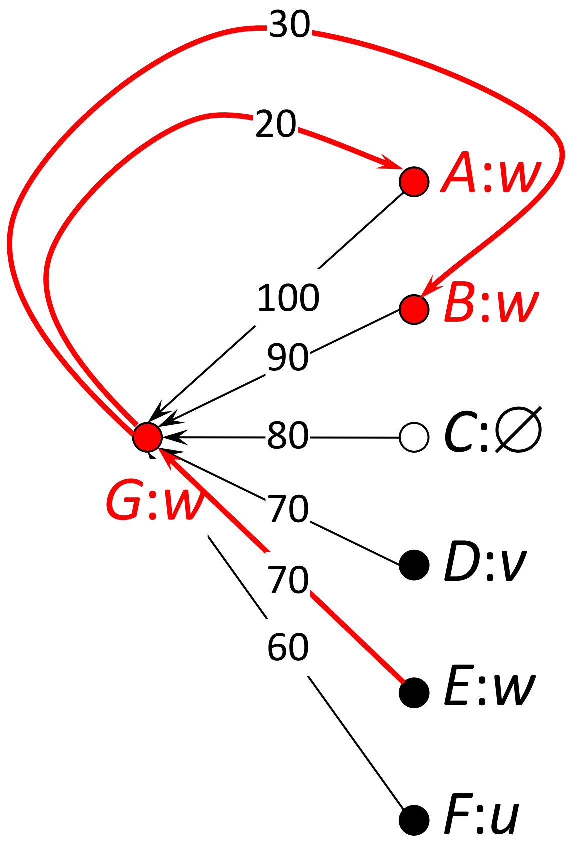
* each node with at least one ancestor with explicit belief

Stable solutions: example 2

- Priority trust network (TN)
  - assume a fixed key
  - users (nodes): *A*, *B*, *C*
  - values (beliefs): *v*, *w*, *u*
  - trust mappings (arcs) from "parents"

- Stable solution
  - assignment of values to each node*, s.t. each belief has a "*non-dominated lineage*" to an explicit belief

- Certain values
  - all stable solution determine, for each node, a possible value ("poss")
  - certain value ("cert") = intersection of all stable solutions

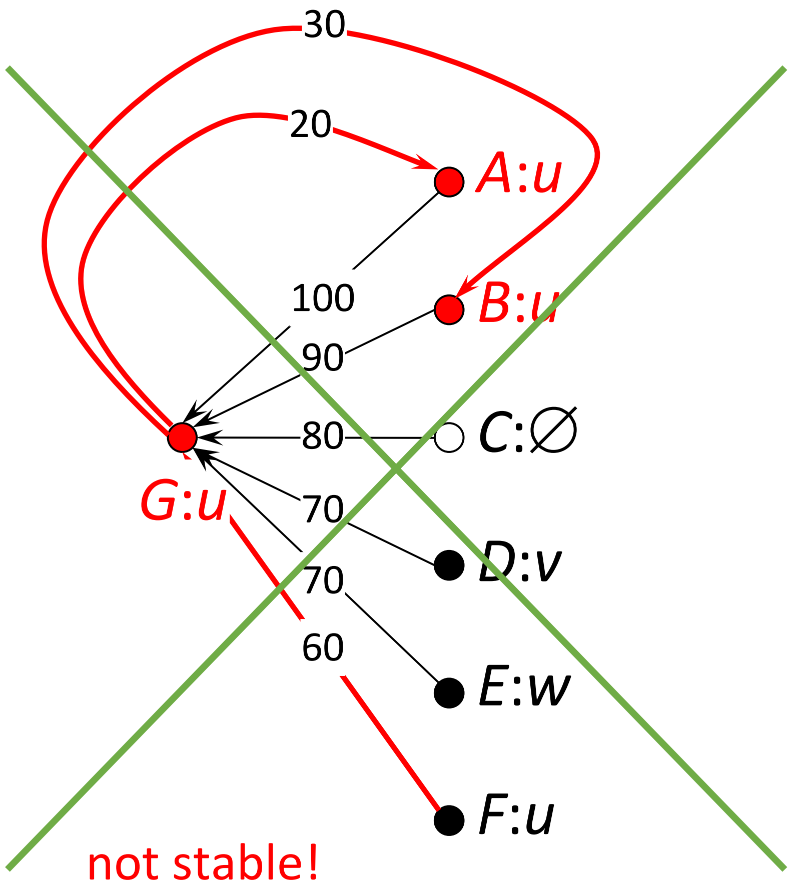


poss(*G*) = {*v*,*w*,...}

* each node with at least one ancestor with explicit belief

Stable solutions: example 2

- Priority trust network (TN)
  - assume a fixed key
  - users (nodes): *A*, *B*, *C*
  - values (beliefs): *v*, *w*, *u*
  - trust mappings (arcs) from "parents"
- Stable solution
  - assignment of values to each node*, s.t. each belief has a "*non-dominated lineage*" to an explicit belief
- Certain values
  - all stable solution determine, for each node, a possible value ("poss")
  - certain value ("cert") = intersection of all stable solutions



not stable!
F→G dominated by E→G

poss(G) = {v,w}
cert(G) = ∅

* each node with at least one ancestor with explicit belief