Ridhwan Dewoprabowo[1] Johannes K. Fichte[2] Piotr Jerzy Gorczyca[1] Markus Hecher[2]
[1] TU Dresden, [2] TU Wien

A Practical Account into Counting Dung's Extensions by Dynamic Programming

LPNMR'22 conference talk, Genova, September 8th 2022

Abstract Argumentation (AA)

Argumentation

Formal framework [Dung, 1995] to:

- deal with **contentious information** and **draw conclusions** from it
- **represent conflicts between arguments**

Abstract Argumentation (AA)

Argumentation

Formal framework [Dung, 1995] to:

- deal with **contentious information** and **draw conclusions** from it
- **represent conflicts between arguments**

Extensions

- subsets of arguments **congruent mutually**
- **semantics** define how to choose such subsets
  - e.g. admissible, stable, grounded, preferred etc.

Motivation – Counting Extensions

Question

Instead of: **does** the problem **have a solution**? (decision)
We ask: **how many solutions** does the problem have? (quantity)

Motivation – Counting Extensions

Question

Instead of: **does** the problem **have a solution**? (decision)
We ask: **how many solutions** does the problem have? (quantity)

Counting in Argumentation

- Quantitative reasoning
- Probabilistic argumentation
- Counting impacts:
  - Bayesian inference, bounded-length adversarial and contingency planning, reliability estimation

Computational complexity

- #P-complete [Valiant, 1979]

Motivation – Counting Extensions

Question

Instead of: **does** the problem **have a solution**? (decision)
We ask: **how many solutions** does the problem have? (quantity)

Counting in Argumentation

- Quantitative reasoning
- Probabilistic argumentation
- Counting impacts:
  - Bayesian inference, bounded-length adversarial and contingency planning, reliability estimation

Computational complexity

- #P-complete [Valiant, 1979]

Counting $\neq$ Enumeration

Our Research

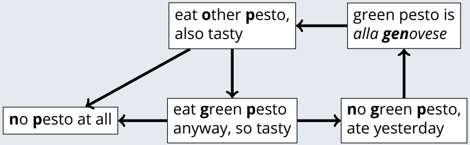
Contributions

- **Implementation** of theoretical algorithms for counting in AA,
- Development of a (first) **dedicated counting solver** for counting extensions of AFs under:
  - stable, admissible and complete semantics
- **Empirical evaluation** illustrating that **our system can be competitive**.

Example of an AA – what's for dinner tonight?

Framework F

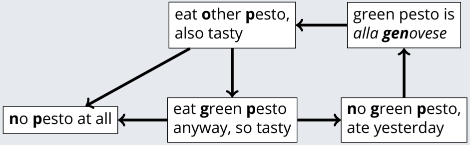
$F = (A, R)$, $R \subseteq A \times A$



Example of an AA – what's for dinner tonight?

Framework F

$F = (A, R)$, $R \subseteq A \times A$



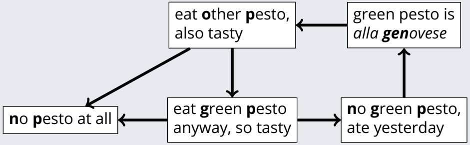
Extension S is

- **stable** if conflict-free, defends itself, and attacks every $a \in F \setminus S$ in F, and for each $a \in A \setminus S, S \rightarrowtail a$.

Example of an AA – what's for dinner tonight?

Framework F

$F = (A, R)$, $R \subseteq A \times A$



Extensions

stable: $\{\{gp, gen\}, \{op, ngp\}\}$

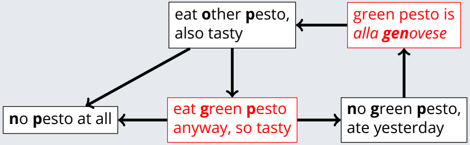
Extension S is

- **stable** if conflict-free, defends itself, and attacks every $a \in F \setminus S$ in F, and for each $a \in A \setminus S, S \rightarrowtail a$.

Example of an AA – what's for dinner tonight?

Framework F

$F = (A, R)$, $R \subseteq A \times A$



Extensions

stable: $\{\{gp, gen\}, \{op, ngp\}\}$

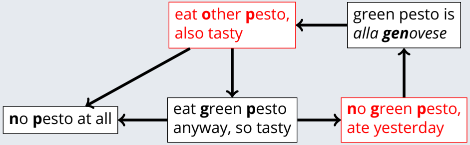
Extension S is

- **stable** if conflict-free, defends itself, and attacks every $a \in F \setminus S$ in F, and for each $a \in A \setminus S, S \rightarrowtail a$.

Example of an AA – what's for dinner tonight?

Framework F

$F = (A, R)$, $R \subseteq A \times A$



Extensions

stable: $\{\{gp, gen\}, \{op, ngp\}\}$

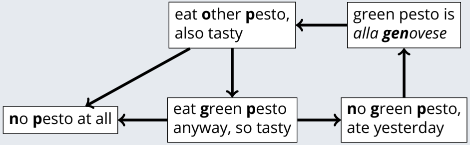
Extension S is

- **stable** if conflict-free, defends itself, and attacks every $a \in F \setminus S$ in F, and for each $a \in A \setminus S, S \rightarrowtail a$.

Example of an AA – what's for dinner tonight?

Framework F

$F = (A, R)$, $R \subseteq A \times A$



Extensions

stable: $\left| \{\{gp, gen\}, \{op, ngp\}\} \right| = 2$

Extension S is

- **stable** if conflict-free, defends itself, and attacks every $a \in F \setminus S$ in F, and for each $a \in A \setminus S, S \rightarrowtail a$.

Counting in Abstract Argumentation – Introduction & Motivation

Utilizing Treewidth and Dynamic Programming

Empirical Evaluation

Conclusions & Future Work

Utilizing Treewidth

Problem

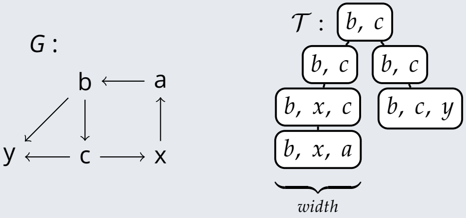
#P-complete problems are **hard to solve**...

Idea

Decompose the initial problem into trivial subproblems, combine the subsolutions
Parameter: overlap between subproblems – treewidth of the primal graph

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G

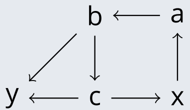


Definition

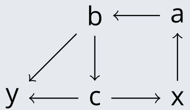
A tree decomposition is a tree obtained from an arbitrary graph s.t.

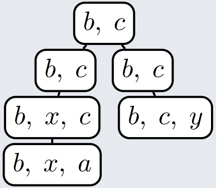
1. Each vertex must occur in some *bag*
2. For each edge, there is a bag containing both endpoints
3. *Connected*: Connected: If a vertex v appears in bags t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1.

Counting Stable Extensions [Charwat, 2012]

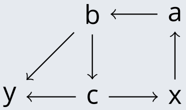


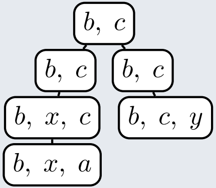
Counting Stable Extensions [Charwat, 2012]





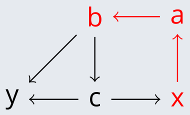
Counting Stable Extensions [Charwat, 2012]

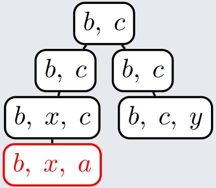




$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions [Charwat, 2012]



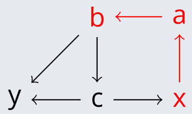


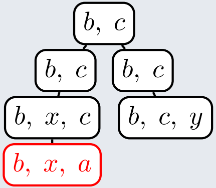
$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

b	x	a
out	*out*	*out*
out	*out*	*def*
out	*out*	*in*
out	*def*	*out*
out	*def*	*def*
out	*def*	*in*
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*out*	*in*
def	*def*	*out*
def	*def*	*def*
def	*def*	*in*
def	*in*	*out*
def	*in*	*def*
def	*in*	*in*

b	x	a
in	*out*	*out*
in	*out*	*def*
in	*out*	*in*
in	*def*	*out*
in	*def*	*def*
in	*def*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

Counting Stable Extensions [Charwat, 2012]



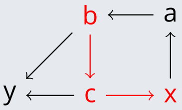


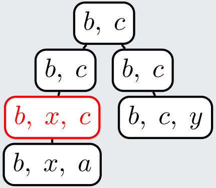
b	x	a
out	*out*	*out*
out	*out*	*def*
out	*out*	*in*
out	*def*	*out*
out	*def*	*def*
out	*def*	*in*
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*out*	*in*
def	*def*	*out*
def	*def*	*def*
def	*def*	*in*
def	*in*	*out*
def	*in*	*def*
def	*in*	*in*

b	x	a
in	*out*	*out*
in	*out*	*def*
in	*out*	*in*
in	*def*	*out*
in	*def*	*def*
in	*def*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions [Charwat, 2012]



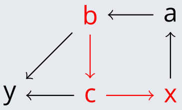


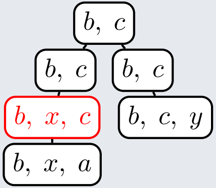
b	x	c
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*out*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

b	x	a
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions [Charwat, 2012]





b	x	c
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*def*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

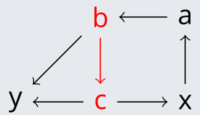
b	x	a
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

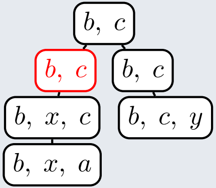
$C(a) = in$ iff $a \in S$

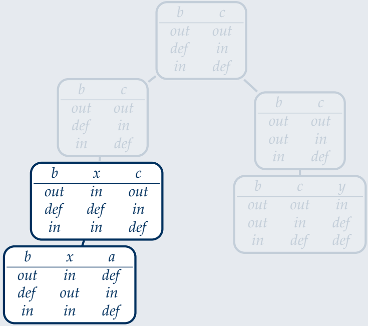
$C(a) = def$ iff $S \rightarrowtail a$

$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions [Charwat, 2012]

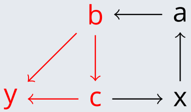


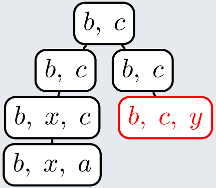


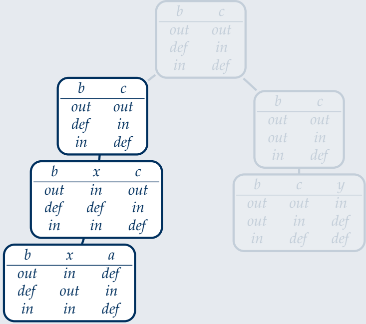


$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions [Charwat, 2012]

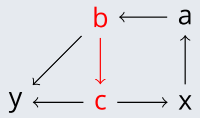


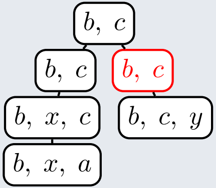


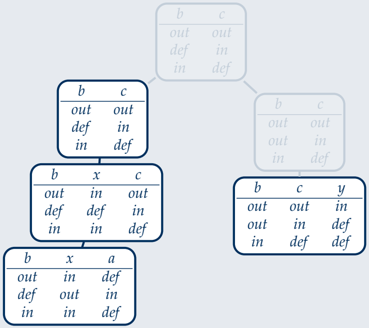


$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions [Charwat, 2012]

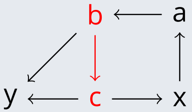


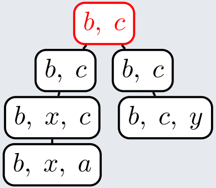


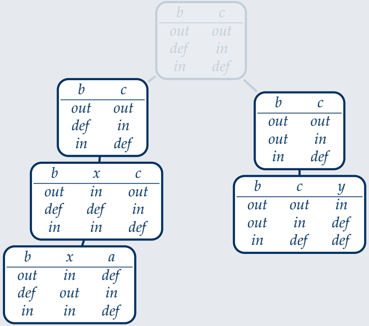


$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions [Charwat, 2012]

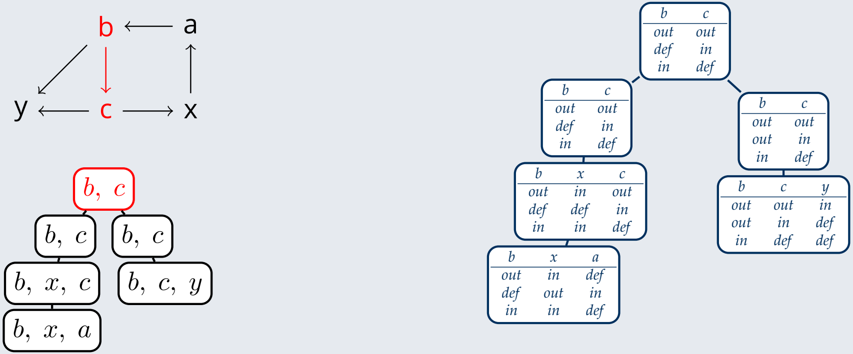




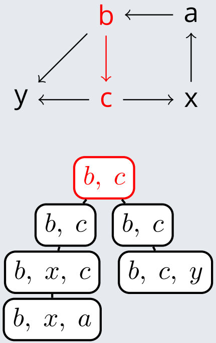


$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions [Charwat, 2012]



Counting Stable Extensions [Charwat, 2012]



b	c	#
out	out	–
def	in	1
in	def	1

b	c	#
out	out	1
def	in	1
in	def	1

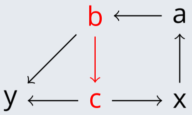
b	c	#
out	out	1
out	in	1
in	def	1

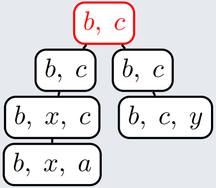
b	x	c	#
out	in	out	1
def	def	in	1
in	in	def	1

b	c	y	#
out	out	in	1
out	in	def	1
in	def	def	1

b	x	a	#
out	in	def	1
def	out	in	1
in	in	def	1

Counting Stable Extensions [Charwat, 2012]





b	c	#
out	*out*	–
def	*in*	1
in	*def*	1

b	c	#
out	*out*	1
def	*in*	1
in	*def*	1

b	c	#
out	*out*	1
out	*in*	1
in	*def*	1

b	x	c	#
out	*in*	*out*	1
def	*def*	*in*	1
in	*in*	*def*	1

b	c	y	#
out	*out*	*in*	1
out	*in*	*def*	1
in	*def*	*def*	1

b	x	a	#
out	*in*	*def*	1
def	*out*	*in*	1
in	*in*	*def*	1

Semantics	Colouring	Runtime
Stable	$C_t : \chi(t) \mapsto \{in, def, out\}$	$3^{O(\text{treewidth})} \cdot \text{poly}(\lvert Args \rvert)$

Implementation

DPDB

DPDB [Fichte et al., 2020] – a general framework utilizing: **Tree Decompositions**, **Dynamic Programming** and **Database Management Systems**

- Currently supporting: #SAT, Vertex Cover ...

Implementation

DPDB

DPDB [Fichte et al., 2020] – a general framework utilizing: **Tree Decompositions**, **Dynamic Programming** and **Database Management Systems**

- Currently supporting: #SAT, Vertex Cover ...
- Now also counting extensions of AA frameworks!

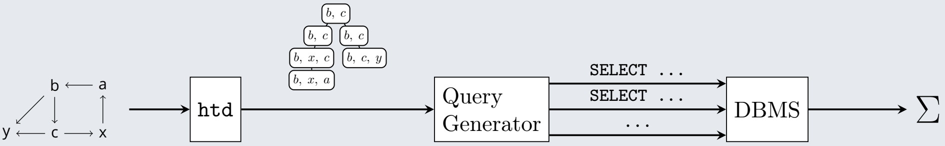
Implementation

DPDB

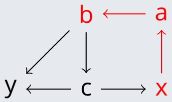
DPDB [Fichte et al., 2020] – a general framework utilizing: **Tree Decompositions**, **Dynamic Programming** and **Database Management Systems**

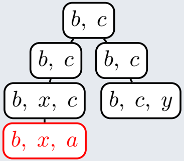
- Currently supporting: #SAT, Vertex Cover …
- Now also counting extensions of AA frameworks!

DPDB Architecture



DPDB in Practice – Generated SQL Query





vi	*di*	*meaning*
0	0	*out*
0	1	*def*
1	–	*in*

Query

```
SELECT va,vb,vx, da,db,dx,
    sum(model_count) AS model_count
FROM (WITH introduce AS
  (SELECT true val UNION ALL SELECT false)
SELECT ia.val va, ib.val vb, ix.val vx,
    (ix.val) AS da,(ia.val) AS db,FALSE AS dx,
    1 AS model_count
FROM introduce ib, /*introduce*/
  introduce ix, introduce ia) AS candidate
WHERE (va OR da) AND /*forget a*/
(NOT (va AND vb)) AND /*conflict-free*/
(NOT (vx AND va))
GROUP BY va,vb,vx, da,db,dx
```

Query output

vb	*db*	*vx*	*dx*	*va*	*da*	*model_count*
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	1	0	0	1	1

Meaning

b	*x*	*a*
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

Counting in Abstract Argumentation – Introduction & Motivation

Utilizing Treewidth and Dynamic Programming

Empirical Evaluation

Conclusions & Future Work

Evaluation – Setup

Other systems

- Leading solvers of the recent editions of ICCMA: μ-`toksia` [Niskanen and Järvisalo, 2020], `aspartix` [Dvořák et al., 2020], `pyglaf` [Alviano, 2017]
- State-of-the-art propositional model counters: `SharpSAT-td`, `d4` [1]

[1] using `aspartix`'s ASP encoding and translating the ground ASP instance into a SAT formula (with the help of `lp2normal` and `lp2sat`)
[2] more recent instances out of reach for DPDB due to TW.

Evaluation – Setup

Other systems

- Leading solvers of the recent editions of ICCMA: μ-toksia [Niskanen and Järvisalo, 2020], aspartix [Dvořák et al., 2020], pyglaf [Alviano, 2017]
- State-of-the-art propositional model counters: SharpSAT-td, d4 [1]

(Virtual) portfolio solvers

- Portfolio solvers: DPDB+X, $X \in \{$aspartix, μ-toksia, pyglaf$\}$
- Virtual portfolio solvers: aspartix+X, $X \in \{$ sharpSAT-td, d4 $\}$

[1] using aspartix's ASP encoding and translating the ground ASP instance into a SAT formula (with the help of lp2normal and lp2sat)
[2] more recent instances out of reach for DPDB due to TW.

Evaluation – Setup

Other systems

- Leading solvers of the recent editions of ICCMA: μ-toksia [Niskanen and Järvisalo, 2020], aspartix [Dvořák et al., 2020], pyglaf [Alviano, 2017]
- State-of-the-art propositional model counters: SharpSAT-td, d4 [1]

(Virtual) portfolio solvers

- Portfolio solvers: DPDB+X, $X \in \{$aspartix, μ-toksia, pyglaf$\}$
- Virtual portfolio solvers: aspartix+X, $X \in \{$ sharpSAT-td, d4 $\}$

Benchmarks

- ICCMA'17 instances [Gaggl et al., 2018][2]
- 600s timeouts

[1] using aspartix's ASP encoding and translating the ground ASP instance into a SAT formula (with the help of lp2normal and lp2sat)

[2] more recent instances out of reach for DPDB due to TW.

Evaluation – Results

solver	adm.	comp.	stab.
aspartix	236	362	469
... /d4	347	406	483
... /sharpSAT-td	**368**	**410**	**487**
dpdb	96	100	113
...+aspartix	*311*	*379*	475
...+μ-toksia21	95	367	468
...+pyglaf	300	372	*478*
μ-toksia21	–	299	446
pyglaf	221	336	463
sharpSAT-td	284	350	387
vbest	371	411	505

(a) Number of solved instances of various solvers.

	adm.	comp.	stab.
median	2.9	0.5	0.0
mean	11.6	8.3	3.8
max	512.6	487.7	498.2
aspartix	7.9	8.3	8.7
dpdb	154.6	119.9	75.0
mu_toksia21	–	5.1	5.2
pyglaf	6.1	6.5	5.8
sharpSAT-td	512.6	487.7	498.2

(b) Observed counts in $\log_{10}$ format.The lower part states the maximum count observed for the respective solver.

Evaluation – Results

solver	adm.	comp.	stab.
aspartix	236	362	469
... /d4	347	406	483
... /sharpSAT-td	**368**	**410**	**487**
dpdb	96	100	113
...+aspartix	*311*	*379*	475
...+μ-toksia21	95	367	468
...+pyglaf	300	372	*478*
μ-toksia21	–	299	446
pyglaf	221	336	463
sharpSAT-td	284	350	387
vbest	371	411	505

(a) Number of solved instances of various solvers.

	adm.	comp.	stab.
median	2.9	0.5	0.0
mean	11.6	8.3	3.8
max	512.6	487.7	498.2
aspartix	7.9	8.3	8.7
dpdb	154.6	119.9	75.0
mu_toksia21	–	5.1	5.2
pyglaf	6.1	6.5	5.8
sharpSAT-td	512.6	487.7	498.2

(b) Observed counts in $\log_{10}$ format.The lower part states the maximum count observed for the respective solver.

Note:

Enumeration works fine only when the number of solutions is small.

Results – Admissible Semantics

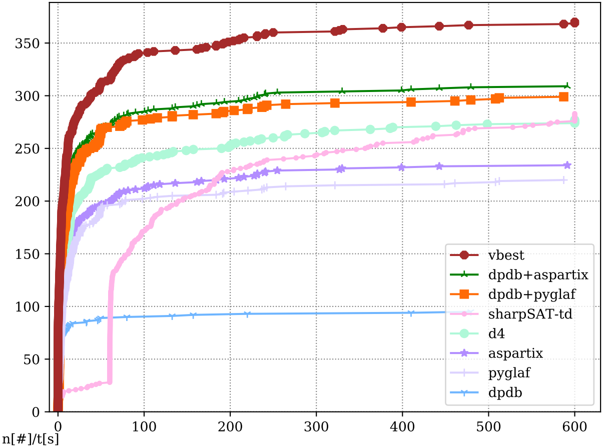


Figure: Runtime of various solvers for admissible semantics.

Conclusions

(Our extension of) DPDB

- utilizes Dynamic Programming algorithms on Tree Decompositions with Database Management Systems

\+ works well with instances of small treewidth and high number of solutions,
  - otherwise enumeration is sufficient

\- cannot deal with instances of high treewidth,

- when used in a portfolio system can be competitive with the state-of-the-art solvers,
- is the first implementation of a dedicated counter for Abstract Argumentation.

Future Work

Upcoming Tasks

- addressing high treewidths with abstractions of Tree Decompositions (nested dynamic programming),
- using parallelization for large instances of low treewidth,
- developing dedicated preprocessing techniques for argumentation.

Future Work

Upcoming Tasks

- addressing high treewidths with abstractions of Tree Decompositions (nested dynamic programming),
- using parallelization for large instances of low treewidth,
- developing dedicated preprocessing techniques for argumentation.

Thank you for listening.

Future Work

Upcoming Tasks

- addressing high treewidths with abstractions of Tree Decompositions (nested dynamic programming),
- using parallelization for large instances of low treewidth,
- developing dedicated preprocessing techniques for argumentation.

Thank you for listening.

Sponsors:

- DFG Grant TRR 248 project ID 389792660 (CPEC); BMBF Grant 01IS20056_NAVAS;
  WWTF grant ICT19-065, FWF grants P32830 and Y698.

Bibliography I

Alviano, M. (2017).
Ingredients of the argumentation reasoner pyglaf: Python, circumscription, and glucose to taste.
In Maratea, M. and Serina, I., editors, *Proceedings of the 24th RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion 2017 co-located with the 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017), Bari, Italy, November 14-15, 2017*, volume 2011 of *CEUR Workshop Proceedings*, pages 1–16. CEUR-WS.org.

Charwat, G. (2012).
Tree-decomposition based algorithms for abstract argumentation framework.
Master's thesis, TU Wien, Vienna, Austria.

Dung, P. M. (1995).
On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.
AIJ, 77(2).

Dvořák, W., Rapberger, A., Wallner, J., and Woltran, S. (2020).
ASPARTIX-V19 - An Answer-Set Programming Based System for Abstract Argumentation.
In *FoIKS 2020*, volume 12012. Springer, Cham.

Bibliography II

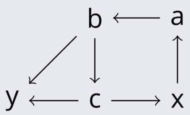
Fichte, J. K., Hecher, M., Thier, P., and Woltran, S. (2020).
Exploiting database management systems and treewidth for counting.
In *PADL*, volume 12007 of *LNCS*, pages 151–167. Springer.

Gaggl, S. A., Linsbichler, T., Maratea, M., and Woltran, S. (2018).
Summary report of the second international competition on computational models of argumentation.
AI Magazine, 39(4):77–79.

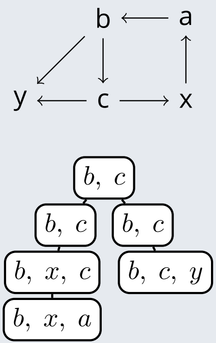
Niskanen, A. and Järvisalo, M. (2020).
μ-toksia: An efficient abstract argumentation reasoner.
In *KR 2020*, pages 800–804.

Valiant, L. (1979).
The complexity of computing the permanent.
Theoretical Computer Science, 8(2):189–201.

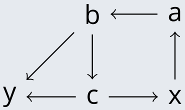
Counting Stable Extensions (Detailed) [Charwat, 2012]

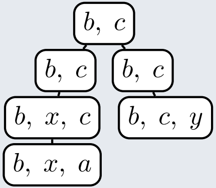


Counting Stable Extensions (Detailed) [Charwat, 2012]



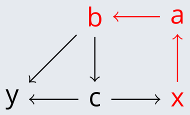
Counting Stable Extensions (Detailed) [Charwat, 2012]

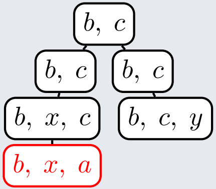




$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



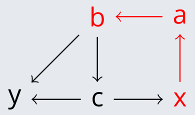


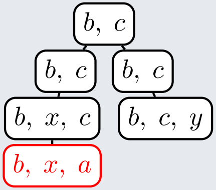
b	x	a
out	*out*	*out*
out	*out*	*def*
out	*out*	*in*
out	*def*	*out*
out	*def*	*def*
out	*def*	*in*
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*out*	*in*
def	*def*	*out*
def	*def*	*def*
def	*def*	*in*
def	*in*	*out*
def	*in*	*def*
def	*in*	*in*

b	x	a
in	*out*	*out*
in	*out*	*def*
in	*out*	*in*
in	*def*	*out*
in	*def*	*def*
in	*def*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



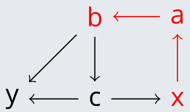


b	x	a
out	*out*	*out*
out	*out*	*def*
out	*out*	*in*
out	*def*	*out*
out	*def*	*def*
out	*def*	*in*
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*out*	*in*
def	*def*	*out*
def	*def*	*def*
def	*def*	*in*
def	*in*	*out*
def	*in*	*def*
def	*in*	*in*

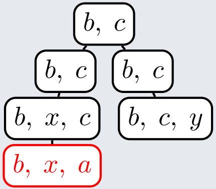
b	x	a
in	*out*	*out*
in	*out*	*def*
in	*out*	*in*
in	*def*	*out*
in	*def*	*def*
in	*def*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]

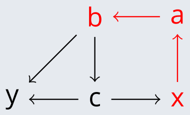


b	x	a
out	out	out
out	in	def
def	out	in
in	out	out
in	in	def

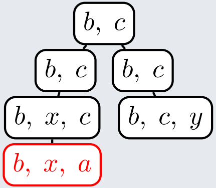


$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]

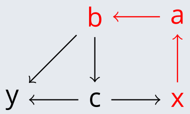


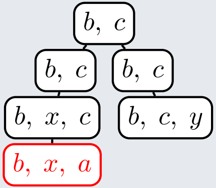
b	x	a
out	out	out
out	in	def
def	out	in
in	out	out
in	in	def



$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]

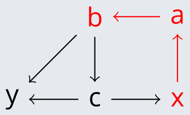


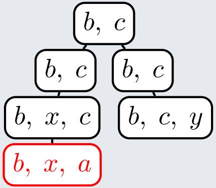


b	x	a
out	*out*	*out*
out	*in*	*def*
def	*out*	*in*
in	*out*	*out*
in	*in*	*def*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]

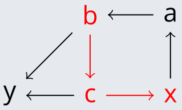


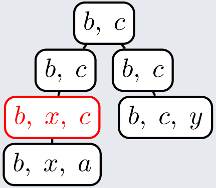


b	x	a
out	in	def
def	out	in
in	in	def

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



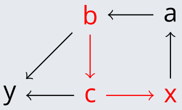


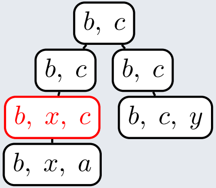
b	x	c
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*out*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

b	x	a
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



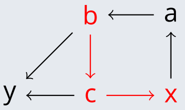


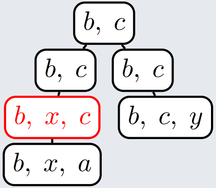
b	x	c
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*out*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

b	x	a
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



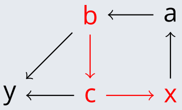


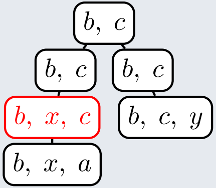
b	x	c
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	~~*out*~~ *def*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

b	x	a
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



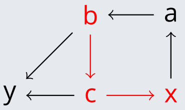


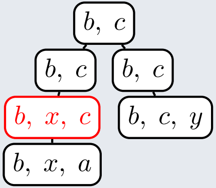
b	x	c
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*def*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

b	x	a
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



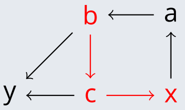


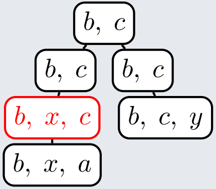
b	x	c
out	*in*	*out*
out	*in*	*def*
out	*in*	*in*
def	*out*	*out*
def	*out*	*def*
def	*def*	*in*
in	*in*	*out*
in	*in*	*def*
in	*in*	*in*

b	x	a
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



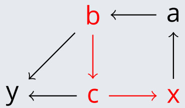


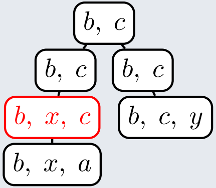
b	x	c
out	*in*	*out*
def	*out*	*out*
def	*def*	*in*
in	*in*	*def*

b	x	a
out	*in*	*def*
def	*out*	*in*
in	*in*	*def*

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]



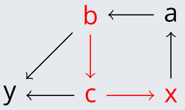


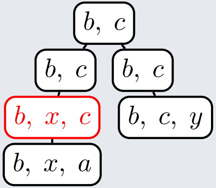
b	x	c
out	in	out
def	out	out
def	def	in
in	in	def

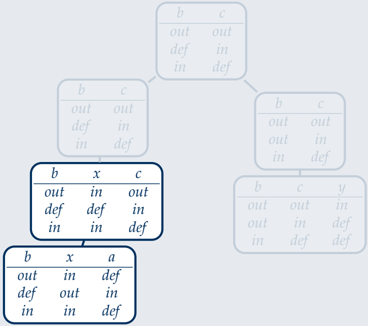
b	x	a
out	in	def
def	out	in
in	in	def

$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]





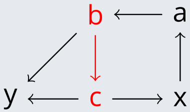


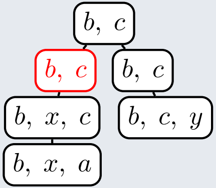
$C(a) = in$ iff $a \in S$

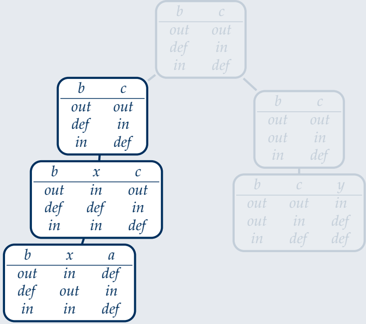
$C(a) = def$ iff $S \rightarrowtail a$

$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]

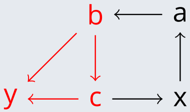


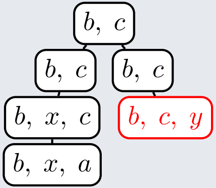


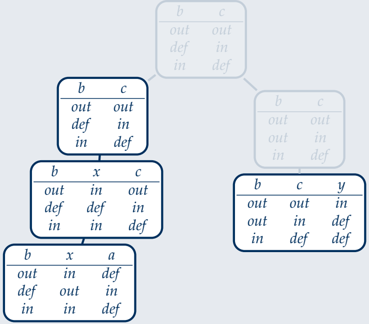


$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]





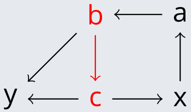


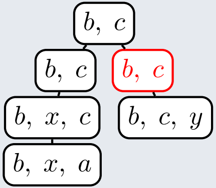
$C(a) = in$ iff $a \in S$

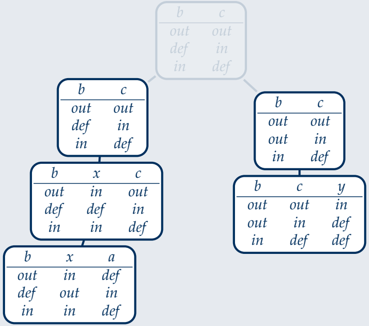
$C(a) = def$ iff $S \rightarrowtail a$

$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]





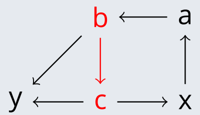


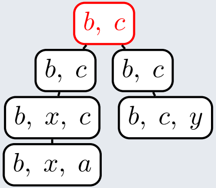
$C(a) = in$ iff $a \in S$

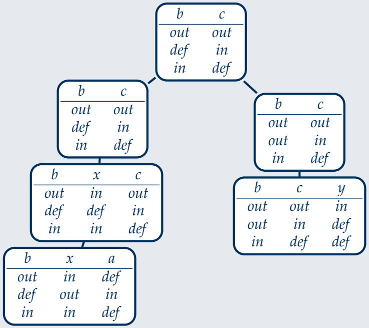
$C(a) = def$ iff $S \rightarrowtail a$

$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]

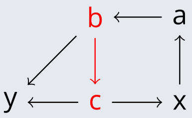


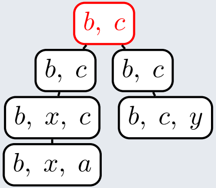


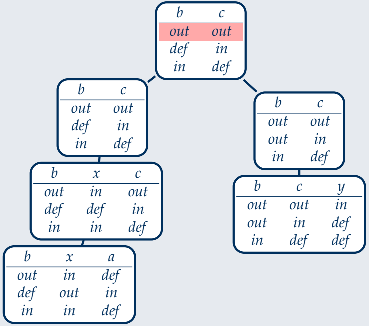


$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]

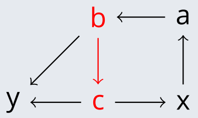


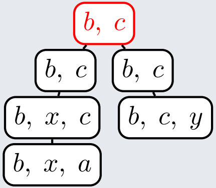




$C(a) = in$ iff $a \in S$
$C(a) = def$ iff $S \rightarrowtail a$
$C(a) = out$ iff $S \not\rightarrowtail a$ and $a \notin S$

Counting Stable Extensions (Detailed) [Charwat, 2012]





b	c	#
out	*out*	–
def	*in*	1
in	*def*	1

b	c	#
out	*out*	1
def	*in*	1
in	*def*	1

b	c	#
out	*out*	1
out	*in*	1
in	*def*	1

b	x	c	#
out	*in*	*out*	1
def	*def*	*in*	1
in	*in*	*def*	1

b	c	y	#
out	*out*	*in*	1
out	*in*	*def*	1
in	*def*	*def*	1

b	x	a	#
out	*in*	*def*	1
def	*out*	*in*	1
in	*in*	*def*	1

Semantics	Colouring	Runtime
Stable	$C_t : \chi(t) \mapsto \{in, def, out\}$	$3^{O(\text{treewidth})} \cdot \text{poly}(\lvert Args \rvert)$
Admissible	$C_t : \chi(t) \mapsto \{in, def, att, out\}$	$4^{O(\text{treewidth})} \cdot \text{poly}(\lvert Args \rvert)$
Complete	$C_t : \chi(t) \mapsto \{in, def, defp, out, outp\}$	$5^{O(\text{treewidth})} \cdot \text{poly}(\lvert Args \rvert)$

Choice of Benchmarks

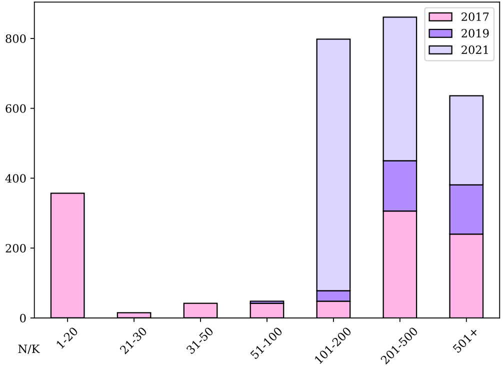


Figure: Distribution of heuristically computed widths. The x-axis lists intervals into which the heuristically computed width of a TD falls (K). The y-axis states the number (N) of instances.

Abstract Argumentation – Basics

Framework

An argumentation framework is a pair $F = (A, R)$ where

- A is a set of arguments and
- $R \subseteq A \times A$ is an attack relation.

Framework

Further, for $a, b \in A$ and $S, S' \subseteq A$ we denote the following attack relation $\rightarrowtail$:

- $a \rightarrowtail b$ if $(a, b) \in R$,
- $S \rightarrowtail a$ if there exists $b \in S$ s.t. $(b, a) \in R$,
- $a \rightarrowtail S$ if there exists $b \in S$ s.t. $(a, b) \in R$ and
- $S \rightarrowtail S'$ if there exists $a \in S$, $b \in S'$ s.t. $(a, b) \in R$.

and that:

- S is conflict-free if there are no $a, b \in S$ s.t. $(a, b) \in R$,
- S defends a if for each b s.t. $b \rightarrowtail a$, $S \rightarrowtail b$.

Abstract Argumentation – Semantics & Extensions

Semantics:

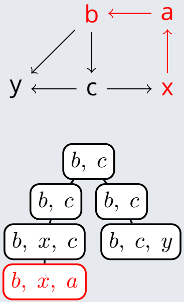
- are functions $\sigma : F \mapsto 2^A$, i.e. map the framework $F = (A, R)$ to set of subsets of A satisfying certain conditions,
- those subsets are called σ-extensions,
- in this work we focus on admissible, stable and complete semantics.

Semantics – definitions

Given an AF $F = (A, R)$, $a \in A$, $S \subseteq A$, S is a(n):

- admissible extension if it is conflict-free in F and each $a \in S$ is defended by S.
- stable extension if it is conflict-free in F and for each $a \in A \setminus S, S \rightarrowtail a$.
- complete extension if it is conflict-free in F and for each $a \in A$, if S defends a, then $a \in S$.

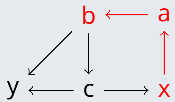
SQL – Admissible Semantics (1)

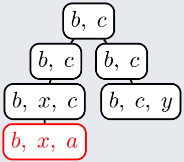


vi	*di*	*meaning*
0	*NULL*	*out*
0	0	*att*
0	1	*def*
1	–	*in*

```
SELECT vb, db, vx, dx, va, da,
    sum(model_count) AS model_count
FROM (WITH introduce AS
    (SELECT false val UNION ALL SELECT true)
SELECT ib.val vb, ix.val vx, ia.val va,
CASE WHEN ia.val THEN true
        WHEN (FALSE) AND (NOT ia.val) OR (FALSE)
            THEN false
        ELSE null::BOOLEAN
END AS db,
CASE WHEN FALSE THEN true
        WHEN (ia.val) AND (TRUE) OR (FALSE)
            THEN false
        ELSE null::BOOLEAN
END AS dx,
CASE WHEN ix.val THEN true
        WHEN (ib.val) AND (NOT ix.val) OR (FALSE)
            THEN false
        ELSE null::BOOLEAN
END AS da,
1 AS model_count
FROM introduce ix, /*introduce*/
introduce ib, introduce ia) AS candidate
WHERE (da IS NOT false) AND /*forget a*/
(NOT (va AND vb)) AND /*conflict-free*/
(NOT (vx AND va))
GROUP BY vb, vx, va, db, dx, da
```

SQL – Admissible Semantics (2)





vi	*di*	*meaning*
0	*NULL*	*out*
0	0	*att*
0	1	*def*
1	–	*in*

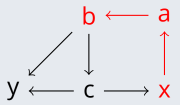
Query output

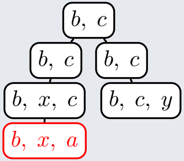
vb	*db*	*vx*	*dx*	*va*	*da*	#
false	*NULL*	*false*	*NULL*	*false*	*NULL*	1
false	*NULL*	*true*	–	*false*	*true*	1
false	*true*	*false*	*false*	1	–	1
true	–	*true*	–	*false*	*true*	1

Meaning

b	*x*	*a*
out	*out*	*out*
out	*in*	*def*
def	*att*	*in*
in	*in*	*def*

SQL – Complete Semantics (1)



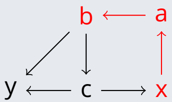


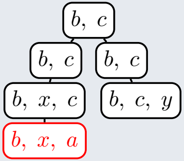
vi	*di*	*meaning*
0	*false*	*in*
1	*false*	*defp*
1	*true*	*def*
2	*false*	*outp*
2	*true*	*out*

```sql

SELECT vb,vx, va, db,dx,da,
    sum(model_count) AS model_count
FROM (WITH introduce AS
    (SELECT 0 val UNION ALL SELECT 1 UNION ALL SELECT 2)
SELECT ib.val vb,ix.val vx,ia.val va,
((ib.val = 1 AND (ia.val = 0)) OR (ib.val = 2 AND (ia.val = 2))) AS db,
(FALSE OR FALSE) AS dx,
((ia.val = 1 AND (ix.val = 0)) OR (ia.val = 2 AND (ix.val = 2))) AS da,
1 AS model_count
FROM introduce ia, /*introduce*/
introduce ib,introduce ix) AS candidate
WHERE (va = 0 OR da) AND  /*forget a*/
(NOT (va = 0 AND vb = 0)) AND
(NOT (vx = 0 AND va = 0)) AND  /*conflict-free*/
(NOT (va = 2 AND vb = 0)) AND
(NOT (vx = 2 AND va = 0)) AND
(NOT (va = 0 AND vb = 2)) AND
(NOT (vx = 0 AND va = 2))      /*colouring*/
GROUP BY vb,vx, va, db,dx,da
```

SQL – complete semantics (2)





vi	di	meaning
0	false	in
1	false	defp
1	true	def
2	false	outp
2	true	out

Query output

vb	db	vx	dx	va	da	#
1	false	0	false	1	true	1
1	true	1	false	0	false	1
1	false	2	false	2	true	1
2	true	2	false	2	true	1
0	false	0	false	1	true	1
2	false	0	false	1	true	1

Meaning

b	x	a
defp	in	def
def	defp	in
defp	outp	out
out	outp	out
in	in	def
outp	in	def

Stable Algorithm in Set Notation

Listing 1: Table algorithm $\mathbb{S}(t, \chi(t), F_t, \langle \tau_1, \ldots, \tau_\ell \rangle)$ for stable semantics on TDs.

In: Node t, bag $\chi(t)$, AF F_t, sequence $\langle \tau_1, \ldots, \tau_\ell \rangle$ of child tables. **Out:** Table τ_t.

if $\mathrm{type}(t) = $ *leaf* **then** $\tau_t := \{\langle \emptyset, 1 \rangle\}$
else if $\mathrm{type}(t) = $ *intr, and* $a \in \chi(t)$ *is introduced* **then**
  $\tau_t := \{\langle J \sqcup \{b \mapsto \mathrm{def} \mid b \in J^{\mathrm{out}}, J^{\mathrm{in}} \rightarrowtail b\}, c \rangle \mid \langle I, c \rangle \in \tau_1, J \in \{I^+_{a \mapsto \mathrm{in}}, I^+_{a \mapsto \mathrm{out}}\}, J^{\mathrm{in}} \not\rightarrowtail J^{\mathrm{in}}\}$
else if $\mathrm{type}(t) = $ *forget, and* $a \notin \chi(t)$ *is removed* **then**
  $\tau_t := \{\langle I^-_a, \Sigma_{\langle J, c \rangle \in \tau_1 : I^-_a = J^-_a, a \notin J^{\mathrm{out}}} c \rangle \mid \langle I, \cdot \rangle \in \tau_1, a \notin I^{\mathrm{out}}\}$
else if $\mathrm{type}(t) = $ *join* **then**
  $\tau_t := \{\langle I_1 \sqcup \{b \mapsto \mathrm{def} \mid b \in I_2^{\mathrm{def}}\}, c_1 \cdot c_2 \rangle \mid \langle I_1, c_1 \rangle \in \tau_1, \langle I_2, c_2 \rangle \in \tau_2, I_1^{\mathrm{in}} = I_2^{\mathrm{in}}\}$

$S^-_s := S \setminus \{s \mapsto \mathrm{in}, s \mapsto \mathrm{def}, s \mapsto \mathrm{out}\}$, $S^l := \{s \mid S(s) = l\}$, $S^+_s := S \cup \{s\}$, $S \sqcup D := \bigcup_{s \in \mathrm{dom}(S) \setminus \mathrm{dom}(D)} \{s \mapsto S(s)\} \cup D$.

Stable Semantics – Table Algorithm using Relational Algebra

Listing 2: Table algorithm $\mathbb{S}(t, \chi(t), F_t, \langle \tau_1, \ldots, \tau_\ell \rangle)$ for stable semantics.

In: Node t, bag $\chi(t)$, framework $F_t = (A_t, R_t)$, sequence $\langle \tau_1, \ldots, \tau_\ell \rangle$ of child tables. **Out:** Table τ_t.

if $\mathrm{type}(t) = $ *leaf* **then** $\tau_t := \{\{(\mathrm{cnt}, 1)\}\}$

else if $\mathrm{type}(t) = $ *intr, and $a \in \chi(t)$ is introduced* **then**

$\quad \tau_t := \dot{\Pi}_{\chi(t), \bigcup_{b \in \chi(t)} \{d_b \leftarrow d_b \vee (\neg b \wedge [\bigvee_{(c,b) \in R_t} c])\}} (\tau_1 \bowtie_{\bigwedge_{(b,c) \in R_t} \neg b \vee \neg c} \{\{(a,1), (d_a, 0)\}, \{(a,0), (d_a,0)\}\})$

else if $\mathrm{type}(t) = $ *forget, and $a \notin \chi(t)$ is removed* **then**

$\quad \tau_t := {}_{\{b, d_b \mid b \in \chi(t)\}} G_{\mathrm{cnt} \leftarrow \mathrm{SUM}(\mathrm{cnt})} (\Pi_{\mathrm{col}(\tau_1) \setminus \{a, d_a\}} (\sigma_{a \vee \neg d_a} (\tau_1)))$

else if $\mathrm{type}(t) = $ *join* **then**

$\quad \tau_t := \dot{\Pi}_{\chi(t), \bigcup_{b \in \chi(t)} \{\mathrm{cnt} \leftarrow \mathrm{cnt} \cdot \mathrm{cnt}', d_b \leftarrow d_b \vee d'_b\}} (\tau_1 \bowtie_{\bigwedge_{b \in \chi(t)} b = b'} \rho_{\bigcup_{x \in \mathrm{col}(\tau_2)} \{x \mapsto x'\}} \tau_2)$

Admissible Semantics – Table Algorithm in Relational Algebra

Listing 3: Table algorithm $\mathbb{A}(t, \chi(t), F_t, \langle \tau_1, \ldots, \tau_\ell \rangle)$ for admissible semantics.

In: Node t, bag $\chi(t)$, framework $F_t = (A_t, R_t)$, sequence $\langle \tau_1, \ldots, \tau_\ell \rangle$ of child tables. **Out:** Table τ_t.

if $\mathrm{type}(t) = $ *leaf* **then** $\tau_t := \{\{(\mathsf{cnt}, 1)\}\}$

else if $\mathrm{type}(t) = $ *intr, and* $a \in \chi(t)$ *is introduced* **then**

$\quad \tau_t := \dot{\Pi}_{\chi(t), \bigcup_{b \in \chi(t)} \{d_b \leftarrow \mathsf{df}_t(d_b, b)\}}(\tau_1 \bowtie_{\bigwedge_{(b,c) \in R_t} \neg b \vee \neg c} \{\{(a,1), (d_a, 0)\}, \{(a,0), (d_a, 0)\}\})$

else if $\mathrm{type}(t) = $ *forget, and* $a \notin \chi(t)$ *is removed* **then**

$\quad \tau_t := {}_{\{b, d_b \mid b \in \chi(t)\}} G_{\mathsf{cnt} \leftarrow \mathsf{SUM(cnt)}}(\Pi_{\mathrm{col}(\tau_1) \setminus \{a, d_a\}}(\sigma_{a \vee d_a = 1}(\tau_1)))$

else if $\mathrm{type}(t) = $ *join* **then**

$\quad \tau_t := \dot{\Pi}_{\chi(t), \bigcup_{b \in \chi(t)} \{\mathsf{cnt} \leftarrow \mathsf{cnt} \cdot \mathsf{cnt}', d_b \leftarrow \mathsf{jn}(d_b, d'_b)\}}(\tau_1 \bowtie_{\bigwedge_{b \in \chi(t)} b = b'} \rho_{\bigcup_{x \in \mathrm{col}(\tau_2)} \{x \mapsto x'\}} \tau_2)$

Let $\mathsf{jn}(d, e) := 2$ if $d = 2$ or $e = 2$; else 1 if $d = 1$ or $e = 1$; else 0, and $\mathsf{df}_t(d, b) := \mathsf{jn}(d, 2$ if $(\bigvee_{(c,b) \in R_t} c)$; else 1 if $(\bigvee_{(b,c) \in R_t} c)$; else $0)$.