# Counting Answer Sets of Disjunctive Answer Set Programs

Mohimenul Kabir[a], Supratik Chakraborty[b], and Kuldeep S. Meel[c,d]

[a]National University of Singapore
[b]Indian Institute of Technology, Bombay
[c]Georgia Institute of Technology
[d]University of Toronto

41st International Conference on Logic Programming (ICLP), Rende, Italy, 2025

# Answer Set Programming (ASP)

- Roots in logic programming and non-monotonic reasoning
- A rule-based language for problem encoding

$$\underbrace{h_1 \vee \ldots h_\ell}_{head} \leftarrow \underbrace{b_1, \ldots, b_k, \sim b_{k+1}, \ldots, \sim b_{k+m}.}_{body}$$

# Answer Set Programming (ASP)

- Roots in logic programming and non-monotonic reasoning
- A rule-based language for problem encoding

$$\underbrace{h_1 \vee \ldots h_\ell}_{head} \leftarrow \underbrace{b_1, \ldots, b_k, \sim b_{k+1}, \ldots, \sim b_{k+m}.}_{body}$$

- An ASP program $P \equiv$ set of rules
- Definition: Program $P$ is called *disjunctive* if $\exists r \in P$ s.t. $|\text{Head}(r)| > 1$ [EG95]; otherwise, program $P$ is called *normal*
- The model of $P$ is an *answer set* (denoted as $\text{AS}(P)$)

# Answer Set Programming (ASP)

- Roots in logic programming and non-monotonic reasoning
- A rule-based language for problem encoding

$$\underbrace{h_1 \vee \ldots h_\ell}_{head} \leftarrow \underbrace{b_1, \ldots, b_k, \sim b_{k+1}, \ldots, \sim b_{k+m}.}_{body}$$

- An ASP program $P \equiv$ set of rules
- Definition: Program $P$ is called *disjunctive* if $\exists r \in P$ s.t. $|\text{Head}(r)| > 1$ [EG95]; otherwise, program $P$ is called *normal*
- The model of $P$ is an *answer set* (denoted as $\text{AS}(P)$)
- Answer set programming is based on *Justification* — everything is false unless there are some justifications.
- Consider an ASP program, $P = \{\underbrace{a \leftarrow b.}_{r_1} \quad \underbrace{b \leftarrow a.}_{r_2} \quad \underbrace{a \vee s \leftarrow \top.}_{r_3} \quad \underbrace{s \leftarrow \sim t.}_{r_4}\}$
  - $\{s\} \in \text{AS}(P)$; since $s$ is justified by $r_3$ or $r_4$
  - $\{a, b, s\} \notin \text{AS}(P)$; since $a$ and $b$ are NOT justified

# Answer Set Programming (ASP)

- Roots in logic programming and non-monotonic reasoning
- A rule-based language for problem encoding

$$\underbrace{h_1 \vee \ldots h_\ell}_{head} \leftarrow \underbrace{b_1, \ldots, b_k, \sim b_{k+1}, \ldots, \sim b_{k+m}.}_{body}$$

- An ASP program $P \equiv$ set of rules
- Definition: Program $P$ is called *disjunctive* if $\exists r \in P$ s.t. $|\text{Head}(r)| > 1$ [EG95]; otherwise, program $P$ is called *normal*
- The model of $P$ is an *answer set* (denoted as $\text{AS}(P)$)
- Answer set programming is based on *Justification* — everything is false unless there are some justifications.
- Consider an ASP program, $P = \{\underbrace{a \leftarrow b.}_{r_1} \quad \underbrace{b \leftarrow a.}_{r_2} \quad \underbrace{a \vee s \leftarrow \top.}_{r_3} \quad \underbrace{s \leftarrow \sim t.}_{r_4}\}$
  - $\{s\} \in \text{AS}(P)$; since $s$ is justified by $r_3$ or $r_4$
  - $\{a, b, s\} \notin \text{AS}(P)$; since $a$ and $b$ are NOT justified
- Answer Set Counting: Given a (disjunctive) program $P$, find $|\text{AS}(P)|$

# State-of-the-art ASP Counters and Our Contribution

Normal Logic Program

Complexity: #P [J2006]

**Theory & Implementation**

JN2011;J2006;ACM+2015;FHM+17

EHK2024;KCM2024;FGH+2024

# State-of-the-art ASP Counters and Our Contribution

Normal Logic Program

Complexity: #P [J2006]

**Theory & Implementation**

JN2011;J2006;ACM$^+$2015;FHM$^+$17
EHK2024;KCM2024;FGH$^+$2024

Disjunctive Logic Program

Complexity: #·co-NP [FHM$^+$17]

**Only Theory**

HK2023

# State-of-the-art ASP Counters and Our Contribution

Normal Logic Program

Complexity: #P [J2006]

**Theory & Implementation**

JN2011;J2006;ACM+2015;FHM+17
EHK2024;KCM2024;FGH+2024

Disjunctive Logic Program

Complexity: #·co-NP [FHM+17]

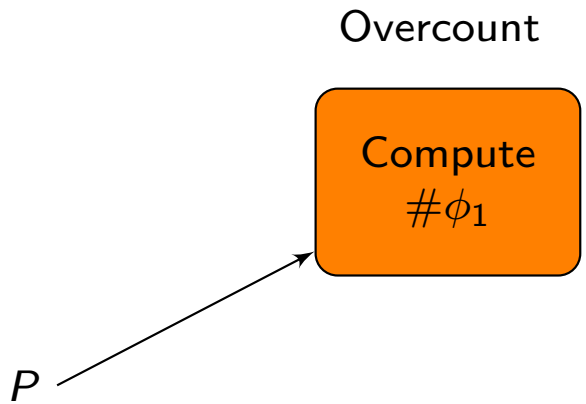
**Only Theory**

HK2023

Our Contribution: SharpASP-SR

- focus on both **theory** and **implementation**
- subtraction-based approach
- formulate to projected model counting

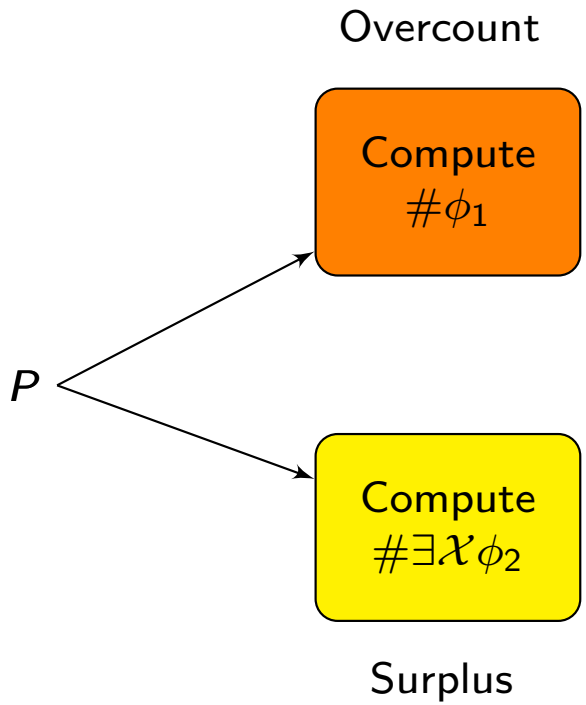
# SharpASP-SR: High-level Methodology




Given the disjunctive logic program $P$

- **Overcount**: Overcount the number of answer sets ($\#\phi_1$)

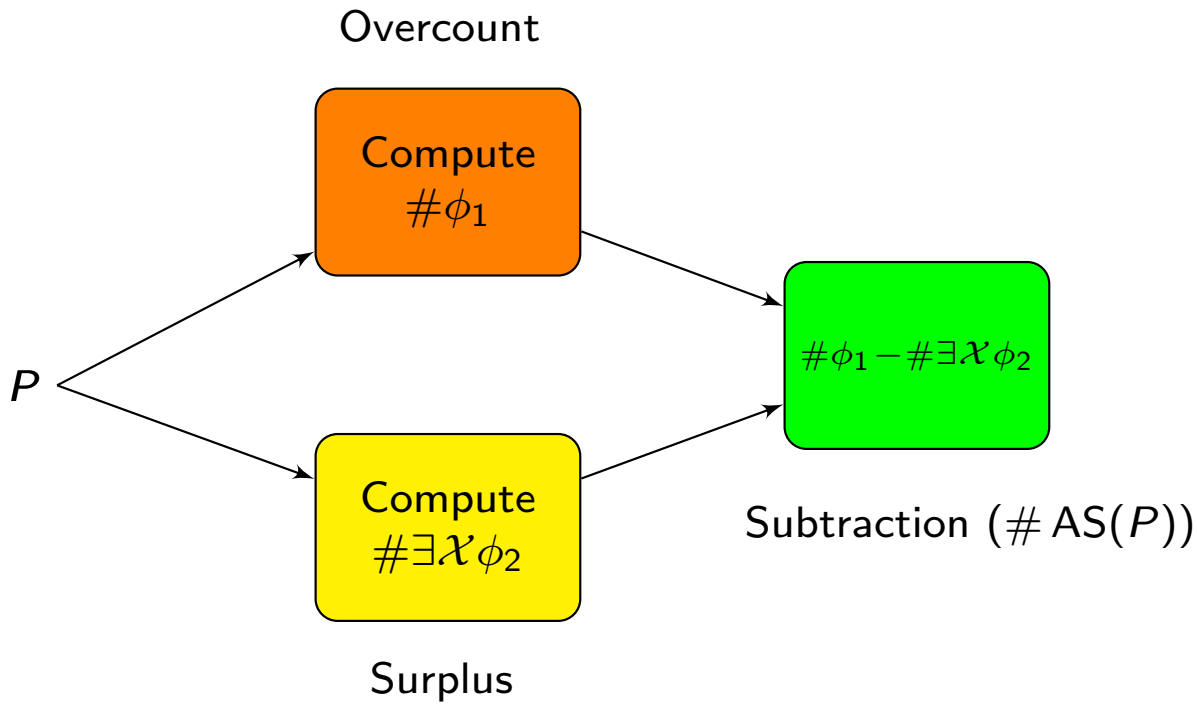
# SharpASP-SR: High-level Methodology




Given the disjunctive logic program $P$

- **Overcount**: Overcount the number of answer sets ($\#\phi_1$)
- **Surplus**: Carefully count the surplus ($\#\exists\mathcal{X}\phi_2$)

# SharpASP-SR: High-level Methodology




Given the disjunctive logic program $P$

- **Overcount**: Overcount the number of answer sets ($\#\phi_1$)
- **Surplus**: Carefully count the surplus ($\#\exists\mathcal{X}\phi_2$)
- **Subtraction**: $\#\phi_1 - \#\exists\mathcal{X}\phi_2$

# Overcount ($\phi_1$)

- Clark completion Comp($P$) provides a linear size translation to CNF [Clark1978]

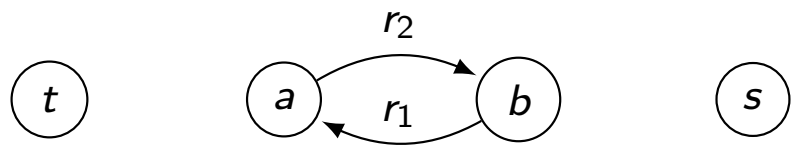
# Overcount ($\phi_1$)

- Clark completion Comp($P$) provides a linear size translation to CNF [Clark1978]
- Comp($P$) overcounts |AS($P$)| particularly for cyclic programs [LL2003]
- Cyclic programs: there exists some cyclic relations between program atoms

# Overcount ($\phi_1$)

- Clark completion Comp($P$) provides a linear size translation to CNF [Clark1978]
- Comp($P$) overcounts $|\mathrm{AS}(P)|$ particularly for cyclic programs [LL2003]
- Cyclic programs: there exists some cyclic relations between program atoms

Example: $P = \{\underset{r_1}{a \leftarrow b.} \quad \underset{r_2}{b \leftarrow a.} \quad \underset{r_3}{s \vee a \leftarrow \top.} \quad \underset{r_4}{s \leftarrow \sim t.}\}$.

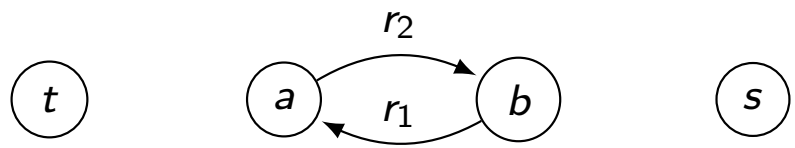



| atom | rule | Comp($P$) |
|---|---|---|
| $a$ | $a \leftarrow b,\ a \vee s \leftarrow \top$ | $a \longrightarrow (b \vee \neg s)$ |
| $b$ | $b \leftarrow a$ | $b \longrightarrow a$ |
| $s$ | $a \vee s \leftarrow \top,\ s \leftarrow \sim t$ | $s \longrightarrow (\neg a \vee \neg t)$ |
| $t$ | - | $\neg t$ |

# Overcount ($\phi_1$)

- Clark completion Comp($P$) provides a linear size translation to CNF [Clark1978]
- Comp($P$) overcounts $|\mathrm{AS}(P)|$ particularly for cyclic programs [LL2003]
- Cyclic programs: there exists some cyclic relations between program atoms

Example: $P = \{\underbrace{a \leftarrow b.}_{r_1} \quad \underbrace{b \leftarrow a.}_{r_2} \quad \underbrace{s \vee a \leftarrow \top.}_{r_3} \quad \underbrace{s \leftarrow \sim t.}_{r_4}\}$.

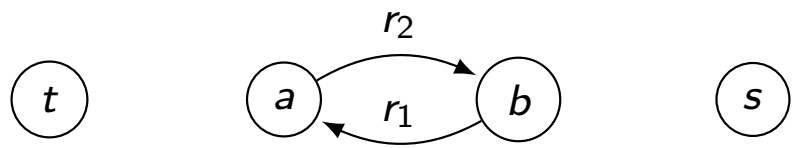



| atom | rule | Comp($P$) | rule | Comp($P$) |
|---|---|---|---|---|
| $a$ | $a \leftarrow b,\ a \vee s \leftarrow \top$ | $a \longrightarrow (b \vee \neg s)$ | $r_1 : a \leftarrow b$ | $b \longrightarrow a$ |
| $b$ | $b \leftarrow a$ | $b \longrightarrow a$ | $r_2 : b \leftarrow a$ | $a \longrightarrow b$ |
| $s$ | $a \vee s \leftarrow \top,\ s \leftarrow \sim t$ | $s \longrightarrow (\neg a \vee \neg t)$ | $r_3 : a \vee s \leftarrow \top$ | $a \vee s$ |
| $t$ | - | $\neg t$ | $r_4 : s \leftarrow \sim t$ | $\neg t \longrightarrow s$ |

# Overcount ($\phi_1$)

- Clark completion Comp($P$) provides a linear size translation to CNF [Clark1978]
- Comp($P$) overcounts $|\mathrm{AS}(P)|$ particularly for cyclic programs [LL2003]
- Cyclic programs: there exists some cyclic relations between program atoms

Example: $P = \{\underset{r_1}{a \leftarrow b.} \quad \underset{r_2}{b \leftarrow a.} \quad \underset{r_3}{s \vee a \leftarrow \top.} \quad \underset{r_4}{s \leftarrow \sim t.}\}$.



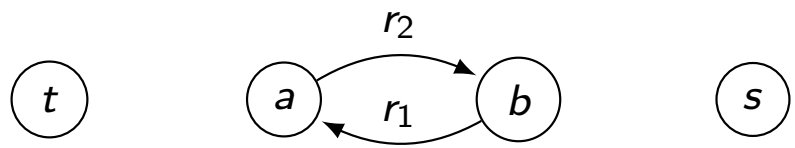

| atom | rule | Comp($P$) | rule | Comp($P$) |
|---|---|---|---|---|
| $a$ | $a \leftarrow b,\ a \vee s \leftarrow \top$ | $a \longrightarrow (b \vee \neg s)$ | $r_1 : a \leftarrow b$ | $b \longrightarrow a$ |
| $b$ | $b \leftarrow a$ | $b \longrightarrow a$ | $r_2 : b \leftarrow a$ | $a \longrightarrow b$ |
| $s$ | $a \vee s \leftarrow \top,\ s \leftarrow \sim t$ | $s \longrightarrow (\neg a \vee \neg t)$ | $r_3 : a \vee s \leftarrow \top$ | $a \vee s$ |
| $t$ | - | $\neg t$ | $r_4 : s \leftarrow \sim t$ | $\neg t \longrightarrow s$ |

- Two assignments ($\{s, \neg a, \neg b\}$ and $\{s, a, b\}$) satisfy Comp($P$)
- Program $P$ has one answer set ($\{s, \neg a, \neg b\}$)

# Overcount ($\phi_1$)

- Clark completion Comp($P$) provides a linear size translation to CNF [Clark1978]
- Comp($P$) overcounts |AS($P$)| particularly for cyclic programs [LL2003]
- Cyclic programs: there exists some cyclic relations between program atoms

Example: $P = \{\underset{r_1}{a \leftarrow b.} \quad \underset{r_2}{b \leftarrow a.} \quad \underset{r_3}{s \vee a \leftarrow \top.} \quad \underset{r_4}{s \leftarrow \sim t.}\}$.




| atom | rule | Comp($P$) | rule | Comp($P$) |
|---|---|---|---|---|
| $a$ | $a \leftarrow b,\ a \vee s \leftarrow \top$ | $a \longrightarrow (b \vee \neg s)$ | $r_1 : a \leftarrow b$ | $b \longrightarrow a$ |
| $b$ | $b \leftarrow a$ | $b \longrightarrow a$ | $r_2 : b \leftarrow a$ | $a \longrightarrow b$ |
| $s$ | $a \vee s \leftarrow \top,\ s \leftarrow \sim t$ | $s \longrightarrow (\neg a \vee \neg t)$ | $r_3 : a \vee s \leftarrow \top$ | $a \vee s$ |
| $t$ | - | $\neg t$ | $r_4 : s \leftarrow \sim t$ | $\neg t \longrightarrow s$ |

- Two assignments ($\{s, \neg a, \neg b\}$ and $\{s, a, b\}$) satisfy Comp($P$)
- Program $P$ has one answer set ($\{s, \neg a, \neg b\}$)
- Note that $\{\underset{\textit{justified}}{s, \neg a, \neg b}\}$ and $\{\ \underset{\textit{unjustified}}{s, a, b}\ \}$

# How to count unjustified ones?

- Answer Set definition says, EACH ATOM OF AN ANSWER SET MUST BE JUSTIFIED.
- Under Clark's completion, all *non-cyclic* atoms are *justified*.
  IT SUFFICES TO JUSTIFY CYCLIC ATOMS ONLY (we formally proved this).

# How to count unjustified ones?

- Answer Set definition says, EACH ATOM OF AN ANSWER SET MUST BE JUSTIFIED.
- Under Clark's completion, all *non-cyclic* atoms are *justified*.
  IT SUFFICES TO JUSTIFY CYCLIC ATOMS ONLY (we formally proved this).
- Example: Consider $P = \{\underset{r_1}{\underline{a \leftarrow b.}}\ \underset{r_2}{\underline{b \leftarrow a.}}\ \underset{r_3}{\underline{a \vee s \leftarrow \top.}}\ \underset{r_4}{\underline{s \leftarrow \sim t.}}\}$.
  - The cyclic atoms: $a$ and $b$; and the non-cyclic atoms: $s$ and $t$
  - For answer set $\{s\}$, the rules $r_3$ or $r_4$ justify the atom $s$
  - Since no rule justifies both atoms $a$ and $b$, $\{a, b, s\}$ is NOT an answer set.

# How to count unjustified ones?

- Answer Set definition says, EACH ATOM OF AN ANSWER SET MUST BE JUSTIFIED.
- Under Clark's completion, all *non-cyclic* atoms are *justified*.
  IT SUFFICES TO JUSTIFY CYCLIC ATOMS ONLY (we formally proved this).
- Example: Consider $P = \{\underbrace{a \leftarrow b.}_{r_1}\ \underbrace{b \leftarrow a.}_{r_2}\ \underbrace{a \vee s \leftarrow \top.}_{r_3}\ \underbrace{s \leftarrow \sim t.}_{r_4}\}$.
  - The cyclic atoms: $a$ and $b$; and the non-cyclic atoms: $s$ and $t$
  - For answer set $\{s\}$, the rules $r_3$ or $r_4$ justify the atom $s$
  - Since no rule justifies both atoms $a$ and $b$, $\{a, b, s\}$ is NOT an answer set.

**Key idea of Surplus**

Counting models of Comp($P$) s.t. at least one loop atoms is not justified

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. $x'$ *tells whether* $x$ *is justified or not*

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. $x'$ *tells whether* $x$ *is justified or not*
  - if $x = 0$, then $x'$ unit propagates to 0
  - if $x = 1$ and justified, then $x'$ unit propagates to 1

  if $x$ is unjustified, then $x'$ does NOT propagate

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. *$x'$ tells whether $x$ is justified or not*
  - if $x = 0$, then $x'$ unit propagates to 0
  - if $x = 1$ and justified, then $x'$ unit propagates to 1

  if $x$ is unjustified, then $x'$ does NOT propagate
- Example: $P = \{\underbrace{a \leftarrow b.}_{r_1} \quad \underbrace{b \leftarrow a.}_{r_2} \quad \underbrace{a \vee s \leftarrow \top.}_{r_3} \quad \underbrace{s \leftarrow \sim t.}_{r_4}\}$.

| atom | copy | $\mathcal{C}_1$ | |
|---|---|---|---|
| a | $\text{Copy}(a) = a'$ | $a' \longrightarrow a$ | if $x$ is 0 then $x'$ is 0 |

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. *$x'$ tells whether $x$ is justified or not*
  - if $x = 0$, then $x'$ unit propagates to 0
  - if $x = 1$ and justified, then $x'$ unit propagates to 1

  if $x$ is unjustified, then $x'$ does NOT propagate
- Example: $P = \{\underbrace{a \leftarrow b.}_{r_1} \quad \underbrace{b \leftarrow a.}_{r_2} \quad \underbrace{a \vee s \leftarrow \top.}_{r_3} \quad \underbrace{s \leftarrow \sim t.}_{r_4}\}$.

| atom | copy | $\mathcal{C}_1$ | |
|---|---|---|---|
| a | $\mathrm{Copy}(a) = a'$ | $a' \longrightarrow a$ | |
| b | $\mathrm{Copy}(b) = b'$ | $b' \longrightarrow b$ | if $x$ is 0 then $x'$ is 0 |

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. *$x'$ tells whether $x$ is justified or not*
  - if $x = 0$, then $x'$ unit propagates to 0
  - if $x = 1$ and justified, then $x'$ unit propagates to 1

  if $x$ is unjustified, then $x'$ does NOT propagate
- Example: $P = \{\underbrace{a \leftarrow b.}_{r_1} \quad \underbrace{b \leftarrow a.}_{r_2} \quad \underbrace{a \vee s \leftarrow \top.}_{r_3} \quad \underbrace{s \leftarrow \sim t.}_{r_4}\}$.

| atom | copy | $\mathcal{C}_1$ | |
|---|---|---|---|
| a | $\mathrm{Copy}(a) = a'$ | $a' \longrightarrow a$ | if $x$ is 0 then $x'$ is 0 |
| b | $\mathrm{Copy}(b) = b'$ | $b' \longrightarrow b$ | |
| s,t | - | - | |

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. $x'$ *tells whether* $x$ *is justified or not*
  - if $x = 0$, then $x'$ unit propagates to 0
  - if $x = 1$ and justified, then $x'$ unit propagates to 1

  if $x$ is unjustified, then $x'$ does NOT propagate
- Example: $P = \{\underbrace{a \leftarrow b.}_{r_1} \quad \underbrace{b \leftarrow a.}_{r_2} \quad \underbrace{a \vee s \leftarrow \top.}_{r_3} \quad \underbrace{s \leftarrow \sim t.}_{r_4}\}$.

| atom | copy | $\mathcal{C}_1$ | |
|---|---|---|---|
| a | $\text{Copy}(a) = a'$ | $a' \longrightarrow a$ | if $x$ is 0 then $x'$ is 0 |
| b | $\text{Copy}(b) = b'$ | $b' \longrightarrow b$ | |
| s,t | - | - | |

| rule | $\mathcal{C}_2$ |
|---|---|
| $r_1 : a \leftarrow b$ | $\text{Copy}(r_1) : b' \longrightarrow a'$ |

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. $x'$ *tells whether* $x$ *is justified or not*
  - if $x = 0$, then $x'$ unit propagates to 0
  - if $x = 1$ and justified, then $x'$ unit propagates to 1

  if $x$ is unjustified, then $x'$ does NOT propagate
- Example: $P = \{\underset{r_1}{\underline{a \leftarrow b.}} \quad \underset{r_2}{\underline{b \leftarrow a.}} \quad \underset{r_3}{\underline{a \vee s \leftarrow \top.}} \quad \underset{r_4}{\underline{s \leftarrow \sim t.}}\}$.

| atom | copy | $\mathcal{C}_1$ | |
|---|---|---|---|
| a | $\text{Copy}(a) = a'$ | $a' \longrightarrow a$ | if $x$ is 0 then $x'$ is 0 |
| b | $\text{Copy}(b) = b'$ | $b' \longrightarrow b$ | |
| s,t | - | - | |

| rule | $\mathcal{C}_2$ | |
|---|---|---|
| $r_1 : a \leftarrow b$ | $\text{Copy}(r_1) : b' \longrightarrow a'$ | |
| $r_2 : b \leftarrow a$ | $\text{Copy}(r_2) : a' \longrightarrow b'$ | |
| | | if $x$ is 1 and Jus then $x'$ is 1 |

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. $x'$ *tells whether* $x$ *is justified or not*
  - if $x = 0$, then $x'$ unit propagates to 0
  - if $x = 1$ and justified, then $x'$ unit propagates to 1

  if $x$ is unjustified, then $x'$ does NOT propagate
- Example: $P = \{\underbrace{a \leftarrow b.}_{r_1} \quad \underbrace{b \leftarrow a.}_{r_2} \quad \underbrace{a \vee s \leftarrow \top.}_{r_3} \quad \underbrace{s \leftarrow \sim t.}_{r_4}\}$.

| atom | copy | $\mathcal{C}_1$ | |
|---|---|---|---|
| a | $\text{Copy}(a) = a'$ | $a' \longrightarrow a$ | |
| b | $\text{Copy}(b) = b'$ | $b' \longrightarrow b$ | if $x$ is 0 then $x'$ is 0 |
| s,t | - | - | |

| rule | $\mathcal{C}_2$ | |
|---|---|---|
| $r_1 : a \leftarrow b$ | $\text{Copy}(r_1) : b' \longrightarrow a'$ | |
| $r_2 : b \leftarrow a$ | $\text{Copy}(r_2) : a' \longrightarrow b'$ | |
| $r_3 : a \vee s \leftarrow \top$ | $\text{Copy}(r_3) : a' \vee s$ | if $x$ is 1 and Jus then $x'$ is 1 |

# How to check "Justification" of atoms?

- Similar to [KCM2024], for each cyclic atom $x$, we introduce a new "Copy" variable $x'$ s.t. $x'$ *tells whether* $x$ *is justified or not*
  - if $x = 0$, then $x'$ unit propagates to 0
  - if $x = 1$ and justified, then $x'$ unit propagates to 1

  if $x$ is unjustified, then $x'$ does NOT propagate
- Example: $P = \{\underset{r_1}{\underline{a \leftarrow b.}} \quad \underset{r_2}{\underline{b \leftarrow a.}} \quad \underset{r_3}{\underline{a \vee s \leftarrow \top.}} \quad \underset{r_4}{\underline{s \leftarrow \sim t.}}\}$.

| atom | copy | $\mathcal{C}_1$ | |
|---|---|---|---|
| a | $\text{Copy}(a) = a'$ | $a' \longrightarrow a$ | if $x$ is 0 then $x'$ is 0 |
| b | $\text{Copy}(b) = b'$ | $b' \longrightarrow b$ | |
| s,t | - | - | |

| rule | $\mathcal{C}_2$ | |
|---|---|---|
| $r_1 : a \leftarrow b$ | $\text{Copy}(r_1) : b' \longrightarrow a'$ | if $x$ is 1 and Jus then $x'$ is 1 |
| $r_2 : b \leftarrow a$ | $\text{Copy}(r_2) : a' \longrightarrow b'$ | |
| $r_3 : a \vee s \leftarrow \top$ | $\text{Copy}(r_3) : a' \vee s$ | |
| $r_4 : s \leftarrow \sim t$ | - | |

$$\text{Copy}(P) = \mathcal{C}_1 \wedge \mathcal{C}_2$$

# How to check "Justification"?

Let $\tau \models \mathrm{Comp}(P)$

- (**Case** 1) $\tau$ is an answer set, then all cyclic atoms of $\tau$ are justified.
- (**Case** 2) $\tau$ is NOT an answer set, then some cyclic atoms of $\tau$ are NOT justified.

[1] $F\ |_\tau$ denotes the *unit propagation* of $\tau$ on $F$

# How to check "Justification"?

Let $\tau \models \mathrm{Comp}(P)$

- (**Case** 1) $\tau$ is an answer set, then all cyclic atoms of $\tau$ are justified.
- (**Case** 2) $\tau$ is NOT an answer set, then some cyclic atoms of $\tau$ are NOT justified.

Recall the example: $P = \{\underbrace{a \leftarrow b.}_{r_1}\ \underbrace{b \leftarrow a.}_{r_2}\ \underbrace{a \vee s \leftarrow \top.}_{r_3}\ \underbrace{s \leftarrow \sim t.}_{r_4}\}$.

- $\mathrm{Comp}(P) = (a \longrightarrow (b \vee \neg s)) \wedge (b \longrightarrow a) \wedge (s \longrightarrow (\neg a \vee \neg t)) \wedge (b \longrightarrow a) \wedge (a \longrightarrow b) \wedge (a \vee s) \wedge (\neg t \longrightarrow s) \wedge (\neg t)$
- $\mathrm{Copy}(P) = (a' \longrightarrow a) \wedge (b' \longrightarrow b) \wedge (a' \longrightarrow b') \wedge (b' \longrightarrow a') \wedge (a' \vee s)$

[1] $F\ |_\tau$ denotes the *unit propagation* of $\tau$ on $F$

# How to check "Justification"?

Let $\tau \models \mathrm{Comp}(P)$

- (**Case** 1) $\tau$ is an answer set, then all cyclic atoms of $\tau$ are justified.
- (**Case** 2) $\tau$ is NOT an answer set, then some cyclic atoms of $\tau$ are NOT justified.

Recall the example: $P = \{\underset{r_1}{a \leftarrow b.}\ \underset{r_2}{b \leftarrow a.}\ \underset{r_3}{a \vee s \leftarrow \top.}\ \underset{r_4}{s \leftarrow \sim t.}\}$.

- $\mathrm{Comp}(P) = (a \longrightarrow (b \vee \neg s)) \wedge (b \longrightarrow a) \wedge (s \longrightarrow (\neg a \vee \neg t)) \wedge (b \longrightarrow a) \wedge (a \longrightarrow b) \wedge (a \vee s) \wedge (\neg t \longrightarrow s) \wedge (\neg t)$
- $\mathrm{Copy}(P) = (a' \longrightarrow a) \wedge (b' \longrightarrow b) \wedge (a' \longrightarrow b') \wedge (b' \longrightarrow a') \wedge (a' \vee s)$

| $\tau$ | $\mathrm{Comp}(P)$ | $\mathrm{Copy}(P)\|_\tau$ | Remark |
|---|---|---|---|
| $\{s\} \in \mathrm{AS}(P)$ | $\models$ | $a' \mapsto 0, b' \mapsto 0$ | $a, b$ are justified |
| $\{a, b, s\} \notin \mathrm{AS}(P)$ | $\models$ | $(a' \longrightarrow b') \wedge (b' \longrightarrow a')$ | $a'$,$b'$ CAN be false |

[1] $F\ |_\tau$ denotes the *unit propagation* of $\tau$ on $F$

# How to check "Justification"?

Let $\tau \models \mathrm{Comp}(P)$

- (**Case** 1) $\tau$ is an answer set, then all cyclic atoms of $\tau$ are justified.
- (**Case** 2) $\tau$ is NOT an answer set, then some cyclic atoms of $\tau$ are NOT justified.

Recall the example: $P = \{\underbrace{a \leftarrow b.}_{r_1}\ \underbrace{b \leftarrow a.}_{r_2}\ \underbrace{a \vee s \leftarrow \top.}_{r_3}\ \underbrace{s \leftarrow \sim t.}_{r_4}\}$.

- $\mathrm{Comp}(P) = (a \longrightarrow (b \vee \neg s)) \wedge (b \longrightarrow a) \wedge (s \longrightarrow (\neg a \vee \neg t)) \wedge (b \longrightarrow a) \wedge (a \longrightarrow b) \wedge (a \vee s) \wedge (\neg t \longrightarrow s) \wedge (\neg t)$
- $\mathrm{Copy}(P) = (a' \longrightarrow a) \wedge (b' \longrightarrow b) \wedge (a' \longrightarrow b') \wedge (b' \longrightarrow a') \wedge (a' \vee s)$

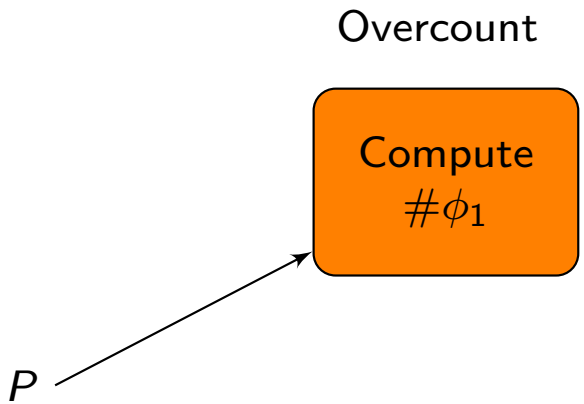
| $\tau$ | $\mathrm{Comp}(P)$ | $\mathrm{Copy}(P)\|_\tau$ | Remark |
|---|---|---|---|
| $\{s\} \in \mathrm{AS}(P)$ | $\models$ | $a' \mapsto 0, b' \mapsto 0$ | $a, b$ are justified |
| $\{a, b, s\} \notin \mathrm{AS}(P)$ | $\models$ | $(a' \longrightarrow b') \wedge (b' \longrightarrow a')$ | $a'$,$b'$ CAN be false |

**$\tau$ is a non-justified models of $\mathrm{Comp}(P)$:**[1]

Some loop atoms of $\tau$ can be assigned false while still satisfying $\mathrm{Copy}(P)\,|_\tau$

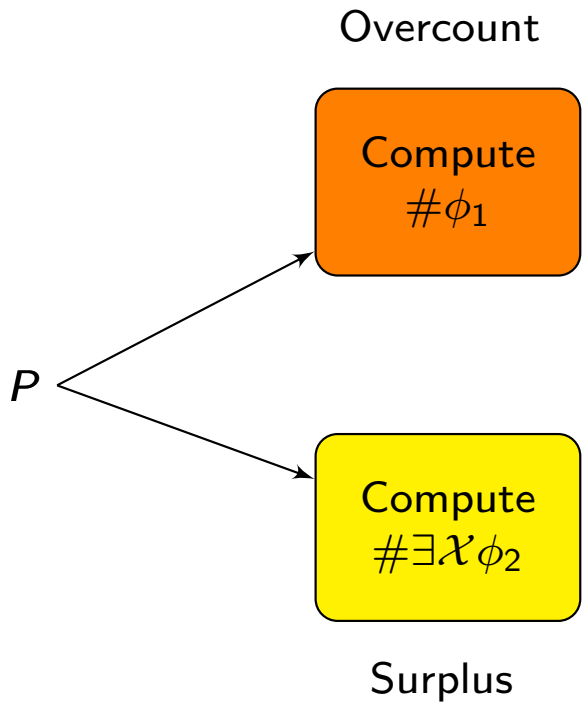
[1] $F\,|_\tau$ denotes the *unit propagation* of $\tau$ on $F$

# Compute $\phi_1, \phi_2$, and $\mathcal{X}$



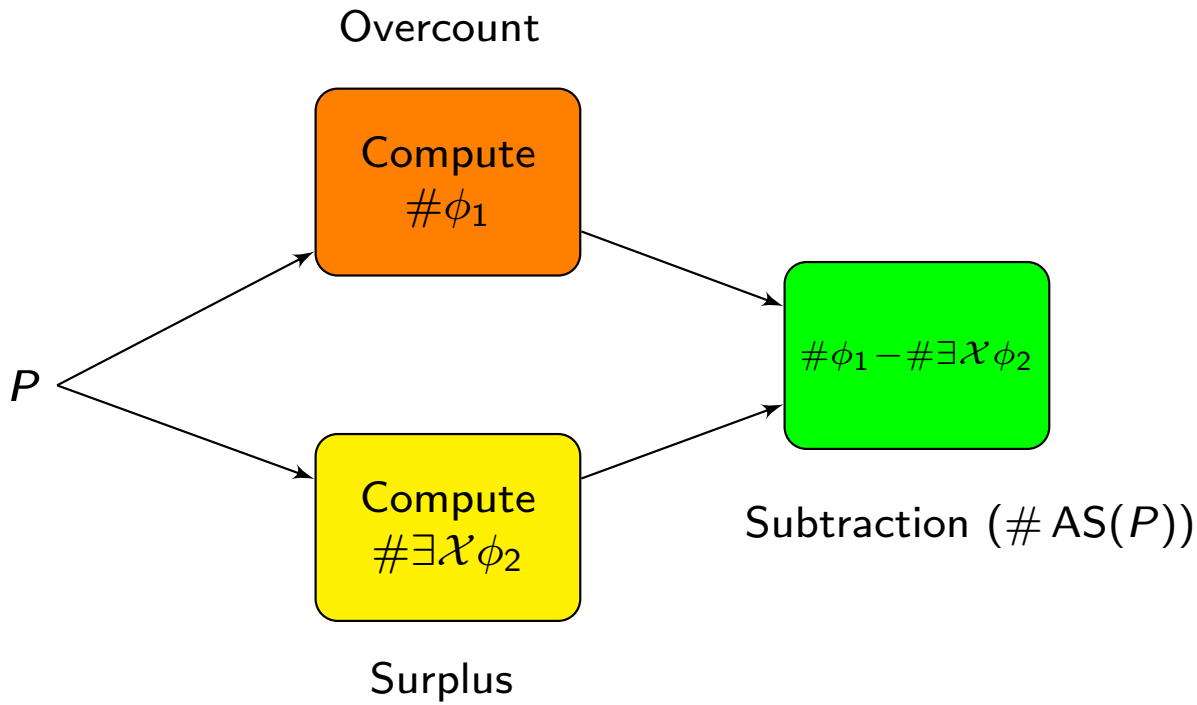

- Overcount: $\phi_1 = \mathsf{Comp}(P)$

# Compute $\phi_1, \phi_2$, and $\mathcal{X}$




- Overcount: $\phi_1 = \text{Comp}(P)$
- Surplus:
  $\phi_2 = \text{Comp}(P) \wedge \text{Copy}(P)' \wedge \text{Copy}(P)^\star \wedge \bigwedge_{x \in \text{LA}(P)}(x' \leq x^\star) \wedge \bigvee_{x \in \text{LA}(P)}(x' < x^\star)$,
  and $\mathcal{X} = \bigcup_{x \in \text{LA}(P)}\{x', x^\star\}$
  - $\text{LA}(P)$ denotes loop atoms of $P$
  - $\text{Copy}(P)'$ and $\text{Copy}(P)^\star$ are two sets of $\text{Copy}(P)$
  - $\bigwedge_{x \in \text{LA}(P)}(x' \leq x^\star) \wedge \bigvee_{x \in \text{LA}(P)}(x' < x^\star)$ enforces that at least one of the loop atoms can be set to false while satisfying $\text{Comp}(P)$ and $\text{Copy}(P)$

# Compute $\phi_1, \phi_2$, and $\mathcal{X}$




- Overcount: $\phi_1 = \text{Comp}(P)$
- Surplus:
  $\phi_2 = \text{Comp}(P) \wedge \text{Copy}(P)' \wedge \text{Copy}(P)^\star \wedge \bigwedge_{x \in \text{LA}(P)}(x' \leq x^\star) \wedge \bigvee_{x \in \text{LA}(P)}(x' < x^\star)$,
  and $\mathcal{X} = \bigcup_{x \in \text{LA}(P)}\{x', x^\star\}$
  - LA($P$) denotes loop atoms of $P$
  - Copy($P$)$'$ and Copy($P$)$^\star$ are two sets of Copy($P$)
  - $\bigwedge_{x \in \text{LA}(P)}(x' \leq x^\star) \wedge \bigvee_{x \in \text{LA}(P)}(x' < x^\star)$ enforces that at least one of the loop atoms can be set to false while satisfying Comp($P$) and Copy($P$)

# Experimental Evaluation

Benchmark sources:

| | |
|---|---|
| 1. QBF [KES+2022] | 2. strategic companies [L2005] |
| 3. abstract argumentation [GMR+2015] | 4. PC configuration [FGR2022] |
| 5. minimal diagnosis [GST+2008] | 6. system biology [TBP+2024] |
| 7. random instance generator [ART2017] | - |

# Experimental Evaluation

Benchmark sources:

| | |
|---|---|
| 1. QBF [KES+2022] | 2. strategic companies [L2005] |
| 3. abstract argumentation [GMR+2015] | 4. PC configuration [FGR2022] |
| 5. minimal diagnosis [GST+2008] | 6. system biology [TBP+2024] |
| 7. random instance generator [ART2017] | - |

Baselines:

| | | |
|---|---|---|
| 1. Clingo [GKS2012] | 2. Wasp [ADL+2015] | 3. DynASP [FHM+2017] |

# Experimental Evaluation

Benchmark sources:

| | |
|---|---|
| 1. QBF [KES+2022] | 2. strategic companies [L2005] |
| 3. abstract argumentation [GMR+2015] | 4. PC configuration [FGR2022] |
| 5. minimal diagnosis [GST+2008] | 6. system biology [TBP+2024] |
| 7. random instance generator [ART2017] | - |

Baselines:

| | | |
|---|---|---|
| 1. Clingo [GKS2012] | 2. Wasp [ADL+2015] | 3. DynASP [FHM+2017] |

Experimental setup:
A single core, with a time limit of 5000 seconds, a memory limit of 8 GB

# Experimental Results

PAR-2: Penalized average runtime that assigns 2× for each unsolved instances

| | clingo | DynASP | Wasp | SharpASP-SR |
|---|---|---|---|---|
| #Solved (1125) | 708 | 89 | 432 | **825** |
| PAR2 | 4118 | 9212 | 6204 | **2939** |

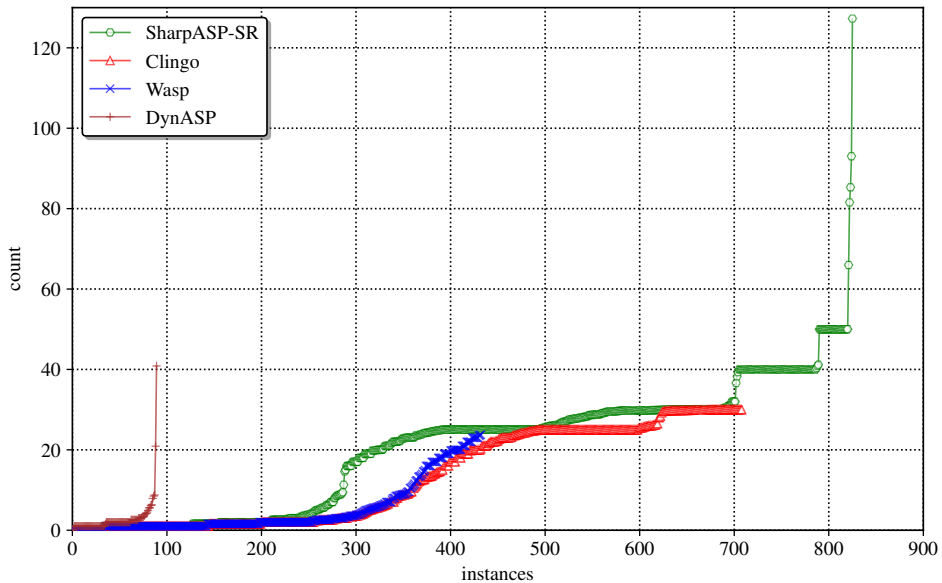
# Experimental Results

PAR-2: Penalized average runtime that assigns 2× for each unsolved instances

| | clingo | DynASP | Wasp | SharpASP-SR |
|---|---|---|---|---|
| #Solved (1125) | 708 | 89 | 432 | **825** |
| PAR2 | 4118 | 9212 | 6204 | **2939** |
| | | clingo ($\leq 10^4$) + | | |
| | clingo | DynASP | Wasp | SharpASP-SR |
| #Solved (1125) | 708 | 377 | 442 | **918** |
| PAR2 | 4118 | 4790 | 4404 | **1600** |

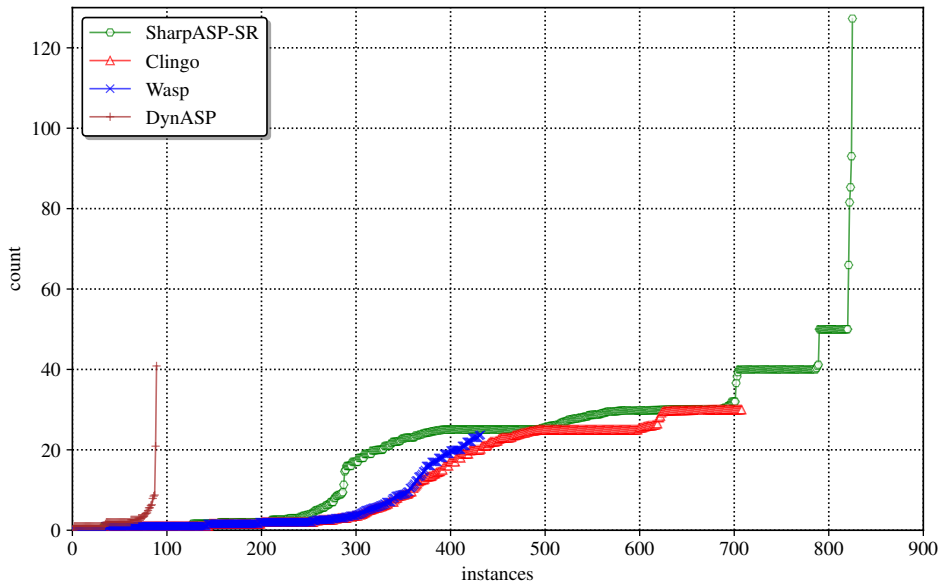
# Strengths and Weaknesses

👍 Strengths: SharpASP-SR counts instances with large number of answer sets



# Strengths and Weaknesses

👍 Strengths: SharpASP-SR counts instances with large number of answer sets




👎 Weaknesses: SharpASP-SR struggles with instances having a large $|\mathrm{LA}(P)|$

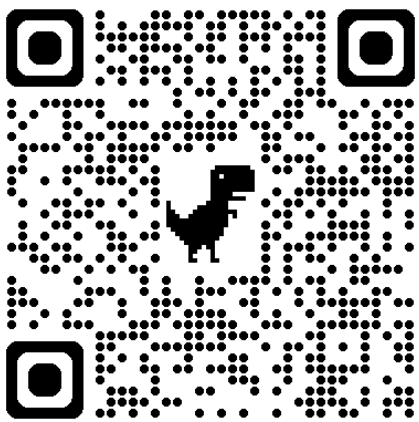
| $\lvert\mathrm{LA}(P)\rvert$ | $\sum$ | clingo | DynASP | Wasp | SharpASP-SR |
|---|---|---|---|---|---|
| $[1, 100]$ | 399 | 248 | 87 | 165 | **386** |
| $[101, 1000]$ | 519 | 316 | 2 | 142 | **398** |
| $> 1000$ | 207 | **144** | 0 | 125 | 41 |

# Conclusion

- We propose SharpASP-SR, a subtractive reduction based approach for answer set counting for disjunctive logic programs
- We reduce answer set counting to projected model counting by exploiting an alternative characterization of non-justified models
- Empirically SharpASP-SR outperforms existing counters on instances with large answer set counts

# Conclusion

- We propose SharpASP-SR, a subtractive reduction based approach for answer set counting for disjunctive logic programs
- We reduce answer set counting to projected model counting by exploiting an alternative characterization of non-justified models
- Empirically SharpASP-SR outperforms existing counters on instances with large answer set counts



`https://github.com/meelgroup/SharpASP-SR`