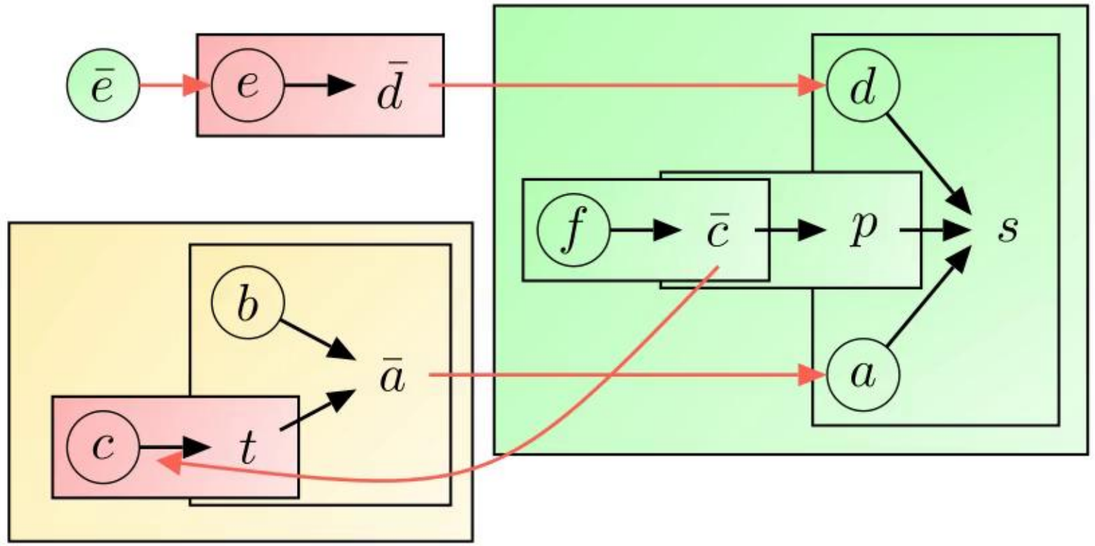
Technische Universität Dresden



ABA Disputes in ASP: Advancing Argument Games through Multi-Shot Solving

Martin Diller, **Piotr Gorczyca**

PRIMA 2025 – 16th December 2025, Modena, Italy

International Center for Computational Logic

Formal Argumentation: Two Perspectives

Model types:

Formal Argumentation: Two Perspectives

Technische Universität Dresden

Model types:

- **Abstract** (e.g. **Dung AFs**) – focus on relations between arguments

Formal Argumentation: Two Perspectives

Technische Universität Dresden

Model types:

- **Abstract** (e.g. **Dung AFs**) – focus on relations between arguments
- **Structured** (e.g. **ABA**) – include premises, rules and inference steps

Formal Argumentation: Two Perspectives

Technische Universität Dresden

Model types:

- **Abstract** (e.g. **Dung AFs**) – focus on relations between arguments
- **Structured** (e.g. **ABA**) – include premises, rules and inference steps

Reasoning views:

Formal Argumentation: Two Perspectives

Technische Universität Dresden

Model types:

- **Abstract** (e.g. **Dung AFs**) – focus on relations between arguments
- **Structured** (e.g. **ABA**) – include premises, rules and inference steps

Reasoning views:

- **Argumentation as inference** – decide acceptable arguments or claims

Formal Argumentation: Two Perspectives

Technische Universität Dresden

Model types:

- **Abstract** (e.g. **Dung AFs**) – focus on relations between arguments
- **Structured** (e.g. **ABA**) – include premises, rules and inference steps

Reasoning views:

- **Argumentation as inference** – decide acceptable arguments or claims
- **Argumentation as process** – argumentation as the reasoning mechanism itself

Formal Argumentation: Two Perspectives

Technische Universität Dresden

Model types:

- **Abstract** (e.g. **Dung AFs**) – focus on relations between arguments
- **Structured** (e.g. **ABA**) – include premises, rules and inference steps

Reasoning views:

- **Argumentation as inference** – decide acceptable arguments or claims
- **Argumentation as process** – argumentation as the reasoning mechanism itself

Argument games: discussion between proponent and opponent; provide dialectical justifications, support interaction, connected to dialogical models $\rightarrow$ useful for XAI

Bias Toward Inference-Based Approaches

Bias Toward Inference-Based Approaches

- Research on implementation methods arguably biased toward **abstract** and **inference**-focused models

Bias Toward Inference-Based Approaches

- Research on implementation methods arguably biased toward **abstract** and **inference**-focused models
- **ICCMA** focuses on decision problems and only added a structured (ABA) track in 2023

Bias Toward Inference-Based Approaches

Technische Universität Dresden

- Research on implementation methods arguably biased toward **abstract** and **inference**-focused models
- **ICCMA** focuses on decision problems and only added a structured (ABA) track in 2023

Some likely reasons:

Bias Toward Inference-Based Approaches

Technische Universität Dresden

- Research on implementation methods arguably biased toward **abstract** and **inference**-focused models
- **ICCMA** focuses on decision problems and only added a structured (ABA) track in 2023

Some likely reasons:

- Simplicity and popularity of Dung's AFs

Bias Toward Inference-Based Approaches

Technische Universität Dresden

- Research on implementation methods arguably biased toward **abstract** and **inference**-focused models
- **ICCMA** focuses on decision problems and only added a structured (ABA) track in 2023

Some likely reasons:

- Simplicity and popularity of Dung's AFs
- Reduction-based methods (**SAT/ASP** work especially well for inference)

Advancing Argument Games: Multi-Shot Solving

Advancing Argument Games: Multi-Shot Solving

- **ASP** widely used for implementing argumentation, but repeated grounding from scratch is costly in iterative scenarios

Advancing Argument Games: Multi-Shot Solving

Technische Universität Dresden

- **ASP** widely used for implementing argumentation, but repeated grounding from scratch is costly in iterative scenarios
- **Multi-shot ASP** allows updating and re-solving $\rightarrow$ avoids unnecessary grounding overhead

Advancing Argument Games: Multi-Shot Solving

Technische Universität Dresden

- **ASP** widely used for implementing argumentation, but repeated grounding from scratch is costly in iterative scenarios
- **Multi-shot ASP** allows updating and re-solving $\rightarrow$ avoids unnecessary grounding overhead
- We present the **first use of multi-shot ASP** for argumentation games

Advancing Argument Games: Multi-Shot Solving

Technische Universität Dresden

- **ASP** widely used for implementing argumentation, but repeated grounding from scratch is costly in iterative scenarios
- **Multi-shot ASP** allows updating and re-solving $\rightarrow$ avoids unnecessary grounding overhead
- We present the **first use of multi-shot ASP** for argumentation games
- Focus on **ABA dispute derivations** and **Dung's AFs**

Advancing Argument Games: Multi-Shot Solving

Technische Universität Dresden

- **ASP** widely used for implementing argumentation, but repeated grounding from scratch is costly in iterative scenarios
- **Multi-shot ASP** allows updating and re-solving $\rightarrow$ avoids unnecessary grounding overhead
- We present the **first use of multi-shot ASP** for argumentation games
- Focus on **ABA dispute derivations** and **Dung's AFs**
- **Rule-based flexible ABA disputes** from [Diller, Gaggl, Gorczyca 2021], elaboration on [Toni 2013], [Craven, Toni 2016]

Advancing Argument Games: Multi-Shot Solving

Technische Universität Dresden

- **ASP** widely used for implementing argumentation, but repeated grounding from scratch is costly in iterative scenarios
- **Multi-shot ASP** allows updating and re-solving $\rightarrow$ avoids unnecessary grounding overhead
- We present the **first use of multi-shot ASP** for argumentation games
- Focus on **ABA dispute derivations** and **Dung's AFs**
- **Rule-based flexible ABA disputes** from [Diller, Gaggl, Gorczyca 2021], elaboration on [Toni 2013], [Craven, Toni 2016]
- **Multi-shot environment** provided by `clingo` [Gebser et al. 2019]

Advancing Argument Games: Multi-Shot Solving

Next:

Advancing Argument Games: Multi-Shot Solving

Technische Universität Dresden

Next:

1. Multi-shot ASP for Dung's AFs

Advancing Argument Games: Multi-Shot Solving

Technische Universität Dresden

Next:

1. Multi-shot ASP for Dung's AFs
2. Extension to ABA disputes

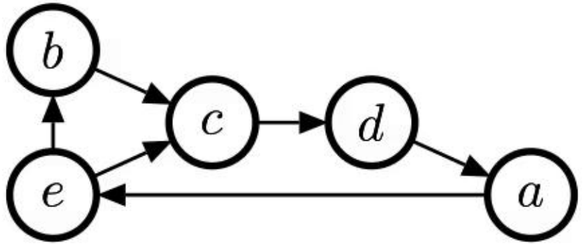
Advancing Argument Games: Multi-Shot Solving

Technische Universität Dresden

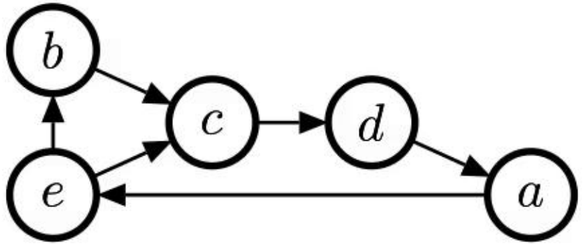
Next:

1. Multi-shot ASP for Dung's AFs
2. Extension to ABA disputes
3. Empirical evaluation

Multi-shot ASP for Dung's AFs



Multi-shot ASP for Dung's AFs




```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

Multi-shot ASP for Dung's AFs

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

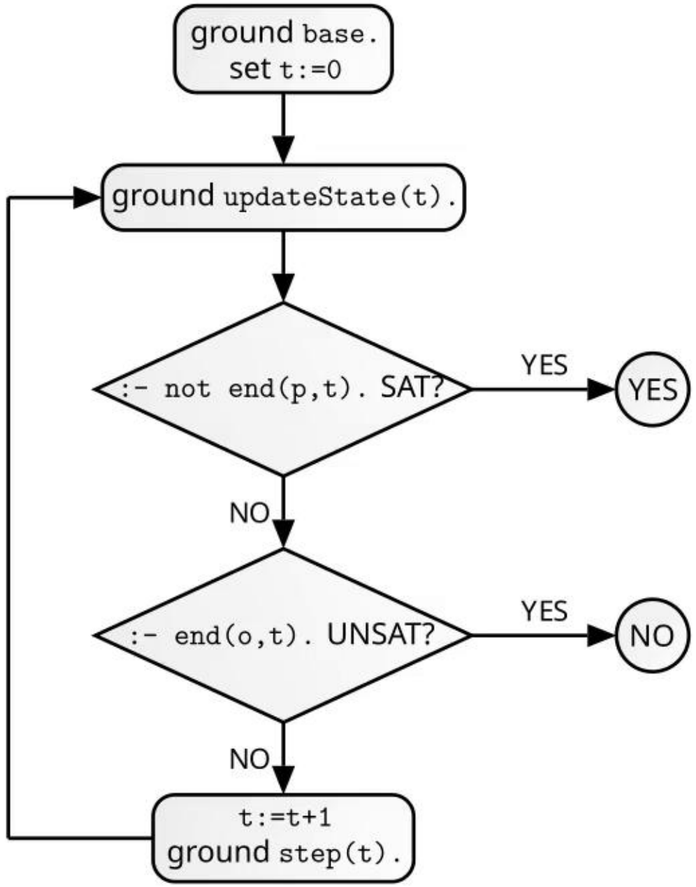
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
from clingo import Control, Number as N
                   Function as F
def main(instance, encoding):
 ctl = Control()
 ctl.load(instance)
 ctl.load(encoding)
 ctl.ground([("base", ())])
 t = 0
 while True:
  ctl.ground([("updateState",[N(t)])])
  p_win = (F("end",[N(t),F("p")]),True)
  res = ctl.solve(assumptions=[p_win],
                  on_model=print)
  if res.satisfiable:
    return True
  o_win = (F("end",[N(t),F("o")]),False)
  res = ctl.solve(assumptions=[o_win])
  if res.unsatisfiable:
    return False
  t += 1
  ctl.ground([("step",[N(t)])])
```

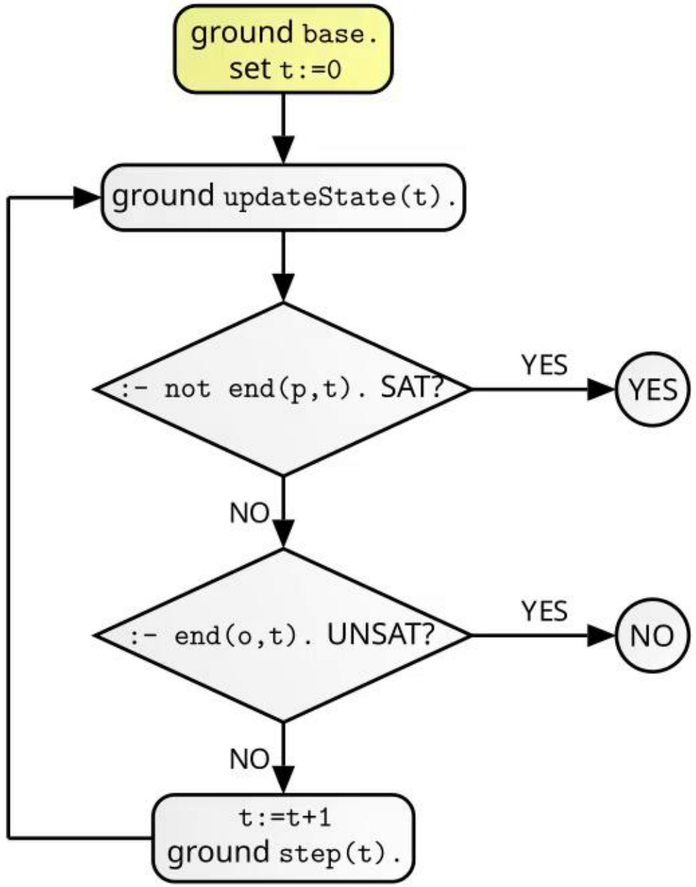
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



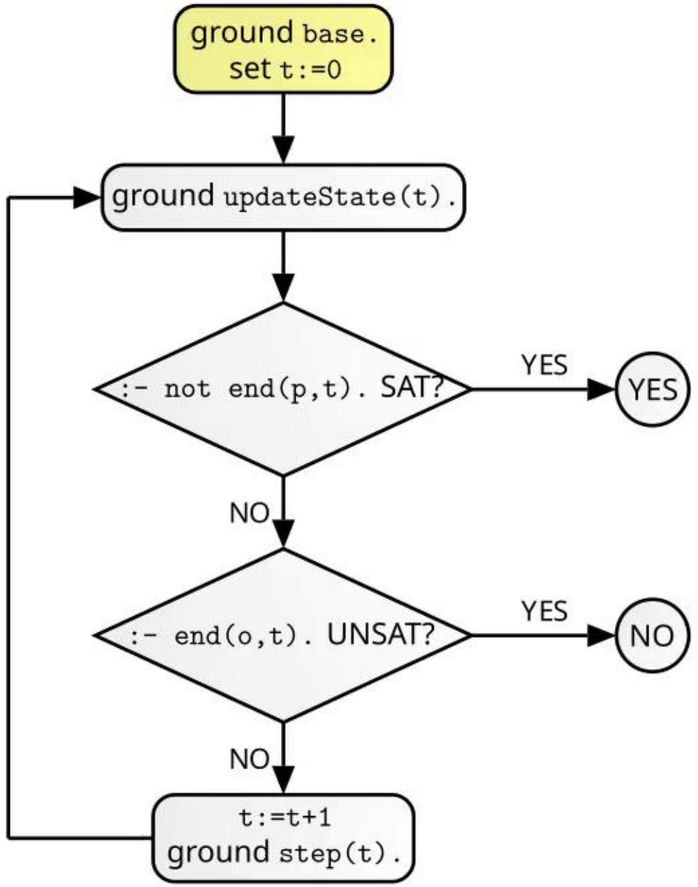
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

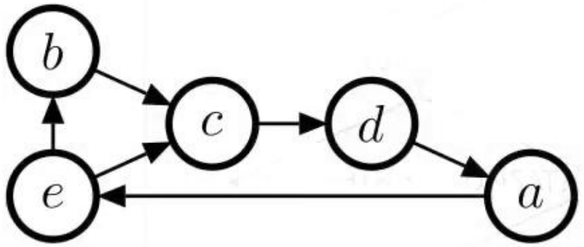
```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

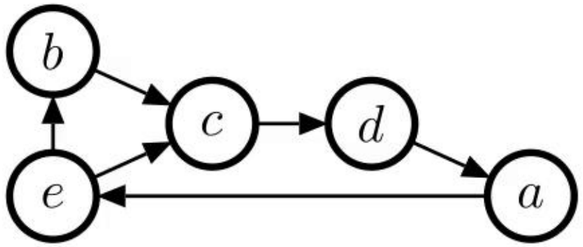
```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

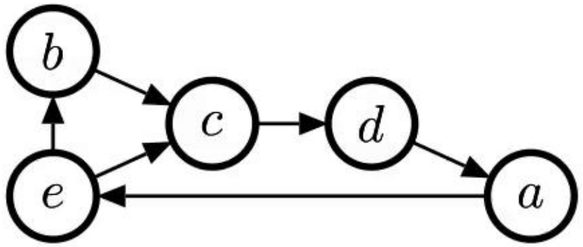


Output:

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

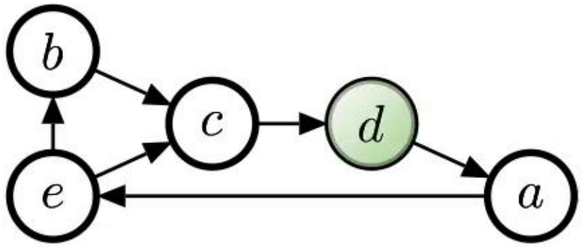


Output:

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

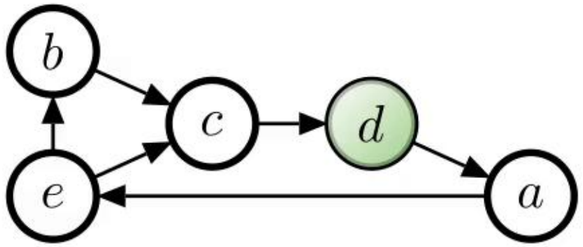


Output:

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

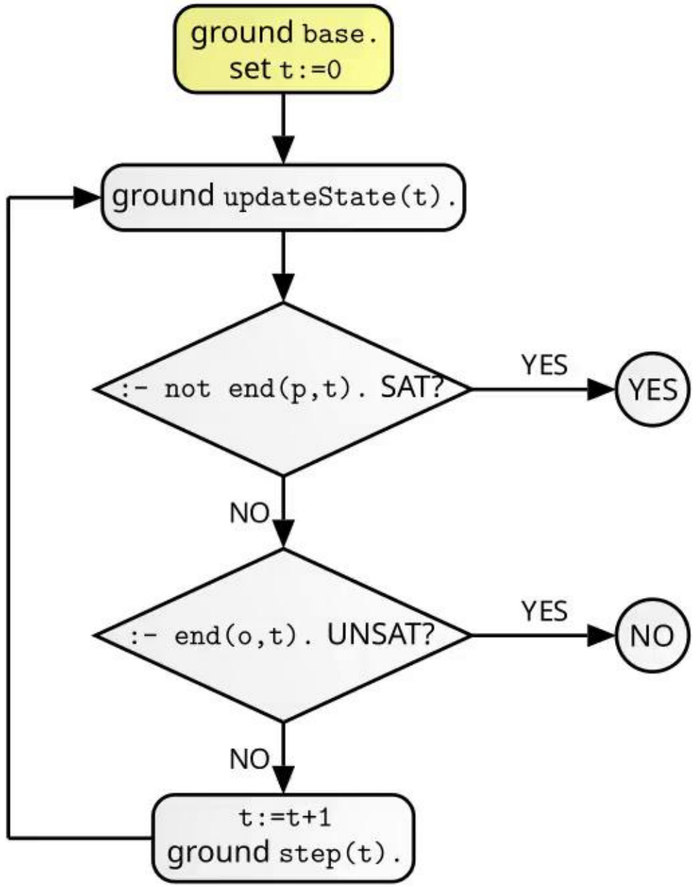
```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
`m(0,p,d).`

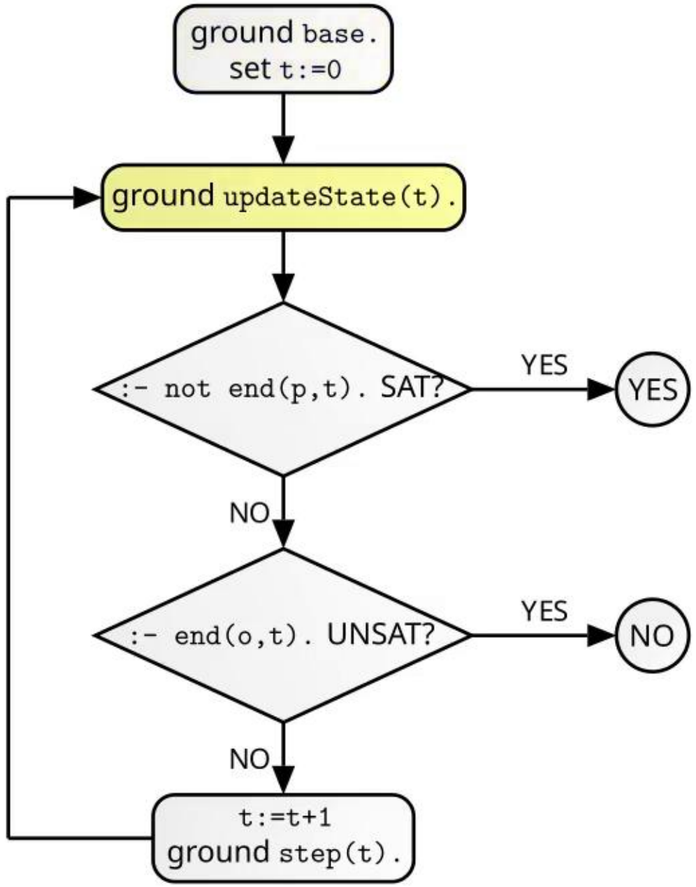
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



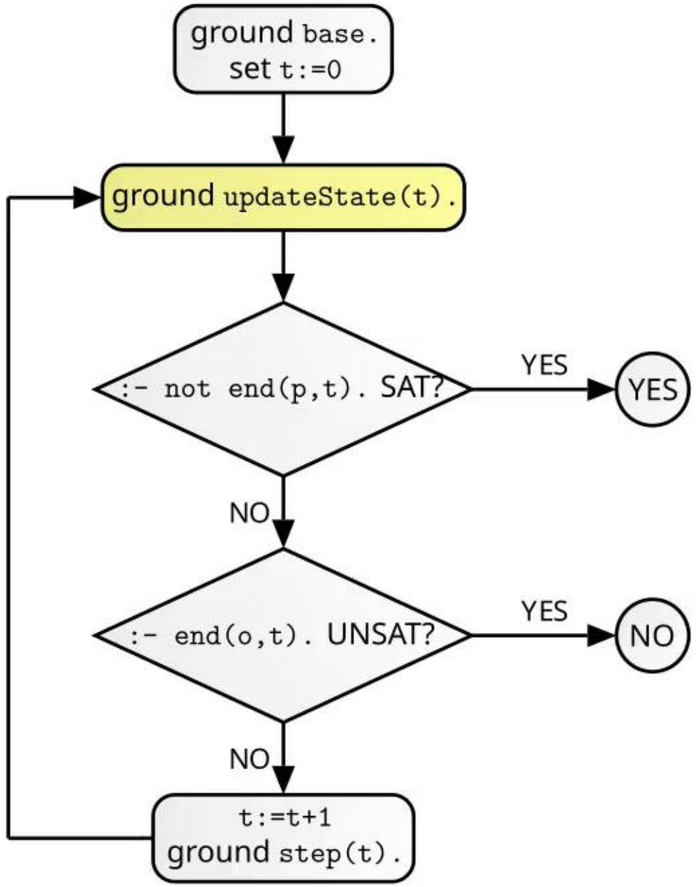
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

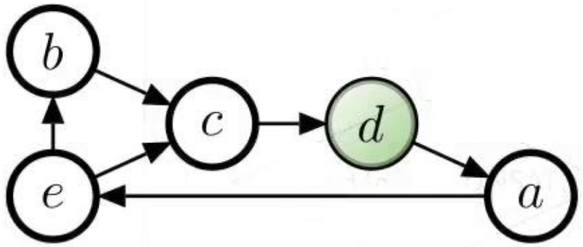
```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

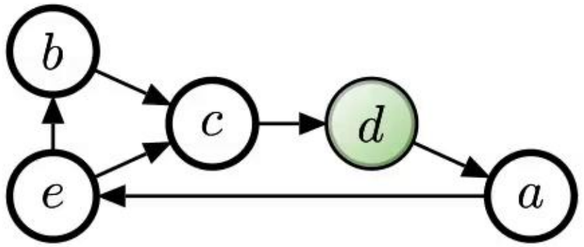


Output:
`m(0,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

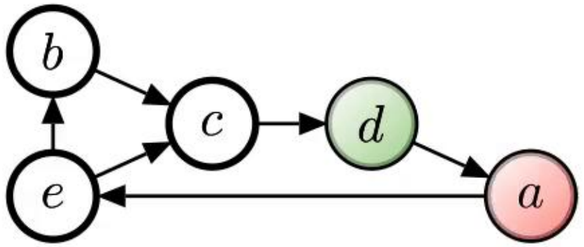


Output:
`m(0,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

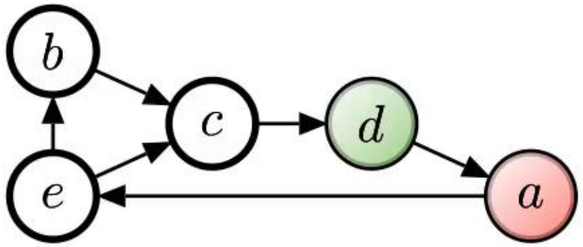


Output:
`m(O,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

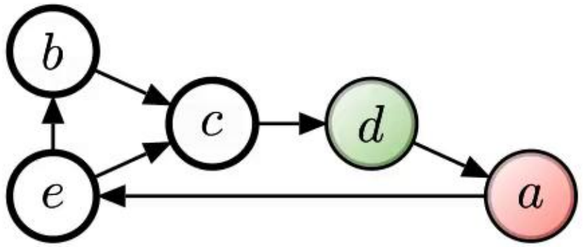


Output:
`m(0,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

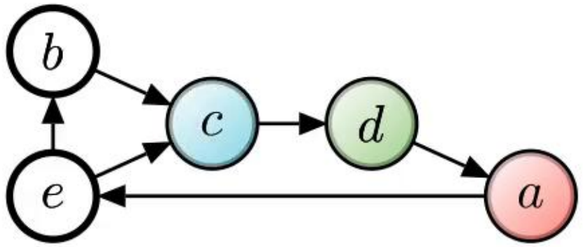


Output:
`m(0,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

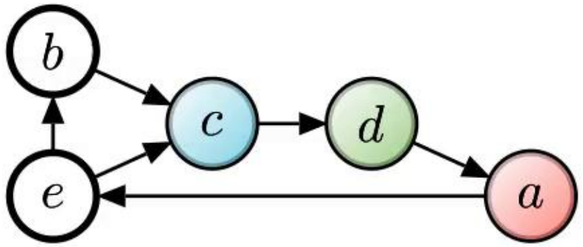


Output:
`m(O,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

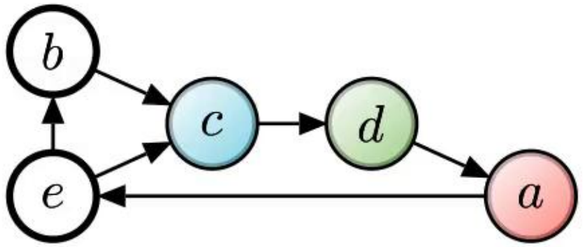


Output:
`m(0,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

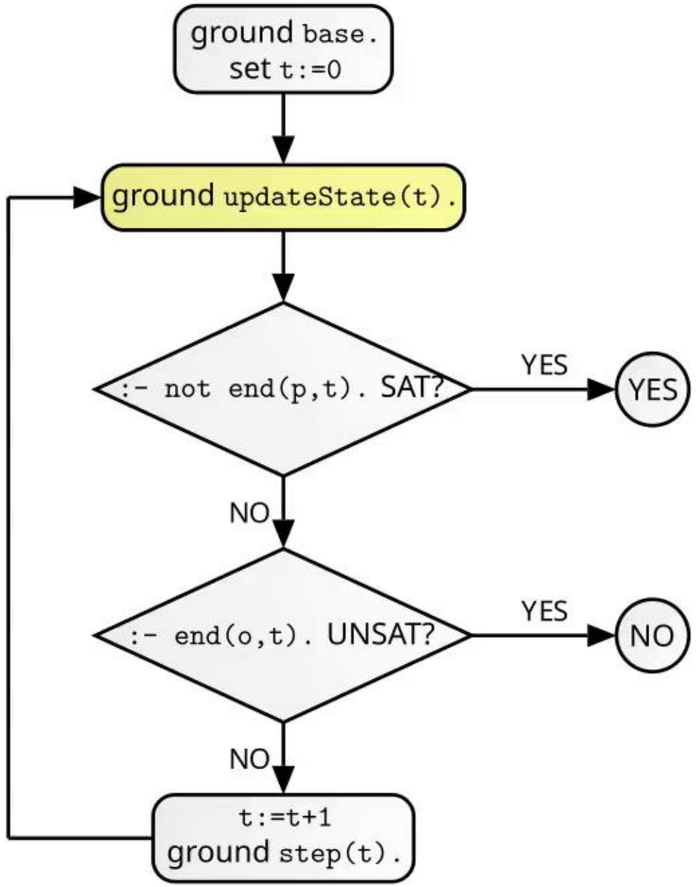
```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
`m(0,p,d).`

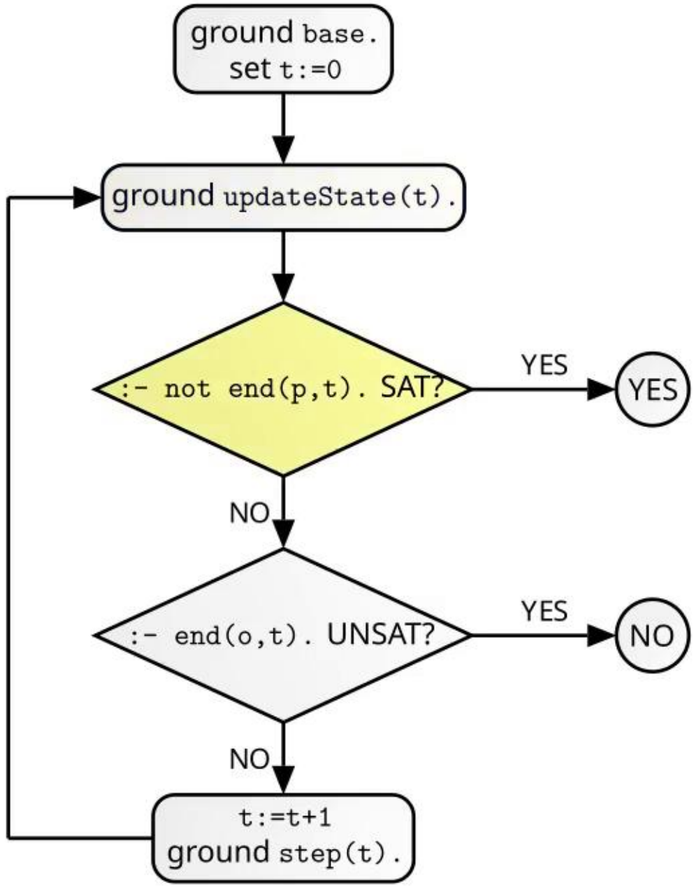
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



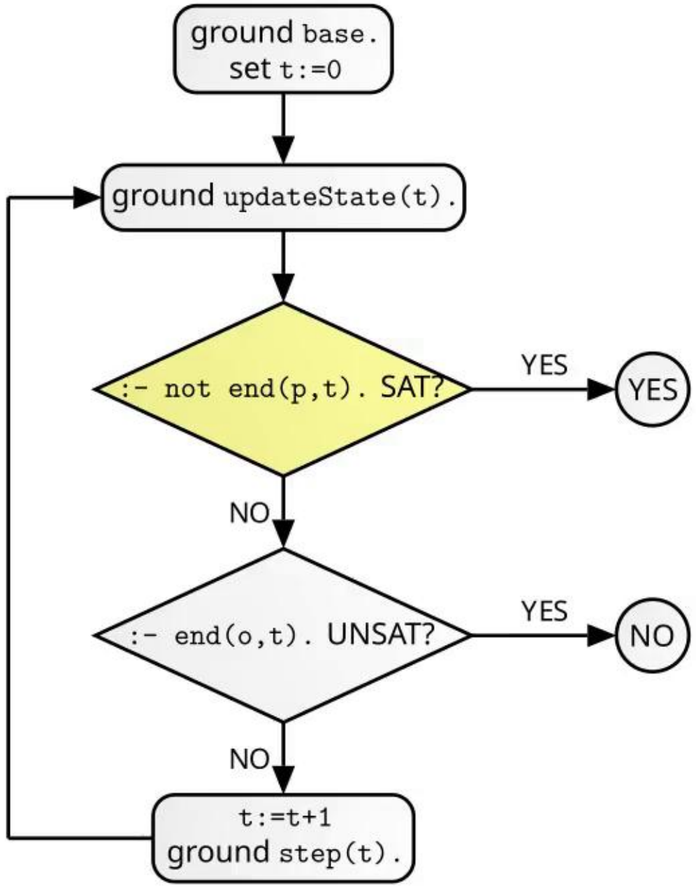
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

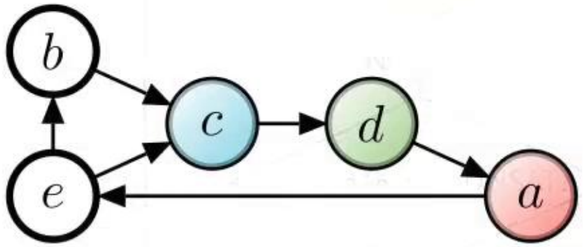
```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

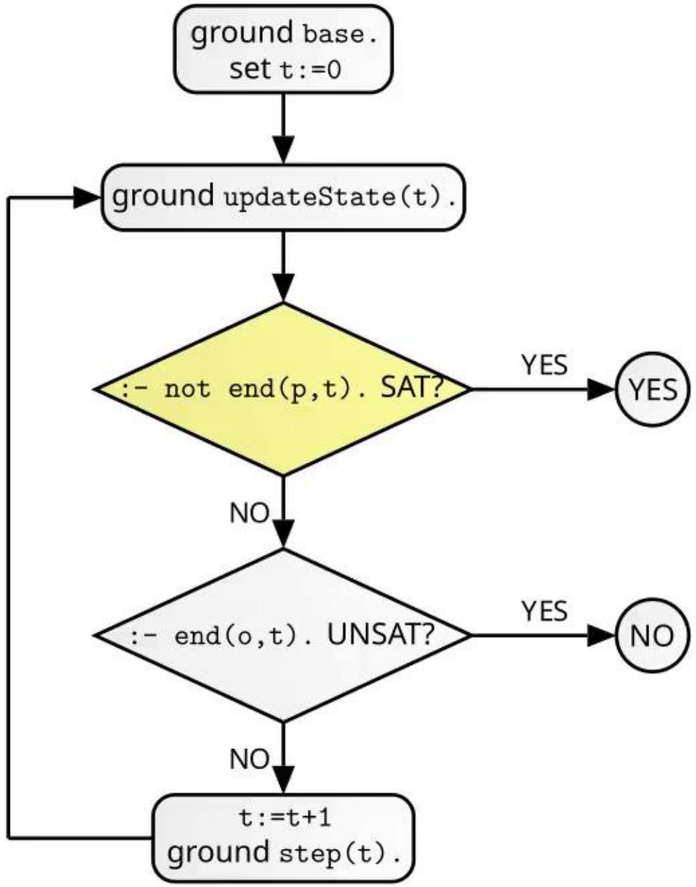
```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
`m(0,p,d).`

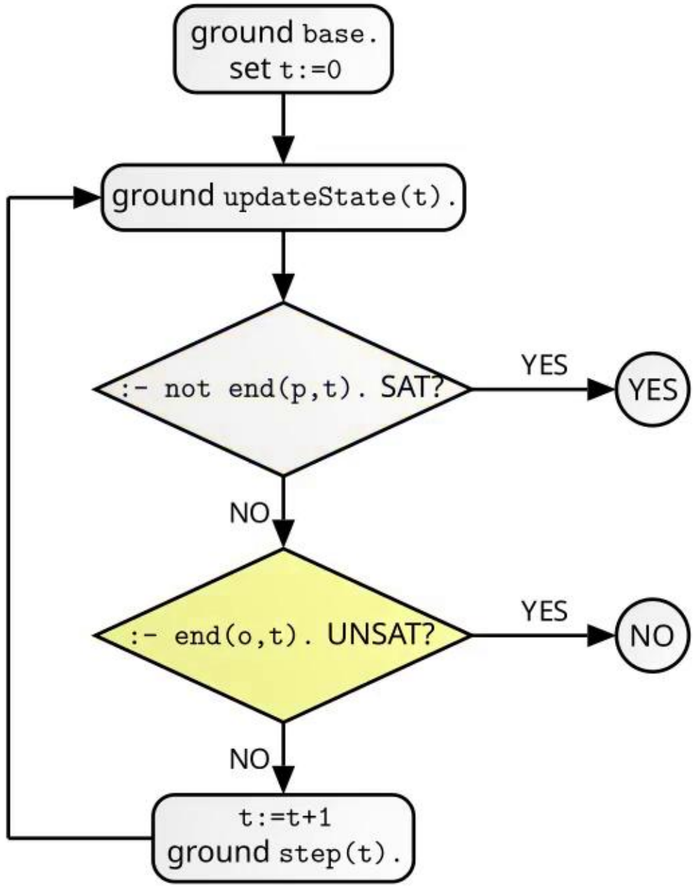
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



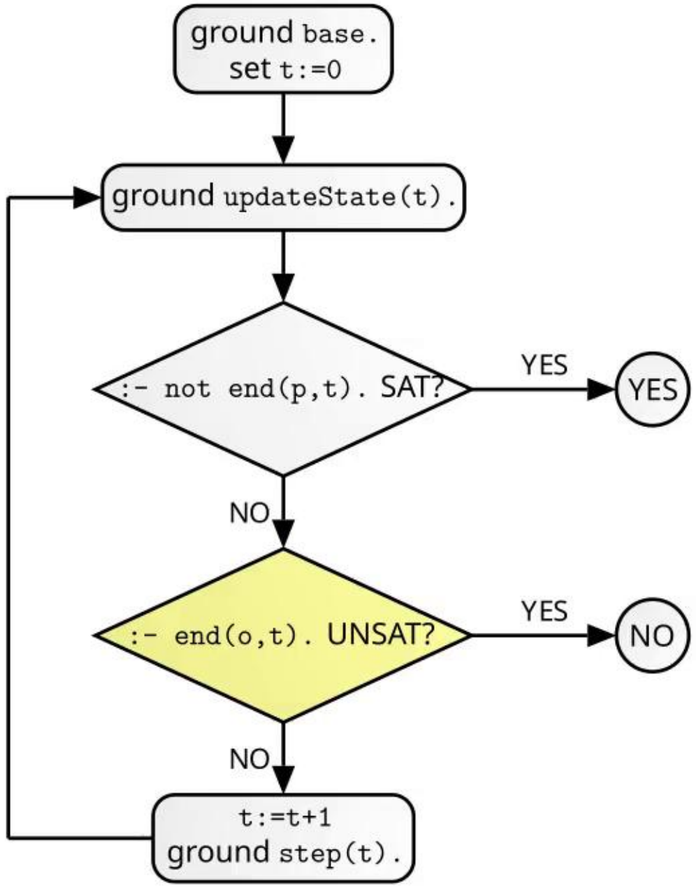
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

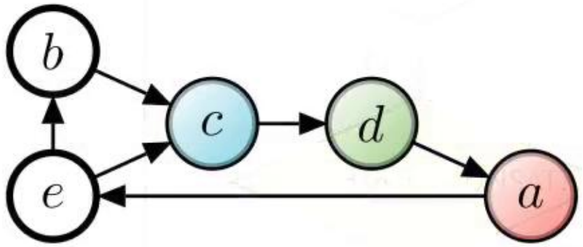
```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

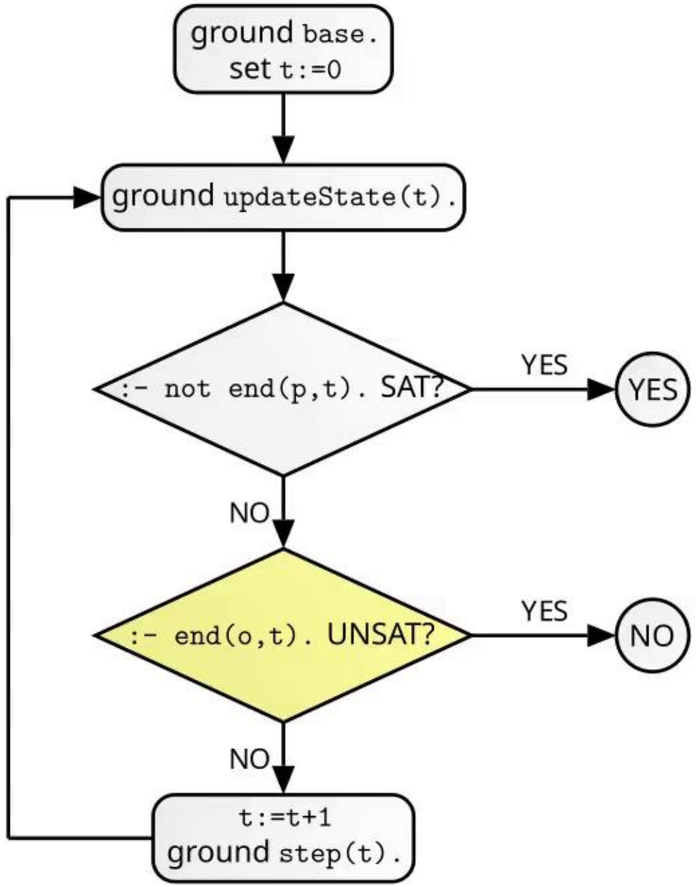
```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
`m(0,p,d).`

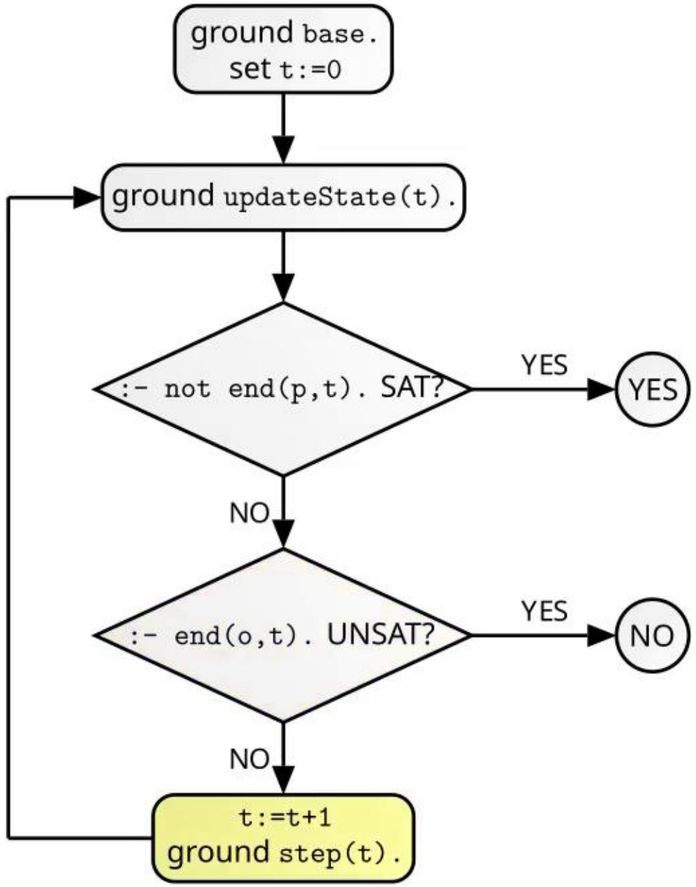
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



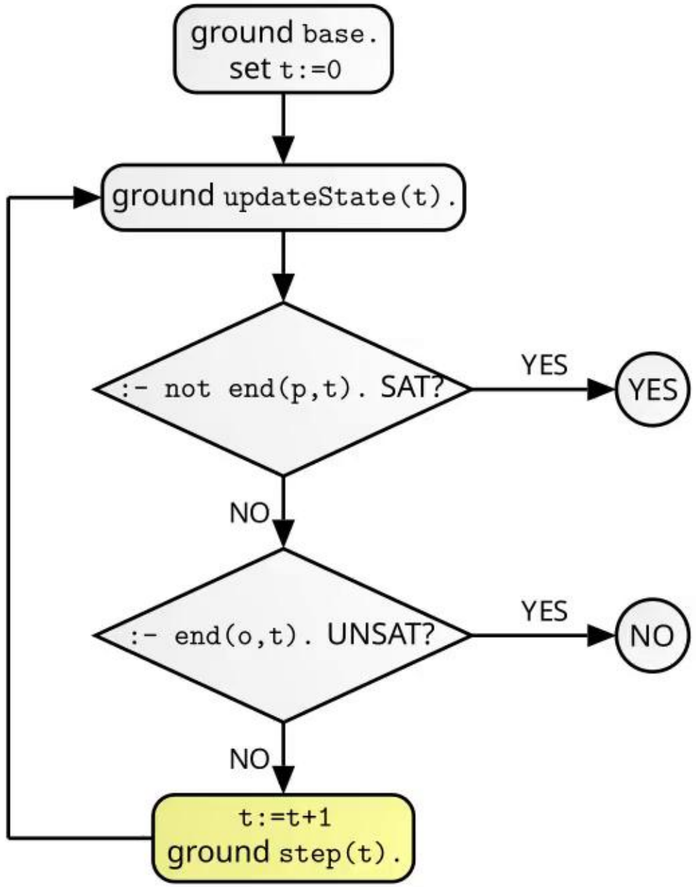
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

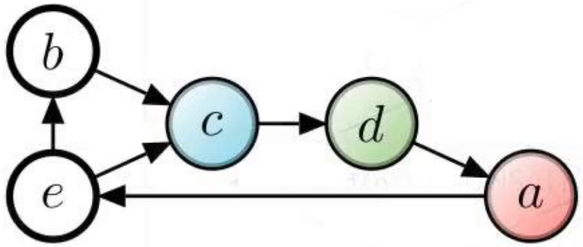
```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

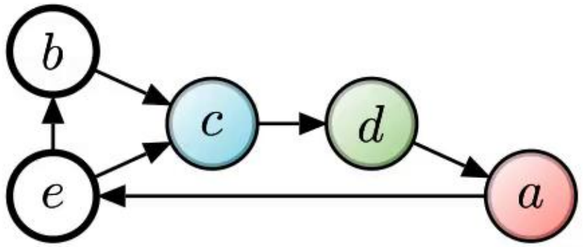


Output:
`m(0,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

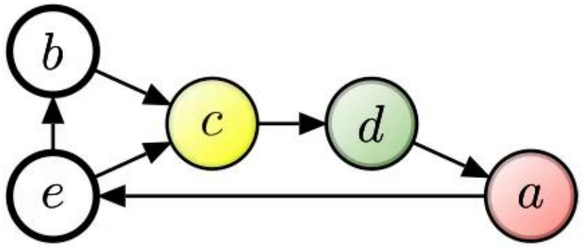


Output:
`m(0,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

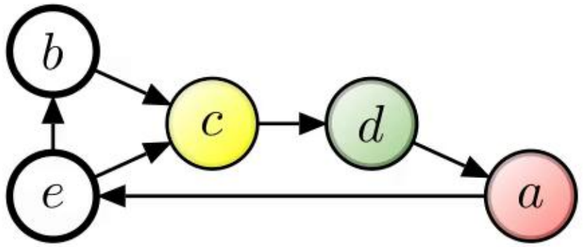


Output:
`m(0,p,d).`

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



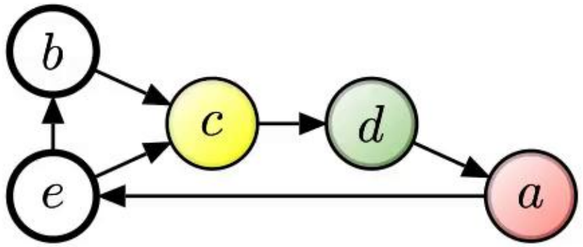
Output:

```
m(0,p,d).
m(1,o,c).
```

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```

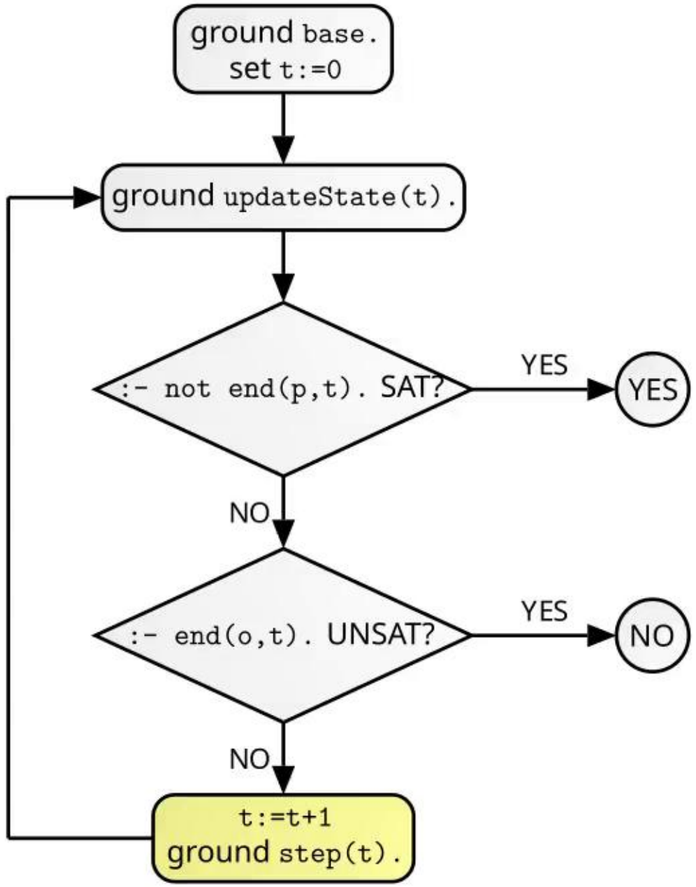


Output:

```
m(0,p,d).
m(1,o,c).
```

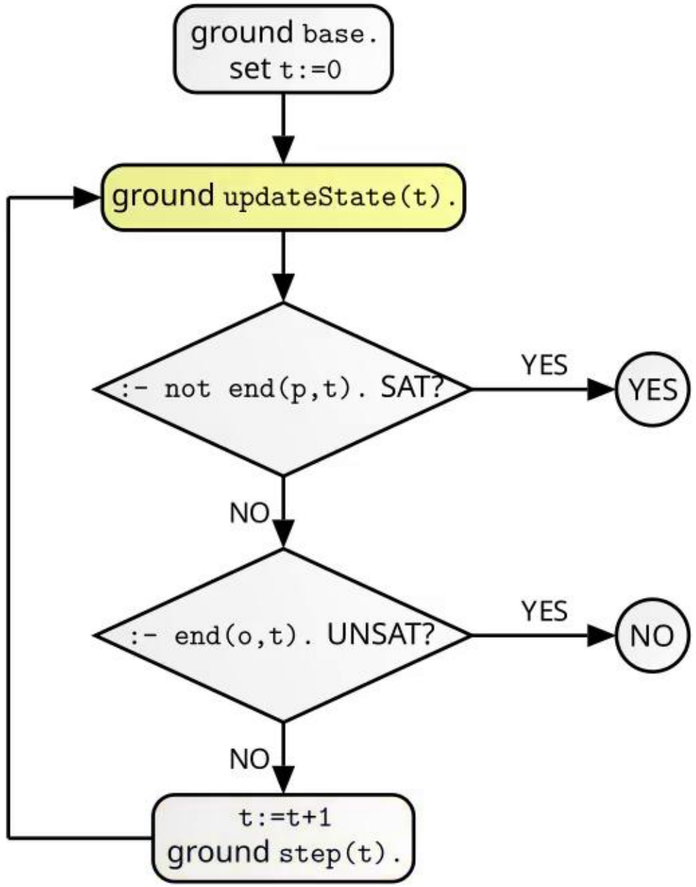
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

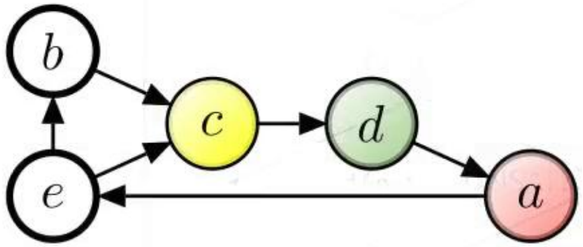
```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```



Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



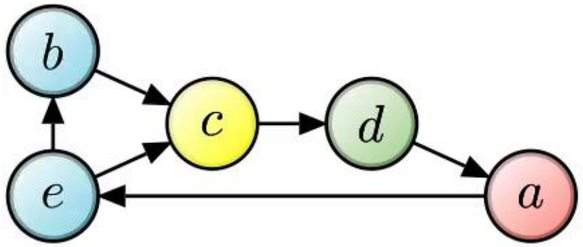
Output:

```
m(0,p,d).
m(1,o,c).
```

Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



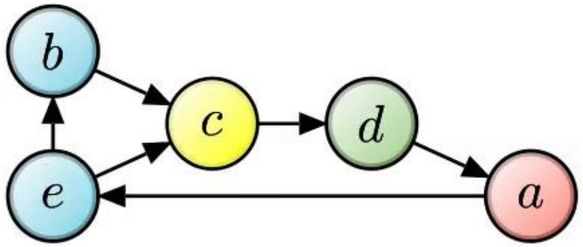
Output:

```
m(0,p,d).
m(1,o,c).
```

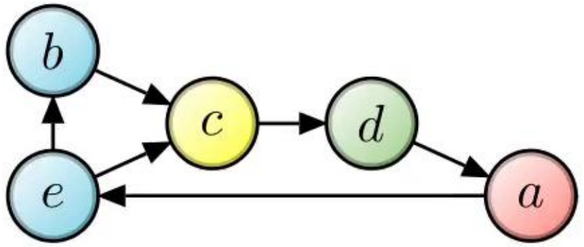
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).

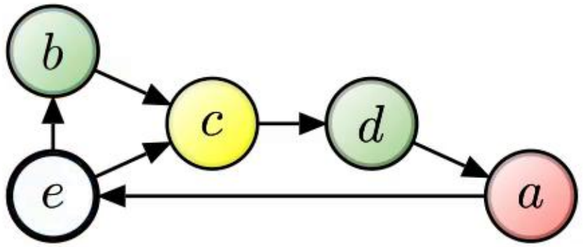


Output:
m(0,p,d).
m(1,o,c).

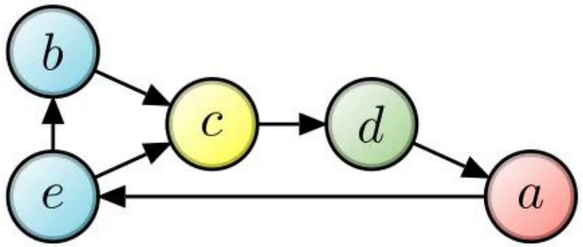
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).

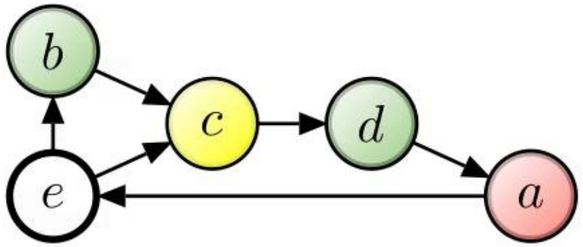


Output:
m(0,p,d).
m(1,o,c).

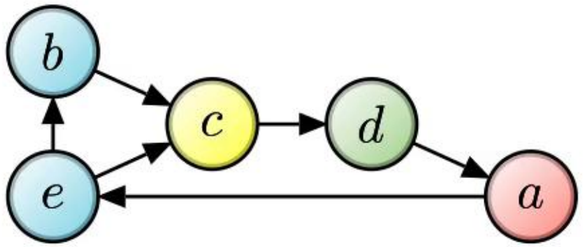
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).

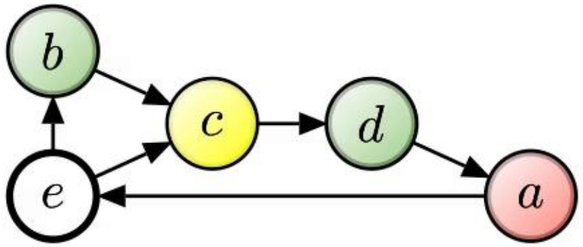


Output:
m(0,p,d).
m(1,o,c).

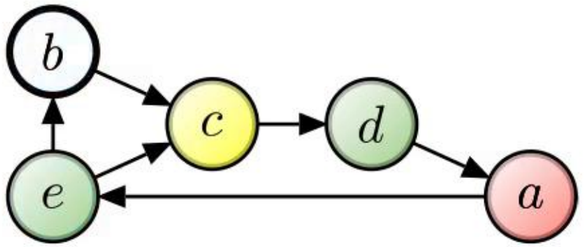
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).

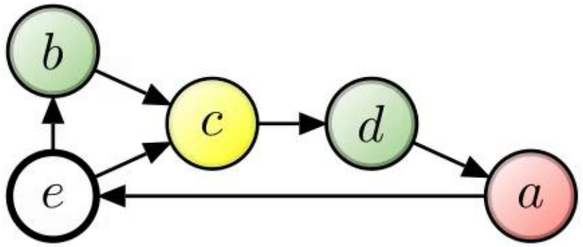


Output:
m(0,p,d).
m(1,o,c).

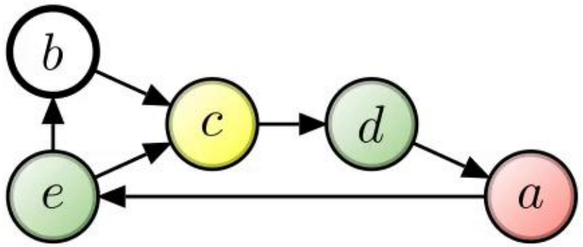
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).

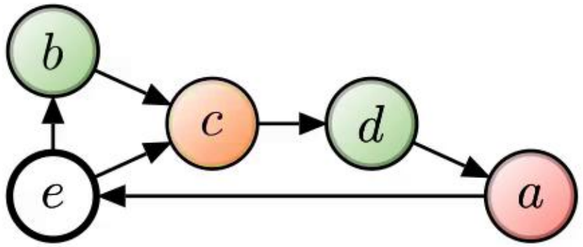


Output:
m(0,p,d).
m(1,o,c).
m(2,p,e).

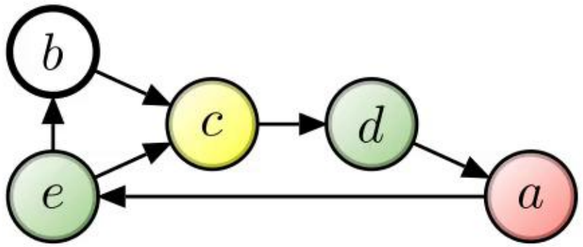
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).

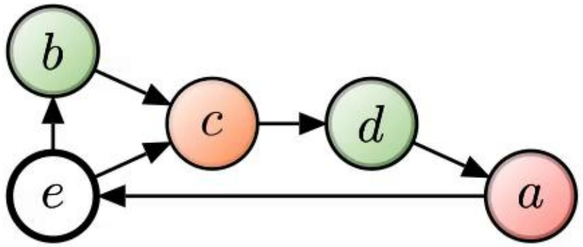


Output:
m(0,p,d).
m(1,o,c).
m(2,p,e).

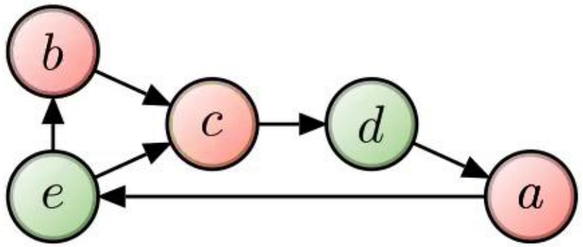
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).

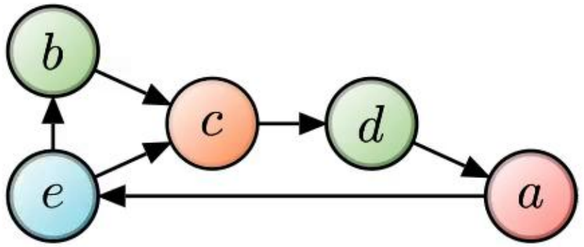


Output:
m(0,p,d).
m(1,o,c).
m(2,p,e).

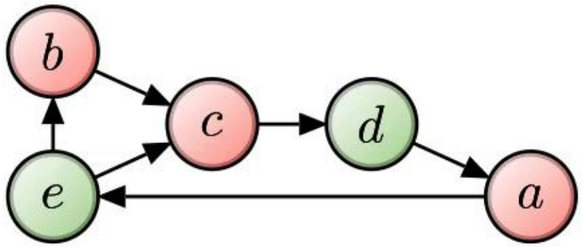
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).

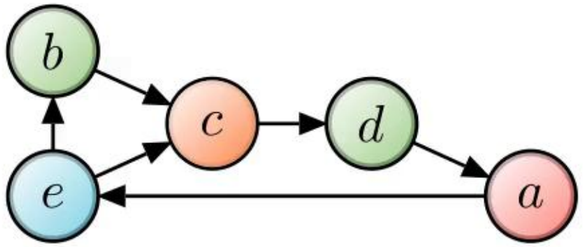


Output:
m(0,p,d).
m(1,o,c).
m(2,p,e).

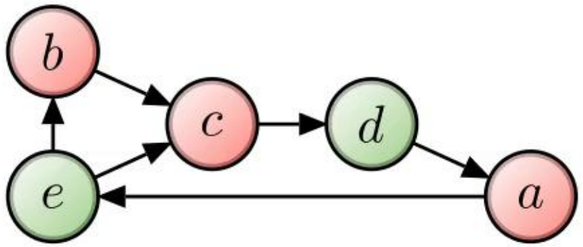
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).

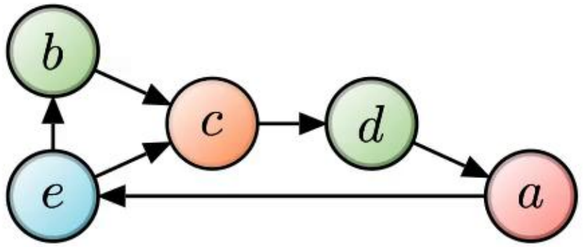


Output:
m(0,p,d).
m(1,o,c).
m(2,p,e).
end(2,p).

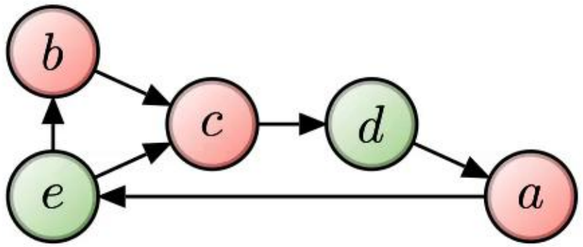
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).

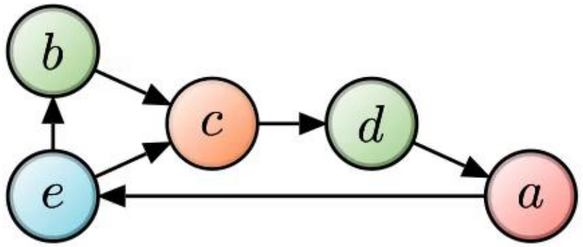


Output:
m(0,p,d).
m(1,o,c).
m(2,p,e).
end(2,p).

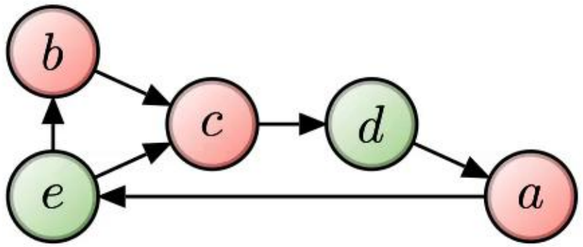
Multi-shot ASP for Dung's AFs

```
#program base.
% initialize
m(0,p,G) :- g(G), not att(G,G), arg(G).

#program updateState(t).
defeat(t,C) :- m(_,p,P), att(P,C). % defeated
pm(t,p,P) :- m(_,o,O), att(P,O), not defeat(_,P),
  not m(_,p,P), not att(P,P),
  not att(P,D1) : m(_,p,D1). % possible p. move
pm(t,o,O) :- m(_,p,P), att(O,P), not defeat(_,O),
  not m(_,o,O). % possible o. move
end(t,p) :- g(G), m(_,p,G), not pm(t,o,_),
  defeat(_,O1) : m(_,o,O1). % p. won
end(t,o) :- not pm(t,p,_), m(_,o,O),
  not defeat(_, O). % opp. won

#program step(t).
m(t,o,A) :- pm(t-1,o,A).
{ m(t,p,A) : pm(t-1,p,A) } = 1 :- not pm(t-1,o,_).

#show m/3.
#show end/2.
```

```
arg(a;b;c;d;e).
att(a,e).
att(b,c).
att(c,d).
att(d,a).
att(e,b).
att(e,c).
g(d).
```



Output:
m(0,p,d).
m(1,o,c).
m(2,p,b).



Output:
m(0,p,d).
m(1,o,c).
m(2,p,e).
end(2,p).

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

- Language $\mathcal{L}$

  $\{a, b, c, d, e, f, g \ldots\}$

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

- Language $\mathcal{L}$

  $\{a, b, c, d, e, f, g \ldots\}$

```
a b c d e f g
```

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

- Language $\mathcal{L}$

  $\{a, b, c, d, e, f, g \ldots\}$

- Assumptions $\mathcal{A} \subseteq \mathcal{L}$

  $\{b, d, \ldots\}$

```
a b c d e f g
```

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

- Language $\mathcal{L}$

  $\{a, b, c, d, e, f, g \ldots\}$

```
a b c d e f g
```

- Assumptions $\mathcal{A} \subseteq \mathcal{L}$

  $\{b, d, \ldots\}$

```
assumption(b).
assumption(d).
```

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

- Language $\mathcal{L}$

  $\{a, b, c, d, e, f, g \ldots\}$

```
a b c d e f g
```

- Assumptions $\mathcal{A} \subseteq \mathcal{L}$

  $\{b, d, \ldots\}$

```
assumption(b).
assumption(d).
```

- Contrary function $^{-} : \mathcal{A} \mapsto \mathcal{L}$

  $\bar{b} = c, \bar{d} = a, \ldots$

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

- Language $\mathcal{L}$

  $\{a, b, c, d, e, f, g \ldots\}$

```
a b c d e f g
```

- Assumptions $\mathcal{A} \subseteq \mathcal{L}$

  $\{b, d, \ldots\}$

```
assumption(b).
assumption(d).
```

- Contrary function $^{-} : \mathcal{A} \mapsto \mathcal{L}$

  $\bar{b} = c, \bar{d} = a, \ldots$

```
contrary(b,c).
contrary(d,a).
```

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

- Language $\mathcal{L}$

  $\{a, b, c, d, e, f, g \ldots\}$

```
a b c d e f g
```

- Assumptions $\mathcal{A} \subseteq \mathcal{L}$

  $\{b, d, \ldots\}$

```
assumption(b).
assumption(d).
```

- Contrary function $^{-} : \mathcal{A} \mapsto \mathcal{L}$

  $\bar{b} = c, \bar{d} = a, \ldots$

```
contrary(b,c).
contrary(d,a).
```

- Rules $h \leftarrow B$ with $h \in \mathcal{L} \setminus \mathcal{A}$, $B \subseteq \mathcal{L}$

  $\{d \leftarrow c, e, \ldots\}$

Extension to ABA Disputes: Framework

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$

- Language $\mathcal{L}$

  $\{a, b, c, d, e, f, g \ldots\}$

```
a b c d e f g
```

- Assumptions $\mathcal{A} \subseteq \mathcal{L}$

  $\{b, d, \ldots\}$

```
assumption(b).
assumption(d).
```

- Contrary function $^{-} : \mathcal{A} \mapsto \mathcal{L}$

  $\bar{b} = c, \bar{d} = a, \ldots$

```
contrary(b,c).
contrary(d,a).
```

- Rules $h \leftarrow B$ with $h \in \mathcal{L} \setminus \mathcal{A}$, $B \subseteq \mathcal{L}$

  $\{d \leftarrow c, e, \ldots\}$

```
head(1,c).   %
body(1,d).
body(1,e).
```

Extension to ABA Disputes: Arguments & Attacks

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Extension to ABA Disputes: Arguments & Attacks

Technische Universität Dresden

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

Extension to ABA Disputes: Arguments & Attacks

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

Extension to ABA Disputes: Arguments & Attacks

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

Extension to ABA Disputes: Arguments & Attacks

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

$a \qquad b \qquad c$

Extension to ABA Disputes: Arguments & Attacks

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

$a \qquad b \qquad c$

Extension to ABA Disputes: Arguments & Attacks

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

$a \qquad b \qquad c$

Extension to ABA Disputes: Arguments & Attacks

$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

$a \qquad b \qquad c$

Extension to ABA Disputes: Arguments & Attacks

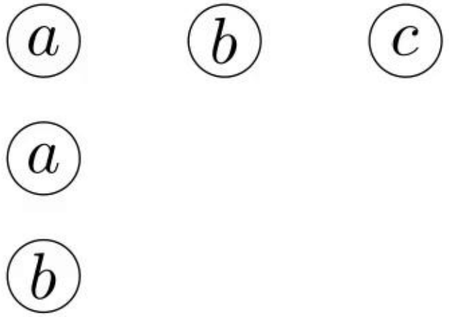
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$



Extension to ABA Disputes: Arguments & Attacks

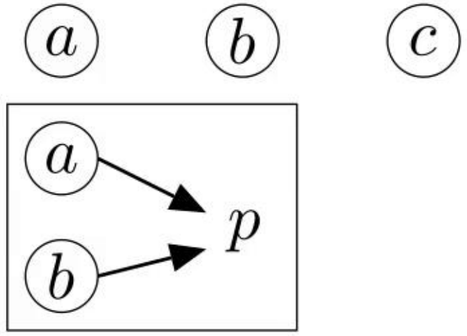
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$



Extension to ABA Disputes: Arguments & Attacks

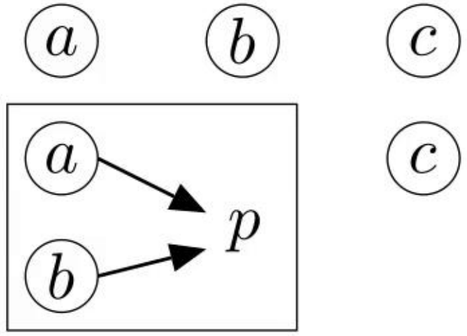
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$



Extension to ABA Disputes: Arguments & Attacks

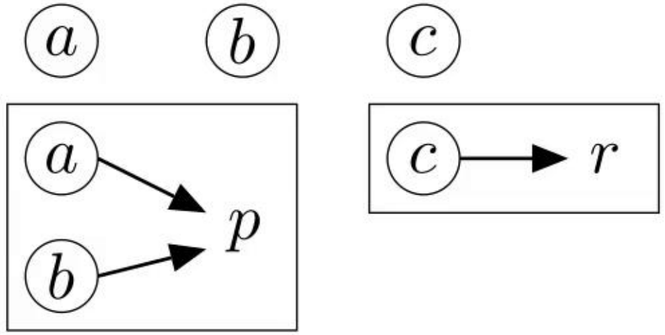
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$



Extension to ABA Disputes: Arguments & Attacks

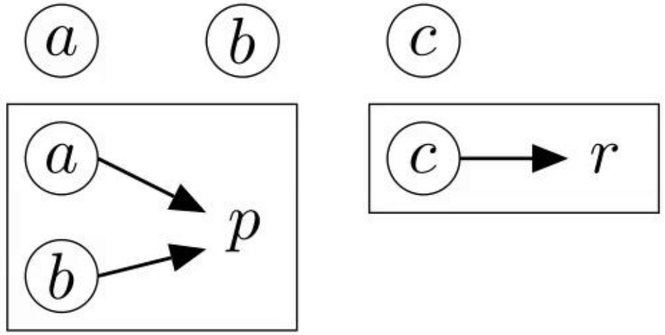
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$



Extension to ABA Disputes: Arguments & Attacks

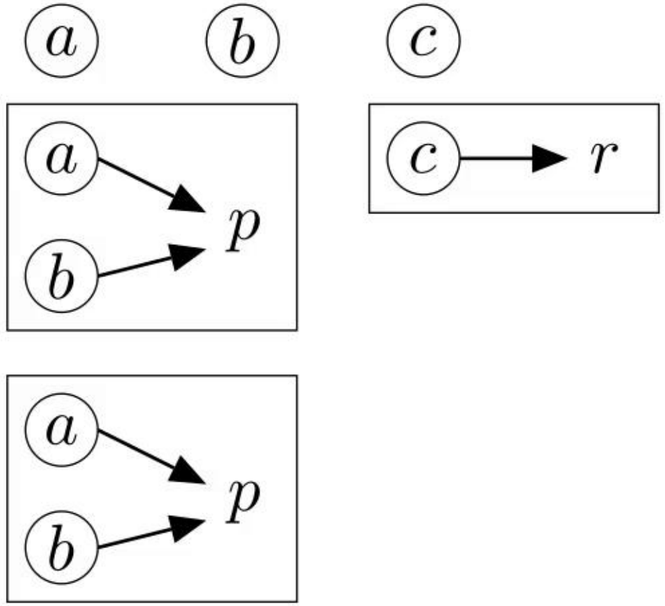
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$



Extension to ABA Disputes: Arguments & Attacks

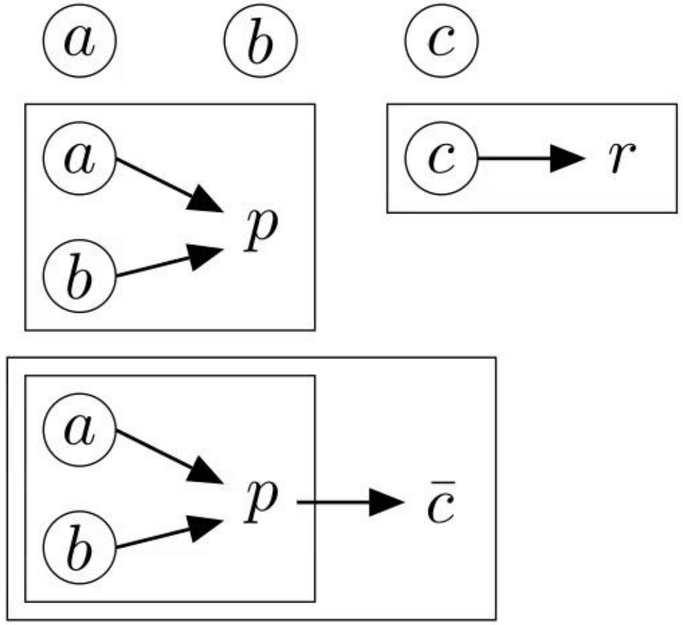
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$



Extension to ABA Disputes: Arguments & Attacks

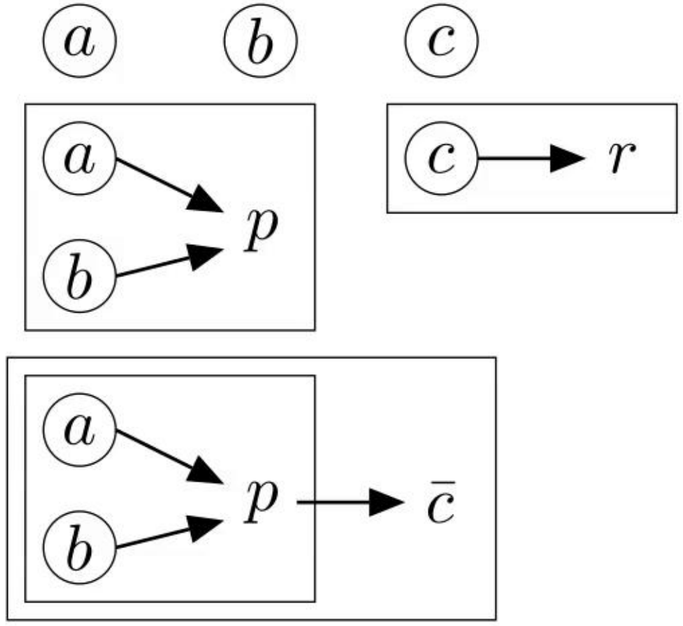
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$
  then $conc(A) = h$, $asm(A) = asm(A_1) \cup \ldots \cup asm(A_n)$

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$



Extension to ABA Disputes: Arguments & Attacks

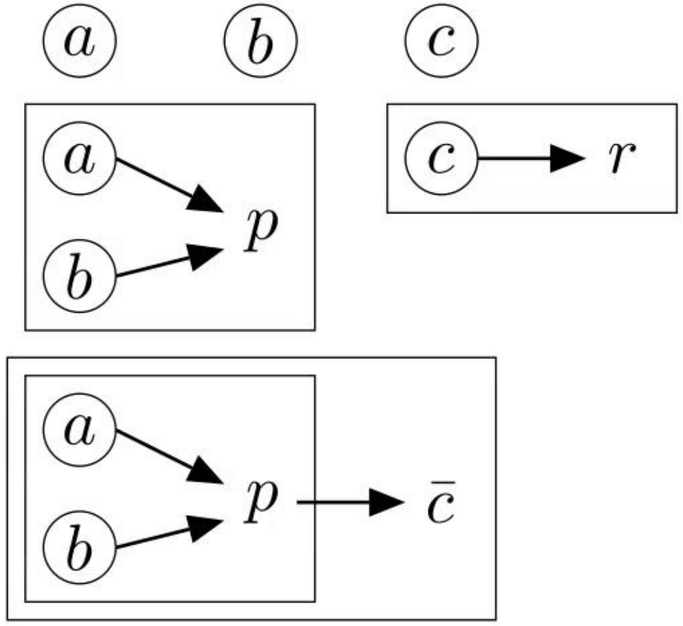
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$
  then $conc(A) = h$, $asm(A) = asm(A_1) \cup \ldots \cup asm(A_n)$



Define $Att \subseteq Args \times Args$:

Extension to ABA Disputes: Arguments & Attacks

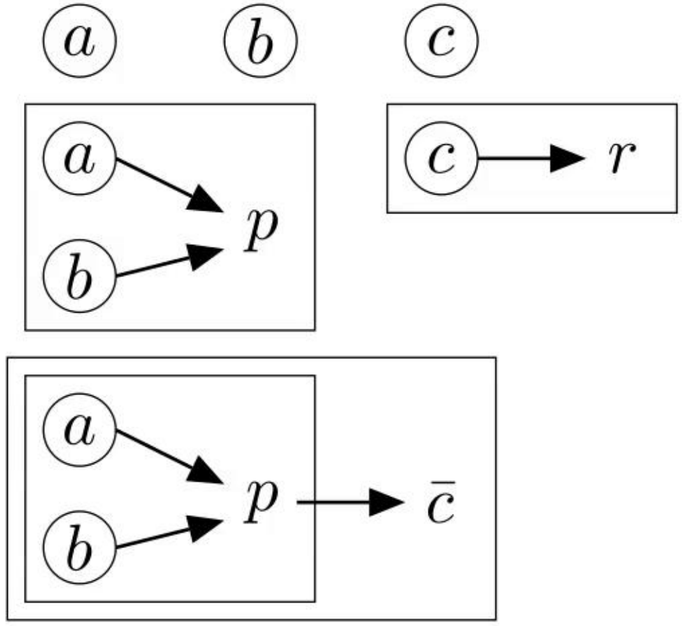
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$
  then $conc(A) = h$, $asm(A) = asm(A_1) \cup \ldots \cup asm(A_n)$



Define $Att \subseteq Args \times Args$:

- $(A_1, A_2) \in Att$

Extension to ABA Disputes: Arguments & Attacks

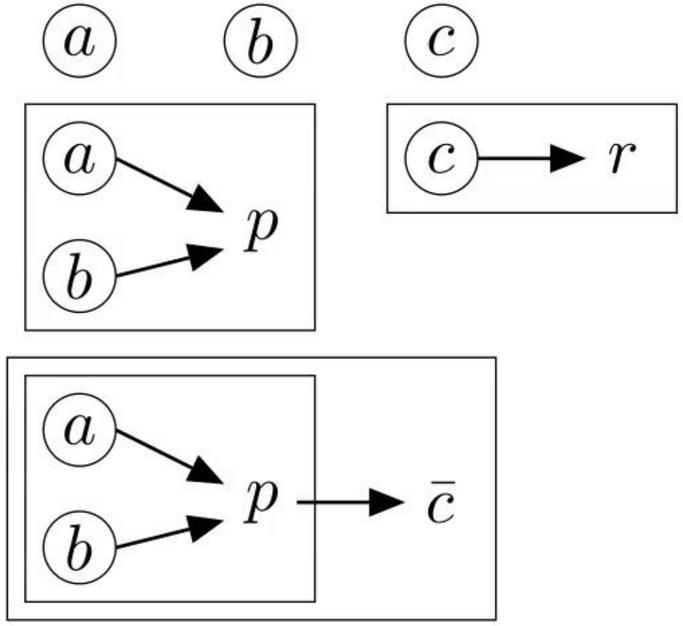
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$
  then $conc(A) = h$, $asm(A) = asm(A_1) \cup \ldots \cup asm(A_n)$



Define $Att \subseteq Args \times Args$:

- $(A_1, A_2) \in Att$
  if $A_1, A_2 \in Args$, $a \in asm(A_2)$, $conc(A_1) = \bar{a}$

Extension to ABA Disputes: Arguments & Attacks

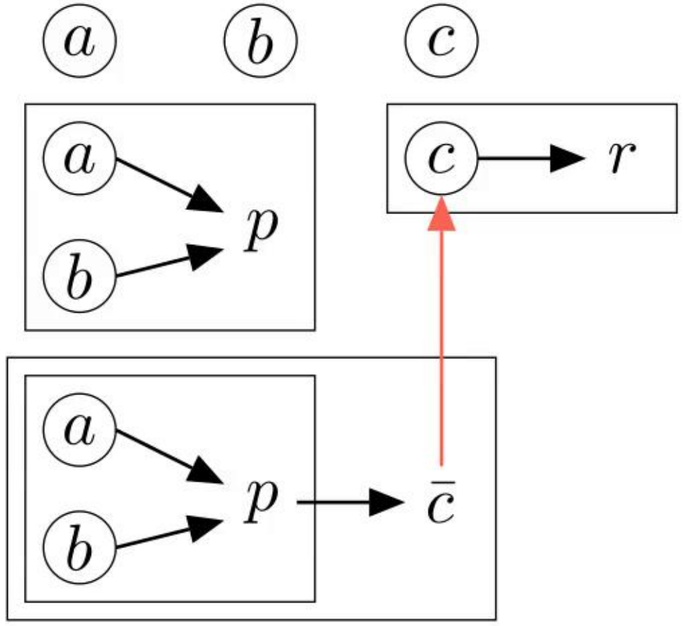
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

Define $Args$:

- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$
  then $conc(A) = h$, $asm(A) = asm(A_1) \cup \ldots \cup asm(A_n)$



Define $Att \subseteq Args \times Args$:

- $(A_1, A_2) \in Att$
  if $A_1, A_2 \in Args$, $a \in asm(A_2)$, $conc(A_1) = \bar{a}$

Extension to ABA Disputes: Arguments & Attacks

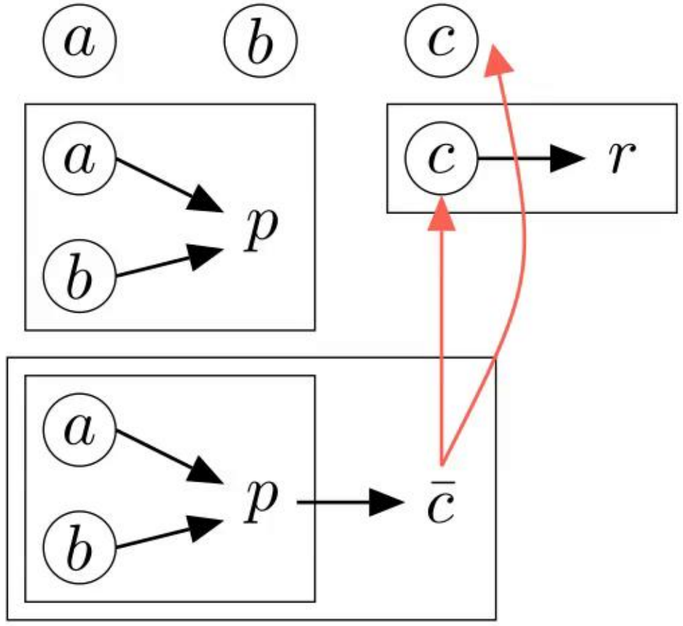
$\mathcal{F} = (\mathcal{L}, \mathcal{A}, ^{-}, \mathcal{R})$.

$\mathcal{A} = \{a, b, c\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{p \leftarrow a, b; \quad \bar{c} \leftarrow p; \quad r \leftarrow c\}$

Define $Args$:

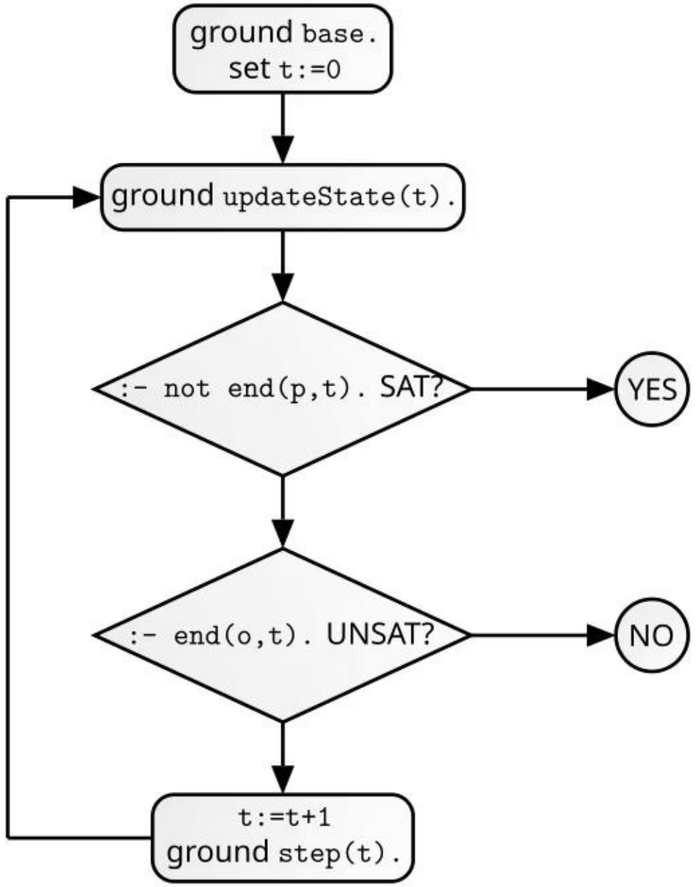
- $a \in Args$ if $a \in \mathcal{A}$
  then $conc(a) = a$, $asm(a) = a$
- $A = h \leftarrow A_1, \ldots, A_n \in Args$
  if $\{A_1, \ldots, A_n\} \subseteq Args$ and $h \leftarrow conc(A_1), \ldots, conc(A_n)$
  then $conc(A) = h$, $asm(A) = asm(A_1) \cup \ldots \cup asm(A_n)$



Define $Att \subseteq Args \times Args$:

- $(A_1, A_2) \in Att$
  if $A_1, A_2 \in Args$, $a \in asm(A_2)$, $conc(A_1) = \bar{a}$

Extension to ABA Disputes: Code



Extension to ABA Disputes: Code

```
#program base.
rMT(pb1;pb2;pf1;ob1;ob2;of1). % rule move types
branchMT(pb1;pb2;pf2). % branching (br.) move types
plr(p;o). % players; proponent (p.) and opponent (o.)
rS(R,S) :- head(R,S). % rule's statements from rule heads
rS(R,S) :- body(R,S). % rule's statements from rule bodies
stS(0,S,P) :- g(S), not contrary(S,S), plr(P). % goal - initial statement
remBloR(0,R,H,p) :- head(R,H), rS(R,S1), rS(R,S2), contrary(S1,S2). % incons. rules
```

Extension to ABA Disputes: Code

```
#program base.
rMT(pb1;pb2;pf1;ob1;ob2;of1). % rule move types
branchMT(pb1;pb2;pf2). % branching (br.) move types
plr(p;o). % players; proponent (p.) and opponent (o.)
rS(R,S) :- head(R,S). % rule's statements from rule heads
rS(R,S) :- body(R,S). % rule's statements from rule bodies
stS(0,S,P) :- g(S), not contrary(S,S), plr(P). % goal - initial statement
remBloR(0,R,H,p) :- head(R,H), rS(R,S1), rS(R,S2), contrary(S1,S2). % incons. rules
```

Extension to ABA Disputes: Code

```
#program base.
rMT(pb1;pb2;pf1;ob1;ob2;of1). % rule move types
branchMT(pb1;pb2;pf2). % branching (br.) move types
plr(p;o). % players; proponent (p.) and opponent (o.)
rS(R,S) :- head(R,S). % rule's statements from rule heads
rS(R,S) :- body(R,S). % rule's statements from rule bodies
stS(0,S,P) :- g(S), not contrary(S,S), plr(P). % goal - initial statement
remBloR(0,R,H,p) :- head(R,H), rS(R,S1), rS(R,S2), contrary(S1,S2). % incons. rules
```

Extension to ABA Disputes: Code

```
#program updateState(t).
def(t,D) :- stS(_,D,p), assumption(D). % defence
cul(t,C) :- stS(_,S,p), contrary(C,S). % culprit
defCtr(t,DC) :- def(_,D), contrary(D,DC). % defence contrary
culCtr(t,CC) :- cul(_,C), contrary(C,CC). % culprit contrary

remR(t,R,H,P) :- not stR(_,R,H,P), head(R,H), plr(P). % remaining player's rule
remBloR(t,R,H,P) :- remR(t,R,H,P), body(R,B), cul(_,B), plr(P). % block. rem. P. rules
remBloR(t,R,H,p) :- remR(t,R,H,p), rS(R,S), defCtr(_,S). % blocked remaining p.'s rule

unexpS(t,H,p) :- stS(_,H,p), not stR(_,_,H,p). % unexpanded statement
stExpS(t,H,o) :- stS(_,H,o), remBloR(_,R,H,o) : remR(t,R,H,o). % fully expanded
stBloS(t,S,o) :- stS(_,S,o), cul(_,S). % state blocked statement
stBloS(t,S,o) :- stExpS(t,S,o), not assumption(S), stBloR(t,R,S,o) : stR(_,R,S,o).
stBloR(t,R,H,o) :- stR(_,R,H,o), body(R,B), stBloS(t,B,o). % state blocked rule
```

Extension to ABA Disputes: Code

```
comS(t,S,p) :- def(_,D).  % complete statement
comS(t,H,p) :- stR(_,R,H,p), comS(t,S,p) : body(R, S).
unbloComS(t,S,o) :- stS(_,S,o), assumption(S), not cul(_,S). % unblocked complete
unbloComS(t,H,o) :- stS(_,H,o), not stBloS(t,H,o), unbloComR(t,_,H,o).
unbloComR(t,R,H,o) :-  stR(_,R,H,o), not stBloR(t,R,H,o), unbloComS(t,B,o) : body(R, B).
unbloSupSS(t,S,o) :- stS(_,S,o), contrary(D,S), def(t,D), not stBloS(t,S,o).
unbloSupSS(t,S,o) :- stS(_,S,o), not stBloS(t,S,o), unbloSupSR(t,R,_,o), body(R, S).
unbloSupSR(t,R,H,o) :- stR(_,R,H,o), not stBloR(t,R,H,o), unbloSupSS(t,H,o).
culCan(t,C) :- assumption(C), unbloSupSS(t,C,o). % culprit candidate

pm(t,p,pb1,R) :- remR(t,R,H,p), not remBloR(_,R,H,p), unexpS(t,H,p). % possible moves
pm(t,p,pb2,R) :- remR(t,R,H,p), not remBloR(_,R,H,p), culCan(t,C), contrary(C, H),
  not stS(_,H,p), not contrary(D, H) : def(_,D).
pm(t,p,pf1,R) :- remR(t,R,H,p), not remBloR(_,R,H,p), comS(t,B,p) : body(R, B).
pm(t,p,pf2,A) :- culCan(t,C), contrary(C,A), assumption(A), not stS(_,A,p),
  not contrary(A,A), not cul(_,A),  not contrary(D,A) : def(_,D).
pm(t,o,ob1,R) :- remR(t,R,H,o), not remBloR(_,R,H,o), unbloSupSS(t,H,o).
pm(t,o,ob2,R) :- remR(t,R,H,o), not remBloR(_,R,H,o), contrary(D, H), def(_,D).
pm(t,o,of1,R) :- remR(t,R,H,o), not remBloR(_,R,H,o), unbloComS(t,B,o) : body(R, B).
pm(t,o,of2,A) :- contrary(D,A), def(_,D), assumption(A), not stS(_,A,o).
```

Extension to ABA Disputes: Code

```
comS(t,S,p) :- def(_,D).  % complete statement
comS(t,H,p) :- stR(_,R,H,p), comS(t,S,p) : body(R, S).
unbloComS(t,S,o) :- stS(_,S,o), assumption(S), not cul(_,S). % unblocked complete
unbloComS(t,H,o) :- stS(_,H,o), not stBloS(t,H,o), unbloComR(t,_,H,o).
unbloComR(t,R,H,o) :-  stR(_,R,H,o), not stBloR(t,R,H,o), unbloComS(t,B,o) : body(R, B).
unbloSupSS(t,S,o) :- stS(_,S,o), contrary(D,S), def(t,D), not stBloS(t,S,o).
unbloSupSS(t,S,o) :- stS(_,S,o), not stBloS(t,S,o), unbloSupSR(t,R,_,o), body(R, S).
unbloSupSR(t,R,H,o) :- stR(_,R,H,o), not stBloR(t,R,H,o), unbloSupSS(t,H,o).
culCan(t,C) :- assumption(C), unbloSupSS(t,C,o). % culprit candidate

pm(t,p,pb1,R) :- remR(t,R,H,p), not remBloR(_,R,H,p), unexpS(t,H,p). % possible moves
pm(t,p,pb2,R) :- remR(t,R,H,p), not remBloR(_,R,H,p), culCan(t,C), contrary(C, H),
  not stS(_,H,p), not contrary(D, H) : def(_,D).
pm(t,p,pf1,R) :- remR(t,R,H,p), not remBloR(_,R,H,p), comS(t,B,p) : body(R, B).
pm(t,p,pf2,A) :- culCan(t,C), contrary(C,A), assumption(A), not stS(_,A,p),
  not contrary(A,A), not cul(_,A),  not contrary(D,A) : def(_,D).
pm(t,o,ob1,R) :- remR(t,R,H,o), not remBloR(_,R,H,o), unbloSupSS(t,H,o).
pm(t,o,ob2,R) :- remR(t,R,H,o), not remBloR(_,R,H,o), contrary(D, H), def(_,D).
pm(t,o,of1,R) :- remR(t,R,H,o), not remBloR(_,R,H,o), unbloComS(t,B,o) : body(R, B).
pm(t,o,of2,A) :- contrary(D,A), def(_,D), assumption(A), not stS(_,A,o).
```

Extension to ABA Disputes: Code

```
stS(t,S,P) :- m(_,p,T,R), rMT(T), rS(R,S), plr(P). % new state statement
stS(t,A,P) :- m(_,p,T,A), not rMT(T), plr(P).
stS(t,S,o) :- m(_,o,T,R), rMT(T), rS(R,S).
stS(t,A,o) :- m(_,o,T,A), not rMT(T).
stR(t,R,H,P) :- m(_,p,T,R), head(R,H), rMT(T), plr(P). % new state rule
stR(t,R,H,o) :- m(_,o,T,R), head(R,H), rMT(T).

term(t) :-  tt(ta), g(G), comS(_,G,p), comS(_,CC,p) : culCtr(_,CC);
  not unbloComS(t,DC,o) : defCtr(_,DC).
end(t,p) :- term(t), not pm(t,o,_,_). % p. won
end(t,o) :- not term(t), not pm(t,p,_,_). % o won
```

Extension to ABA Disputes: Code

```
stS(t,S,P) :- m(_,p,T,R), rMT(T), rS(R,S), plr(P). % new state statement
stS(t,A,P) :- m(_,p,T,A), not rMT(T), plr(P).
stS(t,S,o) :- m(_,o,T,R), rMT(T), rS(R,S).
stS(t,A,o) :- m(_,o,T,A), not rMT(T).
stR(t,R,H,P) :- m(_,p,T,R), head(R,H), rMT(T), plr(P). % new state rule
stR(t,R,H,o) :- m(_,o,T,R), head(R,H), rMT(T).

term(t) :-  tt(ta), g(G), comS(_,G,p), comS(_,CC,p) : culCtr(_,CC);
  not unbloComS(t,DC,o) : defCtr(_,DC).
end(t,p) :- term(t), not pm(t,o,_,_). % p. won
end(t,o) :- not term(t), not pm(t,p,_,_). % o won
```

Extension to ABA Disputes: Code

```
#program step(t).
m(t,P,T,X) :- pm(t-1,P,T,X), not branchMT(T), t > 0.
{ m(t,P,T,X) : pm(t-1,P,T,X) } = 1 :- t > 0, branchMT(T) : pm(t-1,_,T,_).
#show m/4.
```

Extension to ABA Disputes: Code

```
#program step(t).
m(t,P,T,X) :- pm(t-1,P,T,X), not branchMT(T), t > 0.
{ m(t,P,T,X) : pm(t-1,P,T,X) } = 1 :- t > 0, branchMT(T) : pm(t-1,_,T,_).
#show m/4.
```

Extension to ABA Disputes: Code

```
#program step(t).
m(t,P,T,X) :- pm(t-1,P,T,X), not branchMT(T), t > 0.
{ m(t,P,T,X) : pm(t-1,P,T,X) } = 1 :- t > 0, branchMT(T) : pm(t-1,_,T,_).
#show m/4.
```

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;\ \ \bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$$

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \ \}$

Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$

s

Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$

s

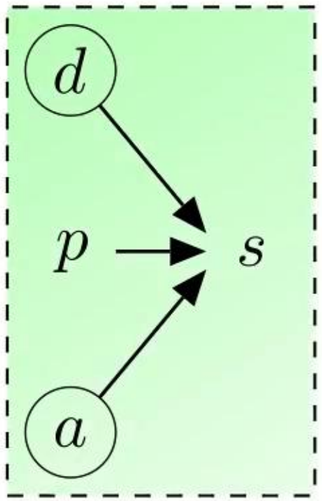
Output:

$\boxed{\texttt{m(1,p,pb1,}s \leftarrow d, p, a\texttt{).}}$
$\texttt{m(2,o,ob2,}\bar{a} \leftarrow b, t\texttt{).}$
$\texttt{m(2,o,ob2,}\bar{d} \leftarrow e\texttt{).}$
$\texttt{m(3,o,ob1,}t \leftarrow c\texttt{).}$
$\texttt{m(4,p,pf2,}\bar{e}\texttt{).}$
$\texttt{m(5,p,pb1,}p \leftarrow \bar{c}\texttt{).}$
$\texttt{m(6,p,pb1,}\bar{c} \leftarrow f\texttt{).}$

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



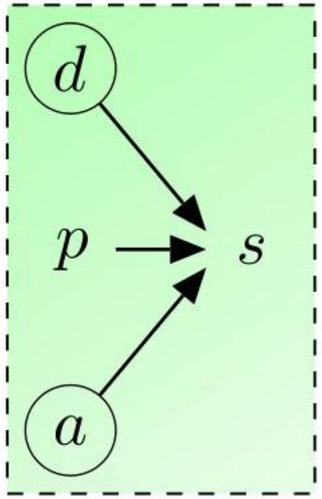
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



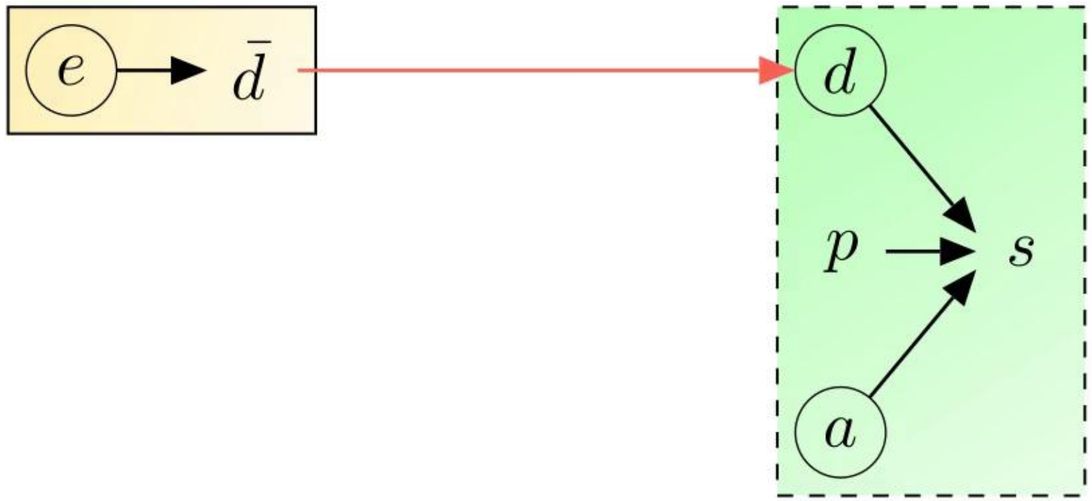
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



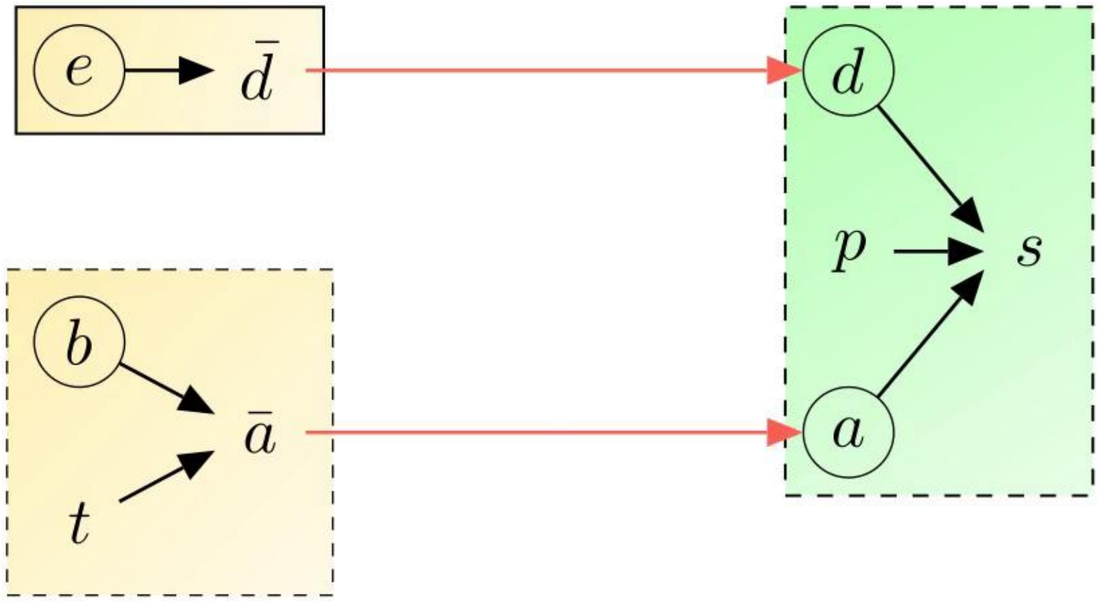
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



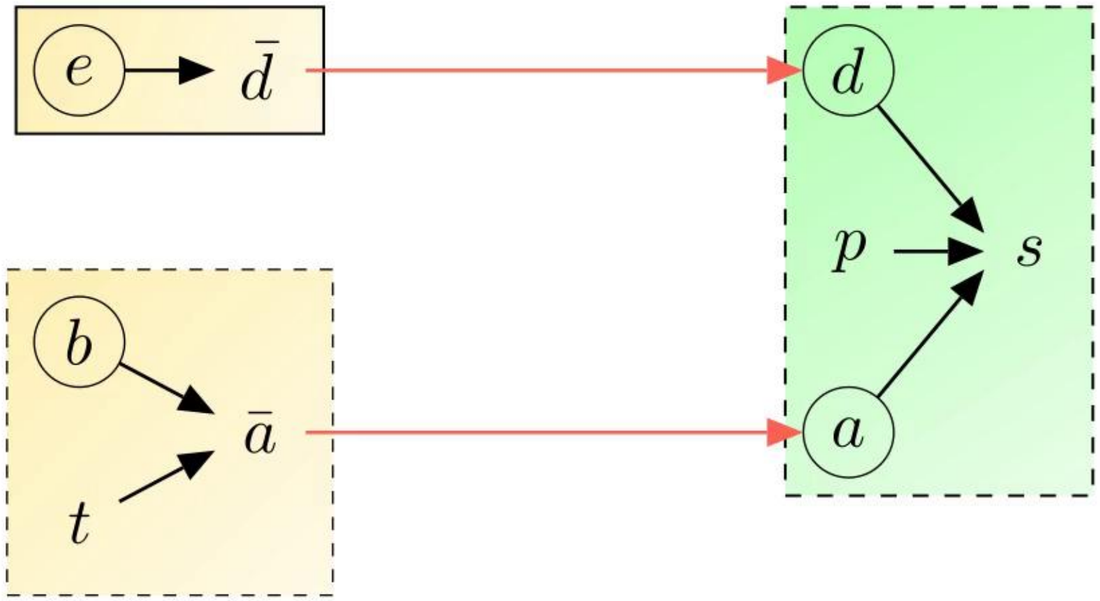
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



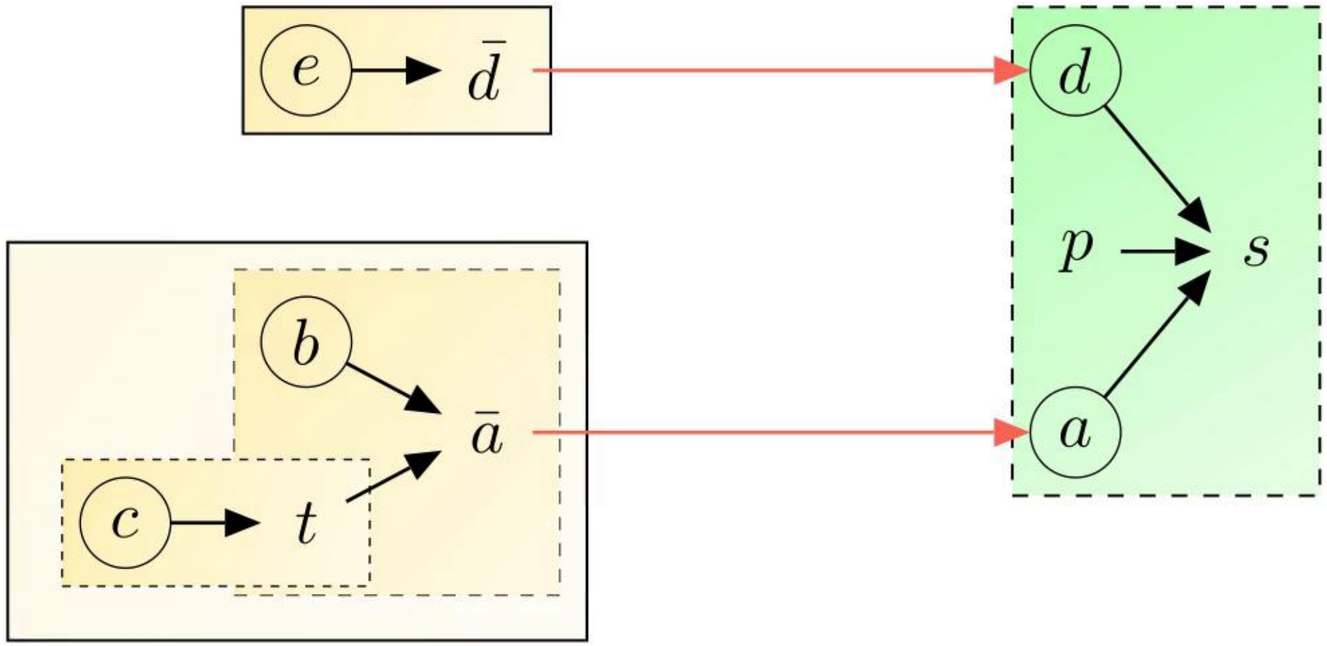
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



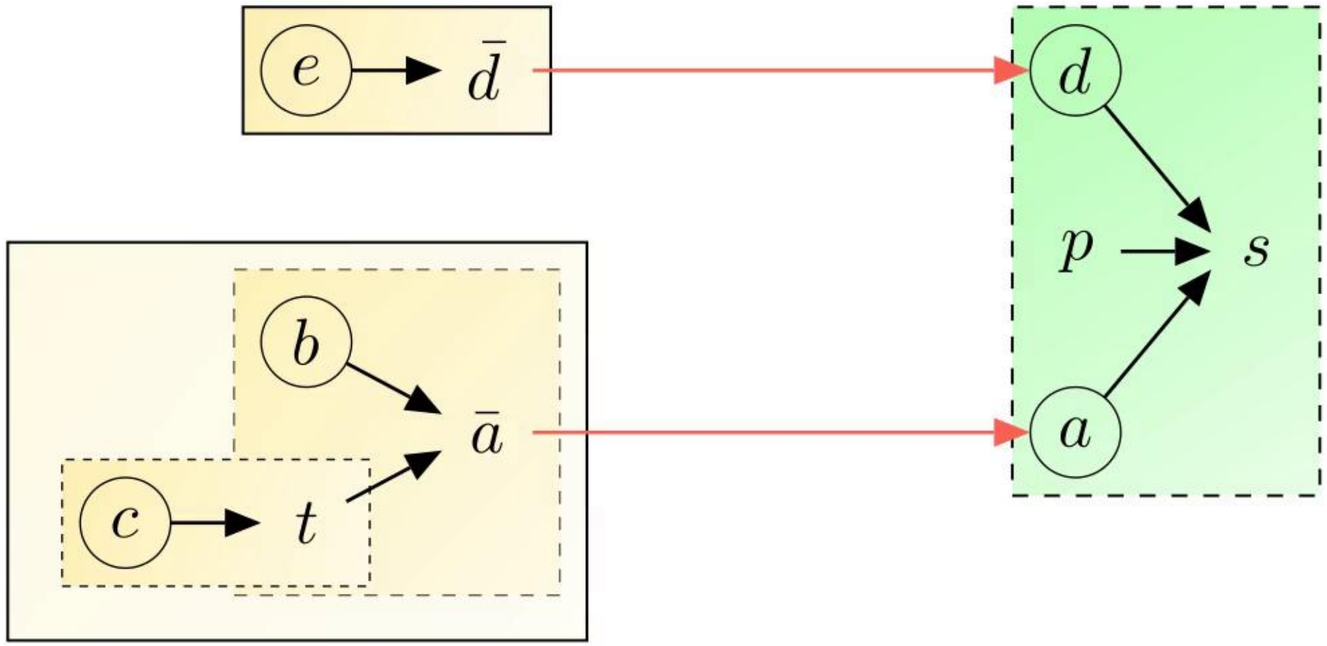
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



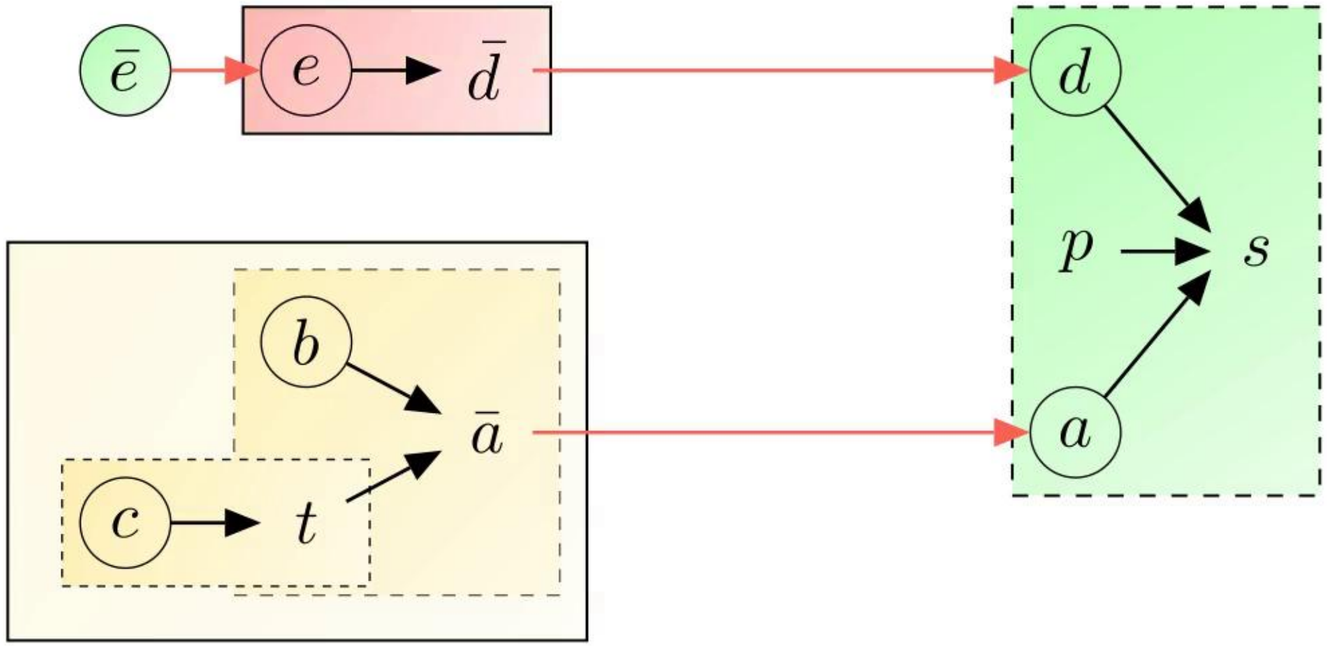
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



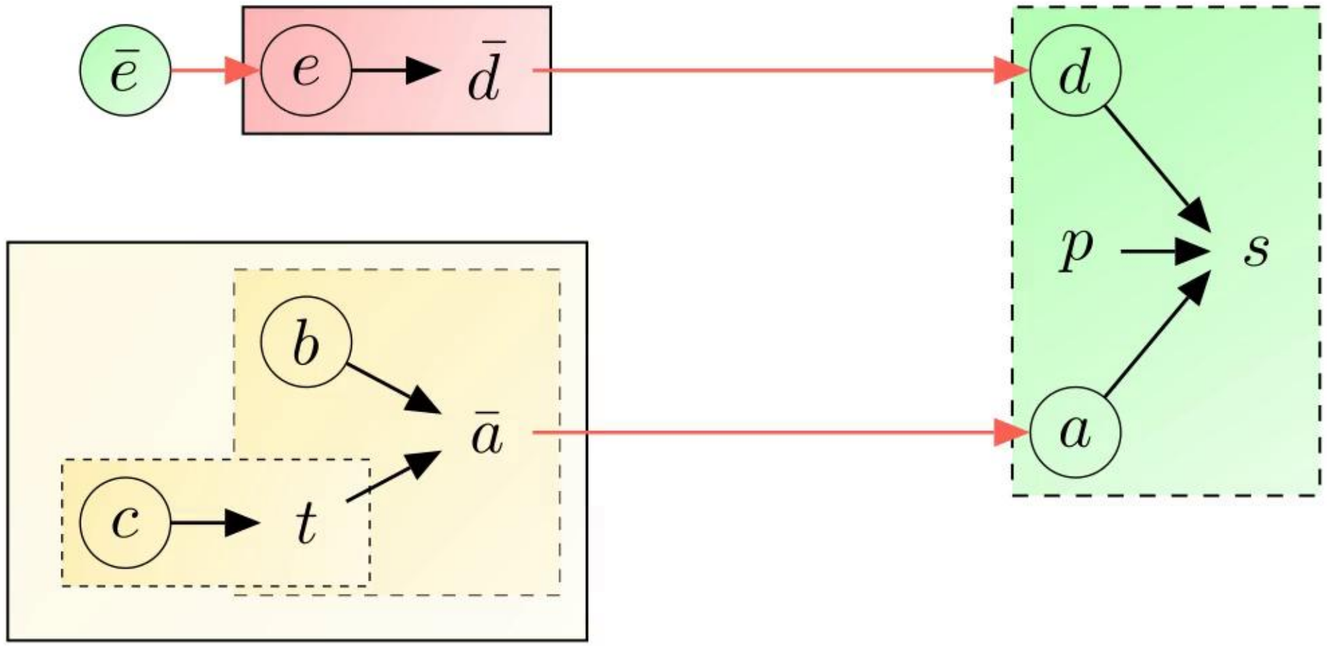
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



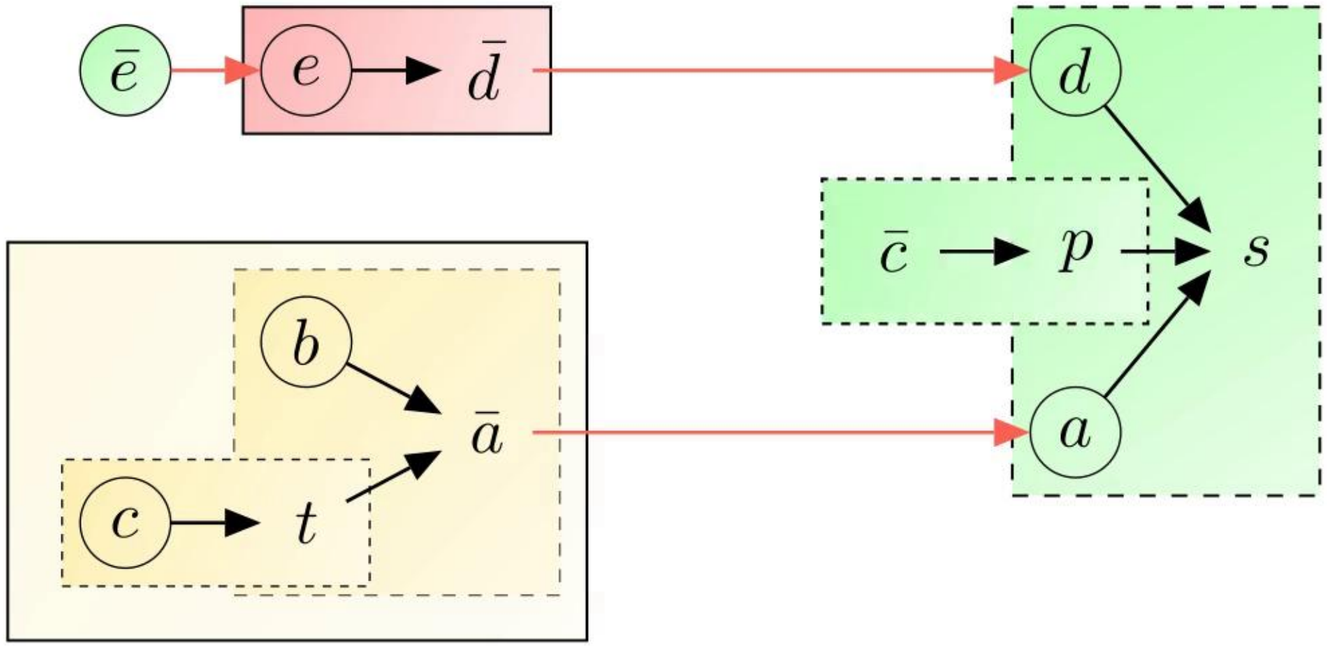
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



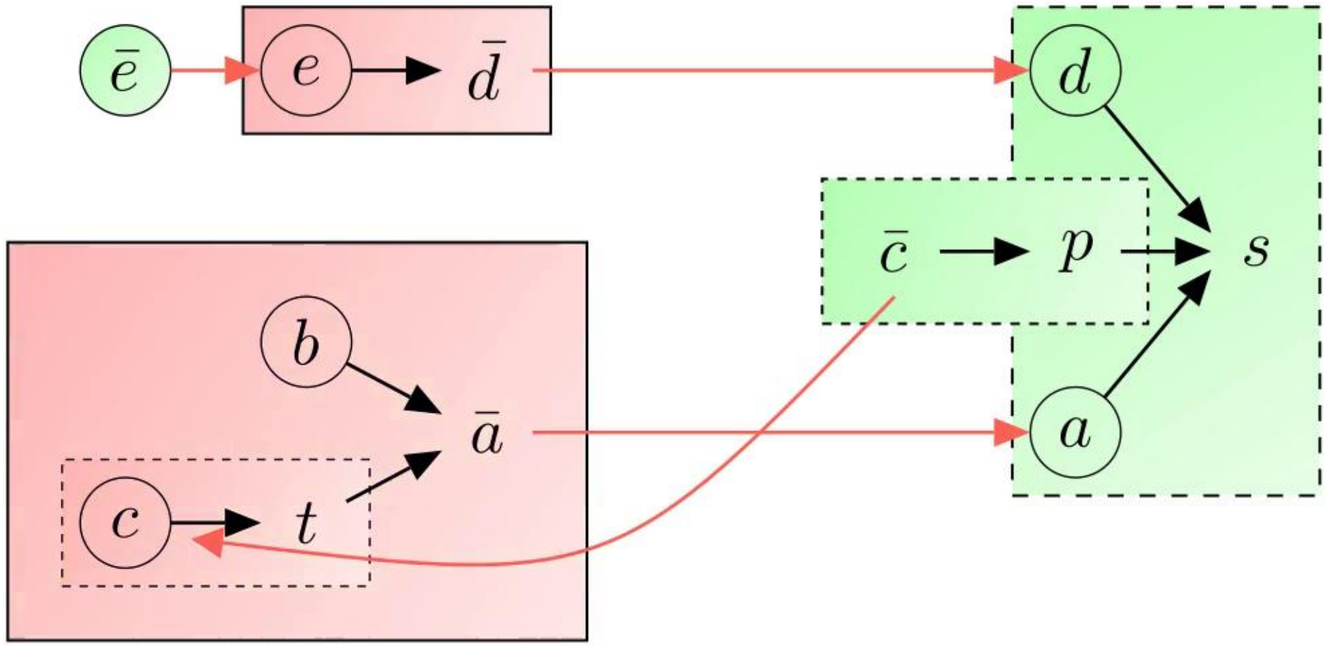
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



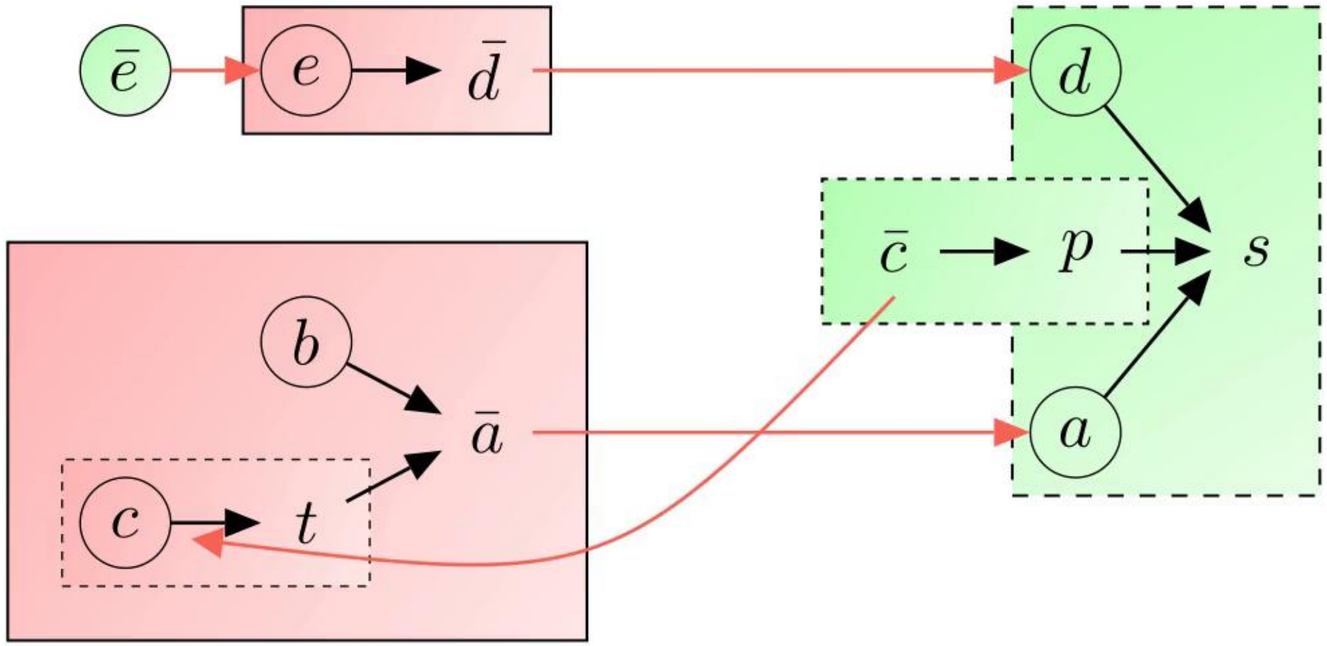
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



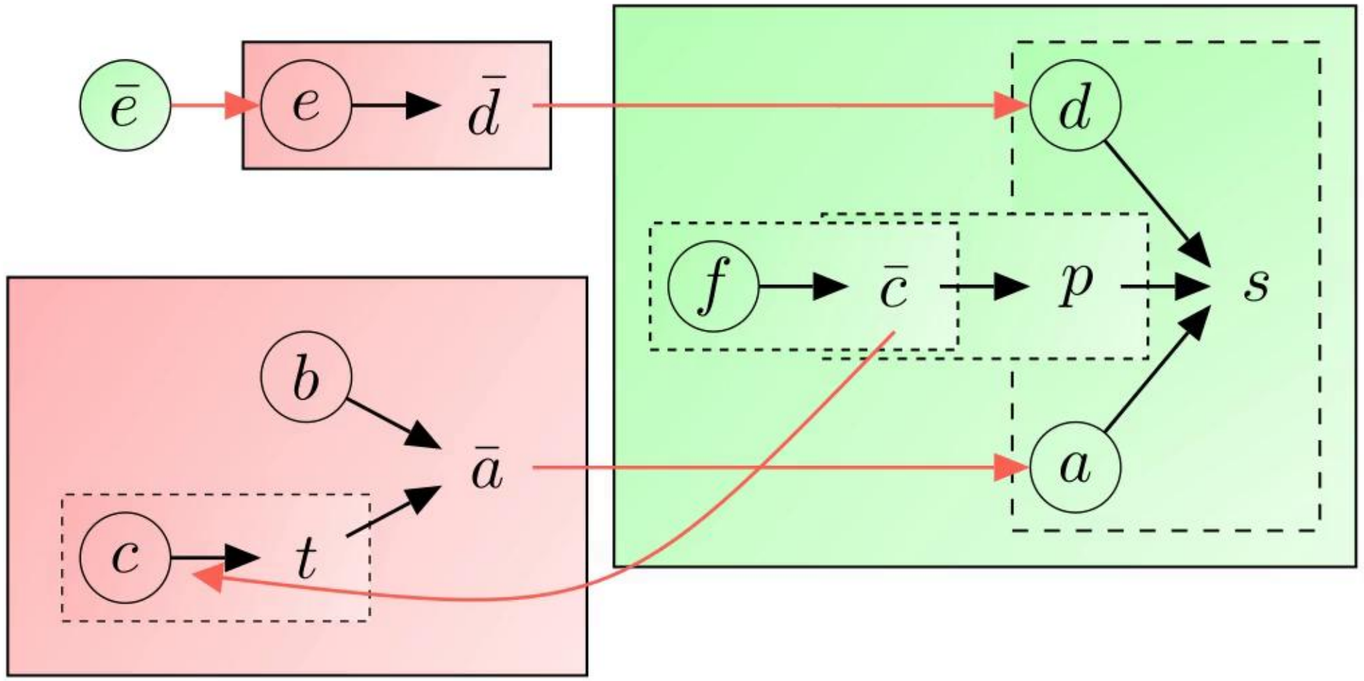
Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Extension to ABA Disputes: Dispute

$\mathcal{A} = \{a, b, c, d, e, \bar{e}, f\}$, $^{-}(x) = \bar{x}$ for $x \in \mathcal{A}$

$\mathcal{R} = \{s \leftarrow d, p, a;\ \ p \leftarrow \bar{c};\ \ \bar{c} \leftarrow f;$
$\bar{a} \leftarrow b, t;\ \ \bar{d} \leftarrow e;\ \ t \leftarrow c\ \}$



Output:

m(1,p,pb1,$s \leftarrow d, p, a$).
m(2,o,ob2,$\bar{a} \leftarrow b, t$).
m(2,o,ob2,$\bar{d} \leftarrow e$).
m(3,o,ob1,$t \leftarrow c$).
m(4,p,pf2,$\bar{e}$).
m(5,p,pb1,$p \leftarrow \bar{c}$).
m(6,p,pb1,$\bar{c} \leftarrow f$).

Evaluation Setup

Solvers:

- **MS-DIS** – lightweight Multi-Shot ASP **dispute-based** solver

Evaluation Setup

Technische Universität Dresden

Solvers:

- **MS-DIS** – lightweight Multi-Shot ASP **dispute-based** solver
- **flexABle** [Diller, Gaggl, Gorczyca 2022] – state-of-the-art ABA **dispute-based** solver, written in Scala

Evaluation Setup

Technische Universität Dresden

Solvers:

- **MS-DIS** – lightweight Multi-Shot ASP **dispute-based** solver
- **flexABle** [Diller, Gaggl, Gorczyca 2022] – state-of-the-art ABA **dispute-based** solver, written in Scala
- **aspforaba** [Lehtonen, Wallner, Järvisalo 2023] – **reduction-based** one-shot ASP solver – upper bound on performance

Evaluation Setup

Technische Universität Dresden

Solvers:

- **MS-DIS** – lightweight Multi-Shot ASP **dispute-based** solver
- **flexABle** [Diller, Gaggl, Gorczyca 2022] – state-of-the-art ABA **dispute-based** solver, written in Scala
- **aspforaba** [Lehtonen, Wallner, Järvisalo 2023] – **reduction-based** one-shot ASP solver – upper bound on performance

Setup:

Evaluation Setup

Technische Universität Dresden

Solvers:

- **MS-DIS** – lightweight Multi-Shot ASP **dispute-based** solver
- **flexABle** [Diller, Gaggl, Gorczyca 2022] – state-of-the-art ABA **dispute-based** solver, written in Scala
- **aspforaba** [Lehtonen, Wallner, Järvisalo 2023] – **reduction-based** one-shot ASP solver – upper bound on performance

Setup:

- ICCMA'23 ABA track instances (containing 25-5000 atoms)

Evaluation Setup

Technische Universität Dresden

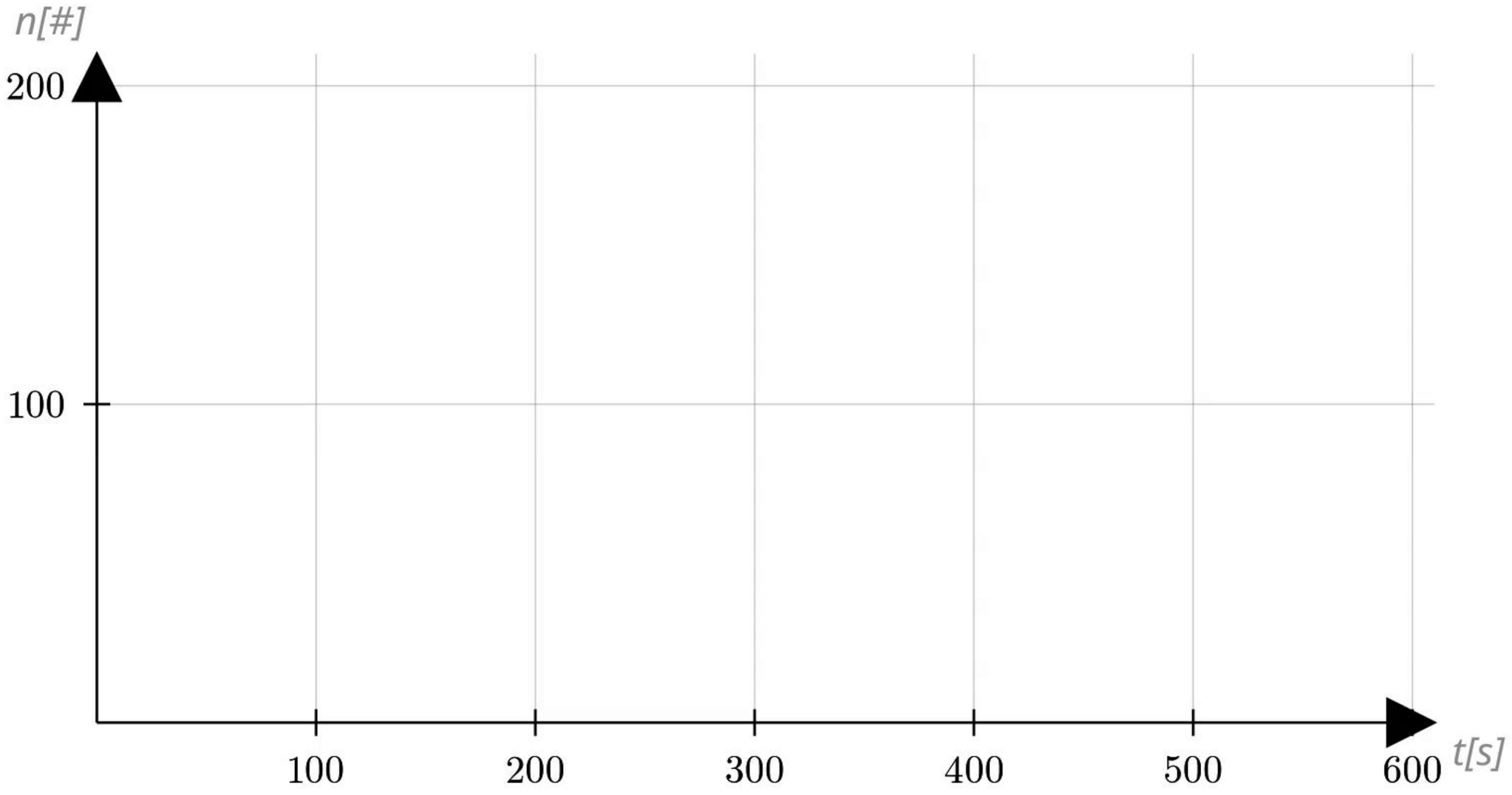
Solvers:

- **MS-DIS** – lightweight Multi-Shot ASP **dispute-based** solver
- **flexABle** [Diller, Gaggl, Gorczyca 2022] – state-of-the-art ABA **dispute-based** solver, written in Scala
- **aspforaba** [Lehtonen, Wallner, Järvisalo 2023] – **reduction-based** one-shot ASP solver – upper bound on performance

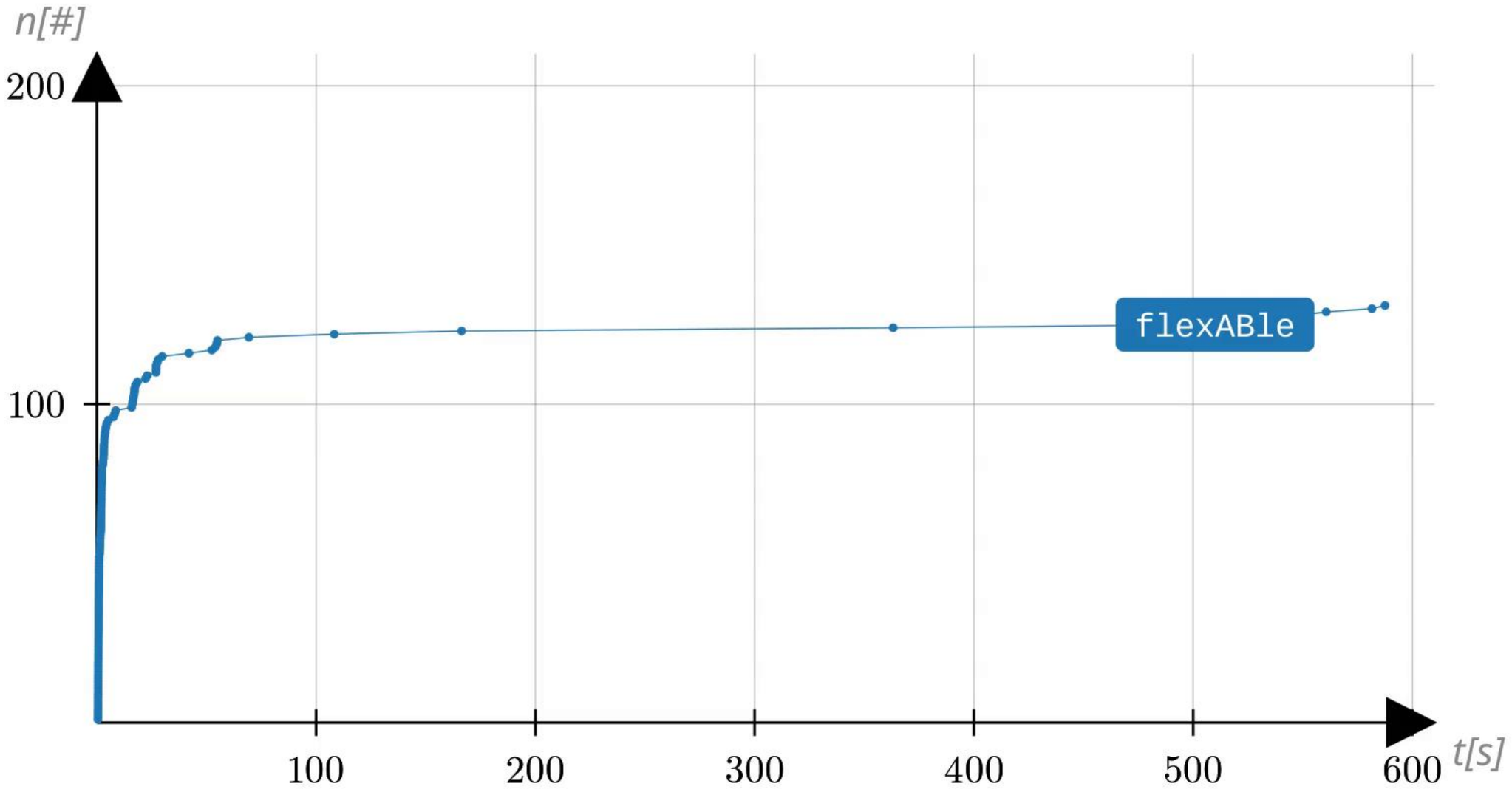
Setup:

- ICCMA'23 ABA track instances (containing 25-5000 atoms)
- 600s timeout

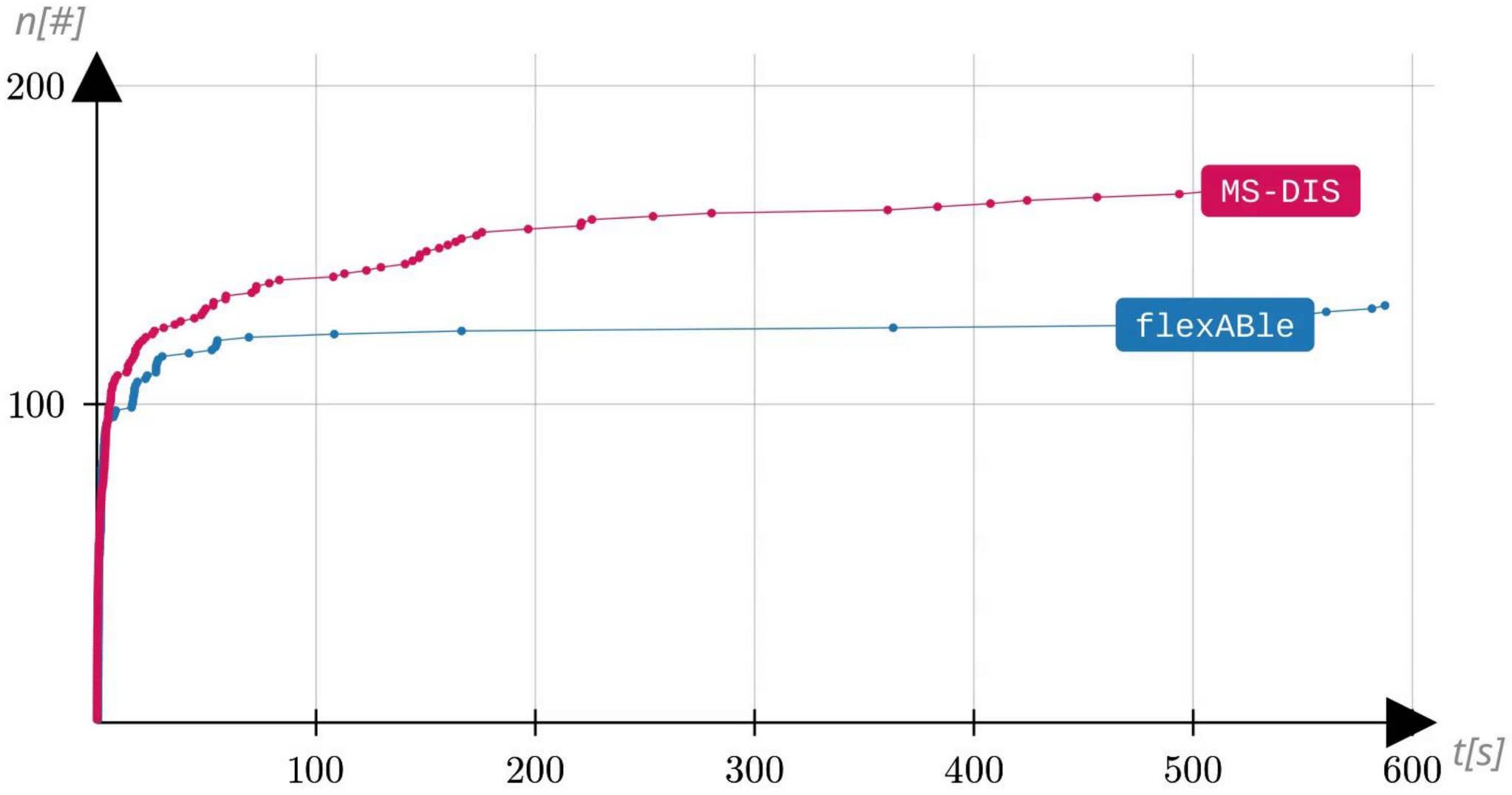
Evaluation (1)



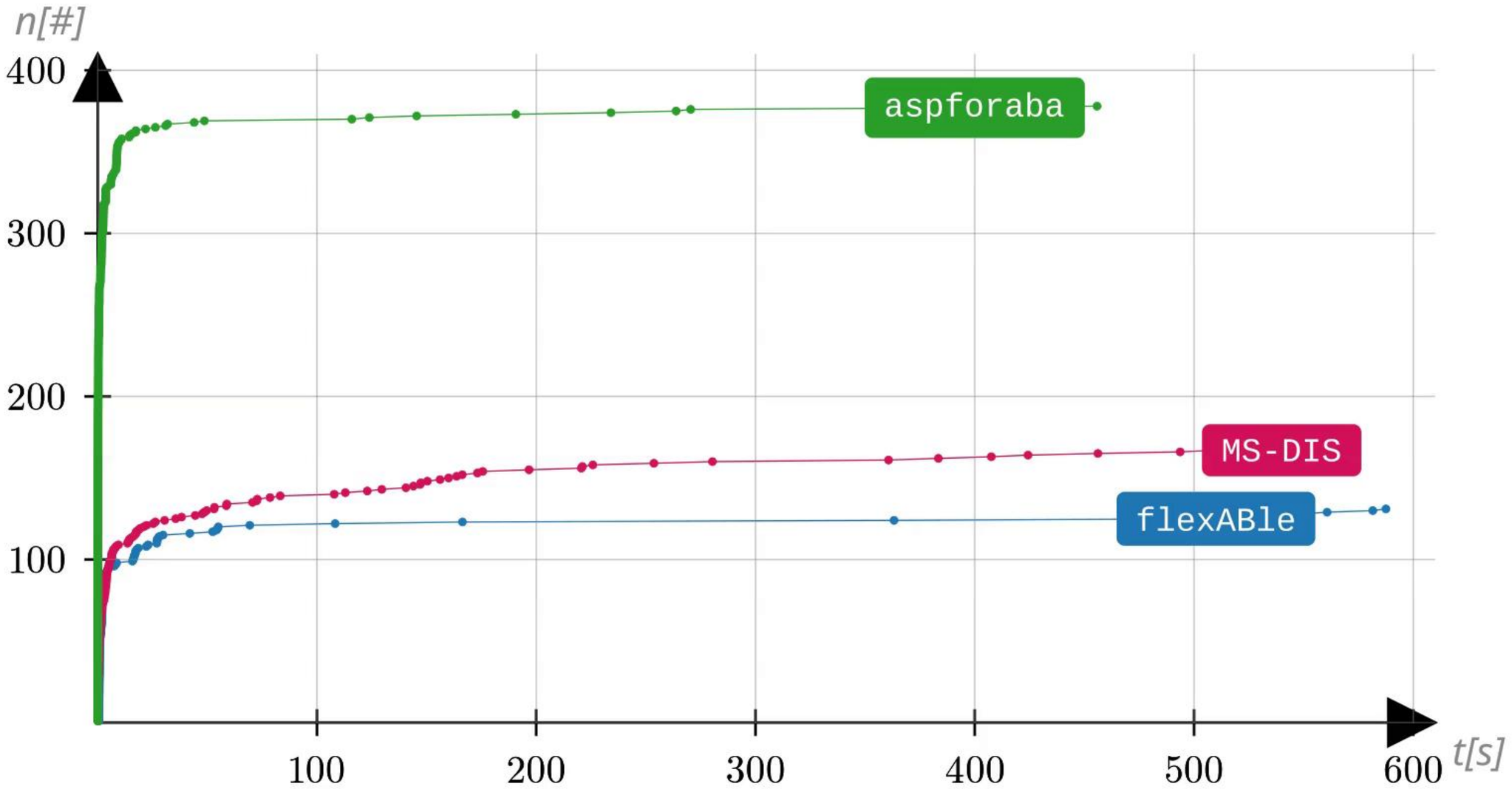
Evaluation (1)



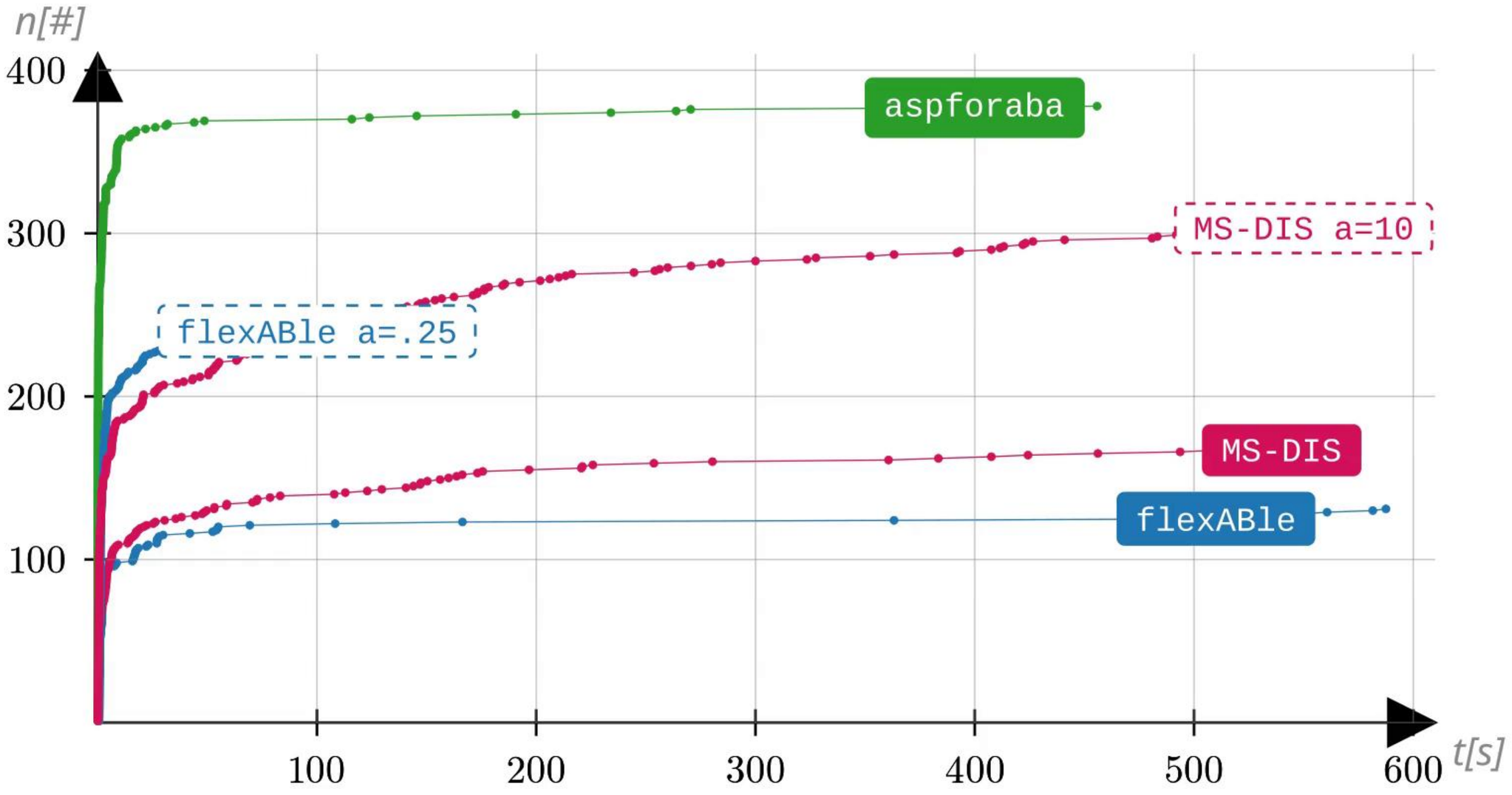
Evaluation (1)



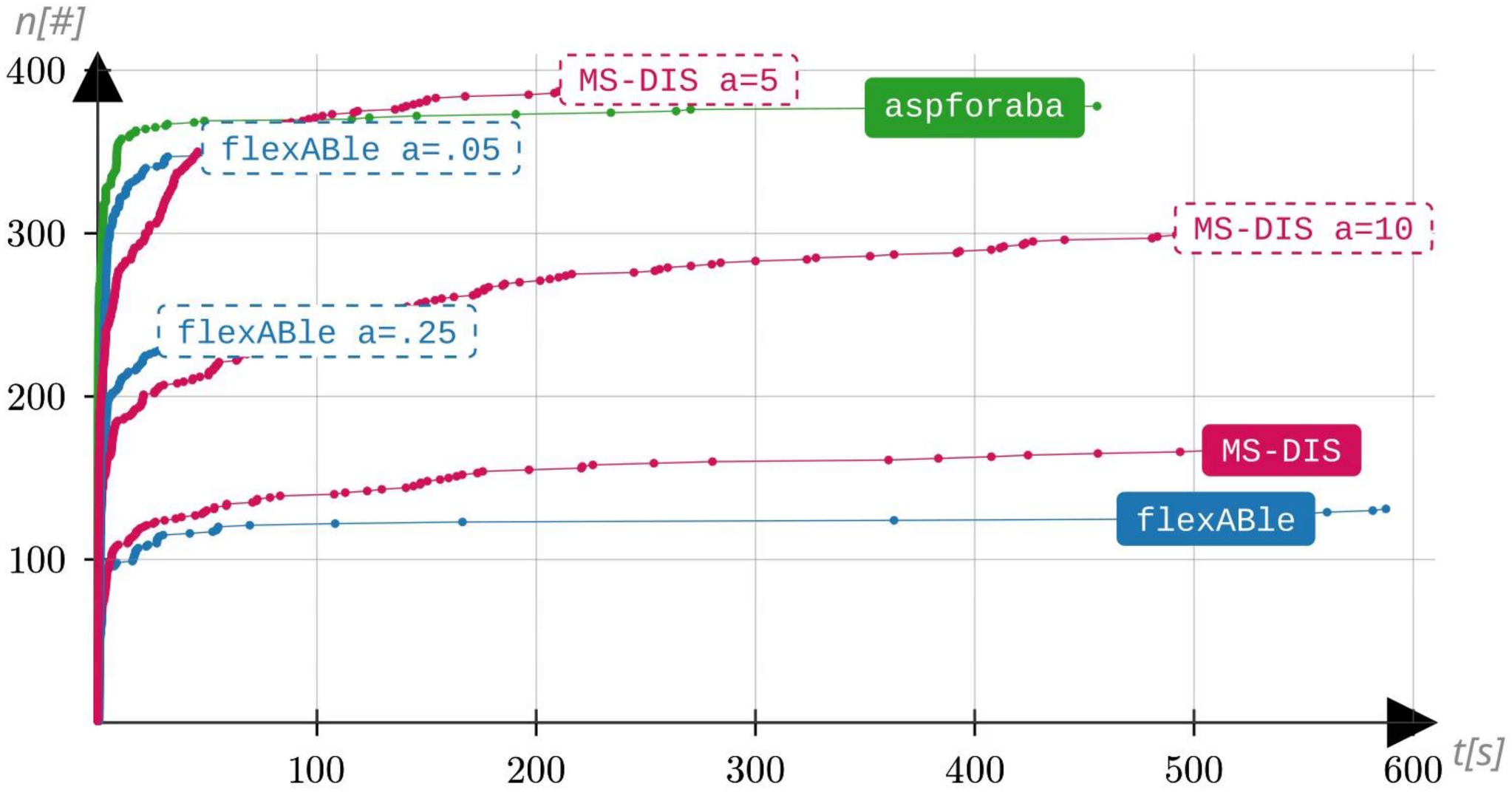
Evaluation (1)



Evaluation (1)



Evaluation (1)



Conclusion

Conclusion

- **first multi-shot ASP implementation** of argument games (ABA & AFs)

Conclusion

- **first multi-shot ASP implementation** of argument games (ABA & AFs)
- modular and declarative design separating **formal definitions** (in ASP) from **procedural control** (in Python)

Conclusion

- **first multi-shot ASP implementation** of argument games (ABA & AFs)
- modular and declarative design separating **formal definitions** (in ASP) from **procedural control** (in Python)
- easily extensible – **interactivity**, **visualization** (via `clingraph` [Hahn et al. 2024]), **additional semantics** (e.g. stable) can be added by providing only a few extra ASP rules

Conclusion

- **first multi-shot ASP implementation** of argument games (ABA & AFs)
- modular and declarative design separating **formal definitions** (in ASP) from **procedural control** (in Python)
- easily extensible – **interactivity**, **visualization** (via `clingraph` [Hahn et al. 2024]), **additional semantics** (e.g. stable) can be added by providing only a few extra ASP rules
- **outperforms** existing dispute-based ABA systems (e.g. flexABle)

Conclusion

Technische Universität Dresden

- **first multi-shot ASP implementation** of argument games (ABA & AFs)
- modular and declarative design separating **formal definitions** (in ASP) from **procedural control** (in Python)
- easily extensible – **interactivity**, **visualization** (via `clingraph` [Hahn et al. 2024]), **additional semantics** (e.g. stable) can be added by providing only a few extra ASP rules
- **outperforms** existing dispute-based ABA systems (e.g. flexABle)
- suggests multi-shot ASP as a **general, adaptable framework** for implementing and comparing argument games across formalisms

Extension: Stable Semantics

Technische Universität Dresden

```
#program updateState(t).
% add another option to perform "PF2" move
pm(t,p,pf2,A) :- at(ds), assumption(A), not cul(_,A), not stS(_,A,p),
  not contrary(A,A), not contrary(D,A) : def(_,D).
% define: remaining assumptions - neither defences, nor culprits
remA(t,A) :- tt(ts), assumption(A), not def(_,A), not cul(_,A).
% modify termination criteria:  require no assumption be remaining
term(t) :- tt(ts), g(G), comS(_,G,p), comS(_,CC,p) : culCtr(_,CC);
  not unbloComS(t,DC,o) : defCtr(_,DC); not remA(t,A) : assumption(A).
```

Extension: Visualisation

Technische Universität Dresden

```
graph(m).
graph((A,l),m) :- arg(A).
node(A,(A,l)) :- arg(A).
edge((A,B),m) :- att(A,B).

attr(node,A,fillcolor,"green") :- m(_,p,A).
attr(node,A,fillcolor,"yellow") :- m(_,o,A).
attr(graph,(A,l),label,T) :- m(T,_,A).

ldefeat(Tm,A) :- defeat(Tm,A), not m(_,_,A),
  Tm = #min { T : defeat(T,A) }.
attr(graph,(A,l),label,Tm) :- ldefeat(Tm,A).
attr(node,A,fillcolor,"red") :- ldefeat(_,A).

lpm(Tm,P,A) :- pm(Tm,P,A), not m(_,_,A), not defeat(_,A),
  Tm = #max { T : pm(T,_,A) }.
attr(graph,(A,l),label,T) :- lpm(T,_,A).
attr(node,A,fillcolor,"lightblue") :- lpm(_,_,A).
```

Evaluation (2)

	aspforaba	MD-DIS			flexABble		
			a=10	a=5		a=.25	a=.05
#t-out.	3	214	81	0	250	146	43
#inc.	0	0	30	153	0	69	114
time[h]	1	38	19	2	43	25	8
%acc. t	99	44	71	60	34	44	59