aspic: Interactive Answer Set Programming

Martin Gebser Philipp Obermeier Torsten Schaub

Outline

Introduction

- Goal Exploration and modification of ASP knowledgebases
- *aspic* user-oriented interactive ASP shell
  - Dynamically load, define, change logic programs
  - Operative ASP solving process
  - Stateful system with state-changing operators and queries
  - Based on *clingo* 4 and its Python API

Outline

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*

  Multi-shot solving: *ground* | *solve*

  ➥ *continuously changing logic programs*

Agents, Assisted Living, Robotics, Planning, Query-answering, etc

clingo 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*

  Multi-shot solving: *ground* | *solve*

  ➥ *continuously changing logic programs*

Agents, Assisted Living, Robotics, Planning, Query-answering, etc

clingo 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: *ground* | *solve*
  - ➥ *continuously changing logic programs*
- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: *ground* | *solve*
  - ➥ *continuously changing logic programs*
- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: *ground** | *solve**
  - ➥ *continuously changing logic programs*
- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*ground** | *solve**)*
  - ➥ *continuously changing logic programs*
- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**)*
  - ➥ *continuously changing logic programs*
- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory*)*
  - ➥ *continuously changing logic programs*
- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*
  - ➥ *continuously changing logic programs*
- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*
  - ➥ *continuously changing logic programs*
- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc
- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*

  ➥ *continuously changing logic programs*

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4

Motivation

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*

  ➥ *continuously changing logic programs*

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4

Clingo = ASP + Control

- ASP

```
#program <name> [ (<parameters>) ]
          #program play(t).
#external <atom> [ :  <body> ]
          #external mark(X,Y,P,t) :  field(X,Y), player(P).
```

- Control

Lua (www.lua.org)

```
          prg:solve(), prg:ground(parts), ...
```

Python (www.python.org)

```
          prg.solve(), prg.ground(parts), ...
```

- Integration

in ASP: embedded scripting language (#script)
in Lua/Python: library import (import gringo)

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    `prg.solve(), prg.ground(parts), ...`
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    `prg.solve(), prg.ground(parts), ...`
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
- Integration

  in ASP: embedded scripting language (`#script`)
  in Lua/Python: library import (`import gringo`)

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve()`, `prg:ground(parts)`, ...
  - Python (`www.python.org`)
    - Example `prg.solve()`, `prg.ground(parts)`, ...
- Integration

  in ASP: embedded scripting language (`#script`)
  in Lua/Python: library import (`import gringo`)

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Vanilla *clingo*

- Emulating *clingo* in *clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Vanilla *clingo*

- Emulating *clingo* in *clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Vanilla *clingo*

- Emulating *clingo* in *clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Hello world!

```
#script (python)
def main(prg):
    print("Hello world!")
#end.
```

```
$ clingo hello.lp
clingo version 4.5.0
Reading from hello.lp
Hello world!
UNKNOWN

Models       : 0+
Calls        : 1
Time         : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Hello world!

```
#script (python)
def main(prg):
    print("Hello world!")
#end.
```

```
$ clingo hello.lp
clingo version 4.5.0
Reading from hello.lp
Hello world!
UNKNOWN

Models       : 0+
Calls        : 1
Time         : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Hello world!

```
#script (python)
def main(prg):
    print("Hello world!")
#end.
```

```
$ clingo hello.lp
clingo version 4.5.0
Reading from hello.lp
Hello world!
UNKNOWN

Models       : 0+
Calls        : 1
Time         : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Hello world!

```
#script (python)
def main(prg):
    print("Hello world!")
#end.
```

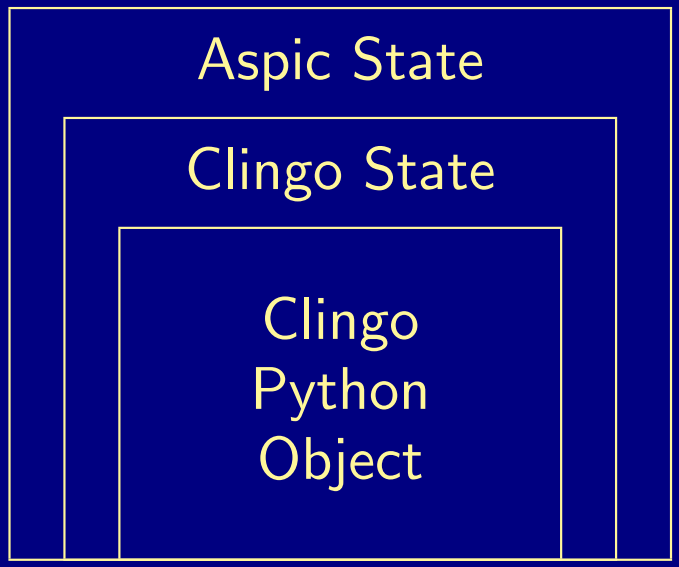
```
$ clingo hello.lp
clingo version 4.5.0
Reading from hello.lp
Hello world!
UNKNOWN

Models       : 0+
Calls        : 1
Time         : 0.009s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Outline

Architecture

- `Class` `Control` object for the grounding/solving process
- `Methods` `__init__`, `add`, `load`, `ground`, `solve`, `assign_external`, `release_external`...



- Clingo state $(\boldsymbol{R}, \mathbb{P}, V)$
  - $\boldsymbol{R}$ is a collection of extensible (non-ground) logic programs
  - $\mathbb{P}$ is a module
  - V is a three-valued assignment over the input atoms of $\mathbb{P}$
- Operations create, add, ground, solve, assignExternal, releaseExternal

Architecture

- `Class Control` object for the grounding/solving process
- `Methods __init__, add, load, ground, solve, assign_external, release_external...`

- Clingo state $(\boldsymbol{R}, \mathbb{P}, V)$
  - $\boldsymbol{R}$ is a collection of extensible (non-ground) logic programs
  - $\mathbb{P}$ is a module
  - V is a three-valued assignment over the input atoms of $\mathbb{P}$
- Operations create, add, ground, solve, assignExternal, releaseExternal

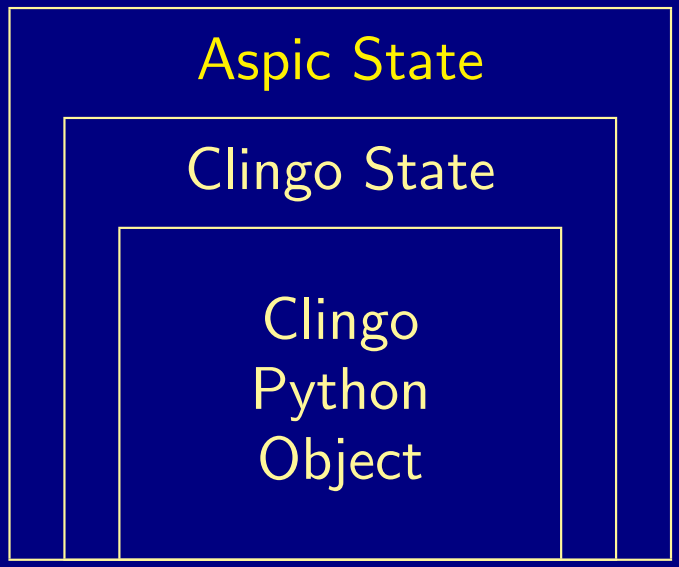
Architecture

- `Class Control` object for the grounding/solving process
- `Methods __init__, add, load, ground, solve, assign_external, release_external...`

- Clingo state $(\boldsymbol{R}, \mathbb{P}, V)$
  - $\boldsymbol{R}$ is a collection of extensible (non-ground) logic programs
  - $\mathbb{P}$ is a module
  - V is a three-valued assignment over the input atoms of $\mathbb{P}$
- Operations create, add, ground, solve, assignExternal, releaseExternal

Architecture

- `Class Control` object for the grounding/solving process
- `Methods __init__, add, load, ground, solve, assign_external, release_external...`



- Clingo state $(\boldsymbol{R}, \mathbb{P}, V)$
  - $\boldsymbol{R}$ is a collection of extensible (non-ground) logic programs
  - $\mathbb{P}$ is a module
  - V is a three-valued assignment over the input atoms of $\mathbb{P}$
- Operations create, add, ground, solve, assignExternal, releaseExternal

System States

- Aspic State (R, I, i, j) with where
  - R is a ground program over a set of ground atoms $\mathcal{A}$
  - $I \subseteq \mathcal{A} \setminus \mathit{head}(R)$ is a set of input atoms
  - i is a three-valued truth assignment over I
  - j is a three-valued truth assignment over $\mathcal{A}$
- State-induced logic program

$$P(R, I, i, j) = R \cup \{a \leftarrow \mid a \in i^t\} \cup \{\{a\} \leftarrow \mid a \in i^u\}$$
$$\cup \{\leftarrow \sim a \mid a \in j^t\} \cup \{\leftarrow a \mid a \in j^f\}$$

- Note Intuitively, i changes the stable models of $P(R, I, i, j)$, whereas j only filters them

System States

- Aspic State (R, I, i, j) with where
  - R is a ground program over a set of ground atoms $\mathcal{A}$
  - $I \subseteq \mathcal{A} \setminus \mathit{head}(R)$ is a set of input atoms
  - i is a three-valued truth assignment over I $\quad (i : \mathcal{A} \to \{t, f, u\})$
  - j is a three-valued truth assignment over $\mathcal{A}$ $\quad (j : \mathcal{A} \to \{t, f, u\})$
- State-induced logic program

$$P(R, I, i, j) = R \cup \{a \leftarrow \mid a \in i^t\} \cup \{\{a\} \leftarrow \mid a \in i^u\}$$
$$\cup \{\leftarrow \sim a \mid a \in j^t\} \cup \{\leftarrow a \mid a \in j^f\}$$

- Note Intuitively, i changes the stable models of $P(R, I, i, j)$, whereas j only filters them

System States

- Aspic State (R, I, i, j) with where
  - R is a ground program over a set of ground atoms $\mathcal{A}$
  - $I \subseteq \mathcal{A} \setminus \mathit{head}(R)$ is a set of input atoms
  - i is a three-valued truth assignment over I
  - j is a three-valued truth assignment over $\mathcal{A}$
- State-induced logic program

$$P(R, I, i, j) = R \cup \{a \leftarrow \mid a \in i^t\} \cup \{\{a\} \leftarrow \mid a \in i^u\}$$
$$\cup \{ \leftarrow \sim a \mid a \in j^t\} \cup \{ \leftarrow a \mid a \in j^f \}$$

- Note Intuitively, i changes the stable models of $P(R, I, i, j)$, whereas j only filters them

System States

- Aspic State (R, I, i, j) with where
  - R is a ground program over a set of ground atoms $\mathcal{A}$
  - $I \subseteq \mathcal{A} \setminus \mathit{head}(R)$ is a set of input atoms
  - i is a three-valued truth assignment over I
  - j is a three-valued truth assignment over $\mathcal{A}$
- State-induced logic program

$$P(R, I, i, j) = R \cup \{a \leftarrow \mid a \in i^t\} \cup \{\{a\} \leftarrow \mid a \in i^u\}$$
$$\cup \{ \leftarrow \sim a \mid a \in j^t\} \cup \{ \leftarrow a \mid a \in j^f \}$$

- Note Intuitively, i changes the stable models of $P(R, I, i, j)$, whereas j only filters them

State-changing Operators at a glance

- *assume* and *cancel*
  - manipulate j
  - *assume* adds and *cancel* removes literals from j
- *assert* *retract*, and *open*
  - manipulate i
  - *assert* sets an input atom to t, *open* to u and *retract* to f
- *external* and *release*
  - manipulate I (and R)
  - *external* adds a new input atom to I and *release* removes it permanently
- *define*
  - manipulates R (and I)
  - *define* adds a new rule set to R

Outline

Assume and Cancel

- *assume* : $\langle \ell, (R, I, i, j_1) \rangle \mapsto (R, I, i, j_2)$
  - Takes ground literal ℓ
  - j_2 maps ℓ to t, if $\ell \in \mathcal{A}$, otherwise to f
- *cancel* : $\langle \ell, (R, I, i, j_1) \rangle \mapsto (R, I, i, j_2)$
  - Takes ground literal ℓ
  - j_2 maps ℓ to u, if $\ell \in \mathcal{A}$; otherwise, it maps $\bar{\ell}$ to u

Assume and Cancel

- *assume* : $\langle \ell, (R, I, i, j_1) \rangle \mapsto (R, I, i, j_2)$
  - Takes ground literal ℓ
  - j_2 maps ℓ to t, if $\ell \in \mathcal{A}$, otherwise to f
- *cancel* : $\langle \ell, (R, I, i, j_1) \rangle \mapsto (R, I, i, j_2)$
  - Takes ground literal ℓ
  - j_2 maps ℓ to u, if $\ell \in \mathcal{A}$; otherwise, it maps $\bar{\ell}$ to u

Assert, Retract and Open

- $\mathit{assert} : \langle a, (R, I, i_1, j)\rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to t, if $a \in I$
- $\mathit{open} : \langle a, (R, I, i_1, j)\rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to u, if $a \in I$
- $\mathit{retract} : \langle a, (R, I, i_1, j)\rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to f, if $a \in I$

Assert, Retract and Open

- $\mathit{assert} : \langle a, (R, I, i_1, j) \rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to t, if $a \in I$
- $\mathit{open} : \langle a, (R, I, i_1, j) \rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to u, if $a \in I$
- $\mathit{retract} : \langle a, (R, I, i_1, j) \rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to f, if $a \in I$

Assert, Retract and Open

- $\mathit{assert} : \langle a, (R, I, i_1, j) \rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to t, if $a \in I$
- $\mathit{open} : \langle a, (R, I, i_1, j) \rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to u, if $a \in I$
- $\mathit{retract} : \langle a, (R, I, i_1, j) \rangle \mapsto (R, I, i_2, j)$
  - Takes ground atom a
  - i_2 maps a to f, if $a \in I$

Define, External, Release

- $\mathit{define} : \langle R, (R_1, I_1, i, j)\rangle \mapsto (R_2, I_2, i, j)$
  - Takes set of ground rules R
  - $R_2 = C_{I_1}(R_1 \cup R)$, if R_1 and R are "modularly compositional"
    - $C_{I_1}(R_1 \cup R)$ "confines $R_1 \cup R$ to its atoms"
  - $I_2 = I_1 \setminus \mathit{head}(R_2)$

- $\mathit{external} : \langle a, (R, I_1, i, j)\rangle \mapsto (R, I_2, i, j)$
  - Takes ground atom a
  - $I_2 = I_1 \cup (\{a\} \setminus \mathit{head}(R))$

- $\mathit{release} : \langle a, (R_1, I_1, i_1, j)\rangle \mapsto (R_2, I_2, i_2, j)$
  - Takes ground atom a
  - $I_2 = I_1 \setminus \{a\}$ and $i_2^v = i_1^v \setminus \{a\}$ for $v \in \{t, f, u\}$
  - $R_2 = R_1 \cup \{a \leftarrow a\}$, if $a \in I_1$, and $R_2 = R_1$ otherwise

Define, External, Release

- $\mathit{define} : \langle R, (R_1, I_1, i, j)\rangle \mapsto (R_2, I_2, i, j)$
  - Takes set of ground rules R
  - $R_2 = C_{I_1}(R_1 \cup R)$, if R_1 and R are "modularly compositional"
    - $C_{I_1}(R_1 \cup R)$ "confines $R_1 \cup R$ to its atoms"
  - $I_2 = I_1 \setminus \mathit{head}(R_2)$

- $\mathit{external} : \langle a, (R, I_1, i, j)\rangle \mapsto (R, I_2, i, j)$
  - Takes ground atom a
  - $I_2 = I_1 \cup (\{a\} \setminus \mathit{head}(R))$

- $\mathit{release} : \langle a, (R_1, I_1, i_1, j)\rangle \mapsto (R_2, I_2, i_2, j)$
  - Takes ground atom a
  - $I_2 = I_1 \setminus \{a\}$ and $i_2^v = i_1^v \setminus \{a\}$ for $v \in \{t, f, u\}$
  - $R_2 = R_1 \cup \{a \leftarrow a\}$, if $a \in I_1$, and $R_2 = R_1$ otherwise

Define, External, Release

- $\mathit{define} : \langle R, (R_1, I_1, i, j)\rangle \mapsto (R_2, I_2, i, j)$
  - Takes set of ground rules R
  - $R_2 = C_{I_1}(R_1 \cup R)$, if R_1 and R are "modularly compositional"
    - $C_{I_1}(R_1 \cup R)$ "confines $R_1 \cup R$ to its atoms"
  - $I_2 = I_1 \setminus \mathit{head}(R_2)$

- $\mathit{external} : \langle a, (R, I_1, i, j)\rangle \mapsto (R, I_2, i, j)$
  - Takes ground atom a
  - $I_2 = I_1 \cup (\{a\} \setminus \mathit{head}(R))$

- $\mathit{release} : \langle a, (R_1, I_1, i_1, j)\rangle \mapsto (R_2, I_2, i_2, j)$
  - Takes ground atom a
  - $I_2 = I_1 \setminus \{a\}$ and $i_2^v = i_1^v \setminus \{a\}$ for $v \in \{t, f, u\}$
  - $R_2 = R_1 \cup \{a \leftarrow a\}$, if $a \in I_1$, and $R_2 = R_1$ otherwise

Outline

Queries

- A filter maps a collection of sets of ground atoms to a subset of the collection
- A entailment mode maps a collection of sets of ground atoms to a subset of the union of the collection
- *query* maps an atomic query $q \in \mathcal{A}$, an entailment mode μ, a filter ν, and a system state $S = (R, I, i, j)$ to the set $\{yes, no\}$:

$$query(q, (\mu, \nu), S) = \begin{cases} yes & \text{if } q \in \mu \circ \nu(AS(P(S))) \\ no & \text{otherwise} \end{cases}$$

- Boolean queries: Boolean expression in negation normal form
- Non-ground conjunctive queries: conjunction of (non-ground) literals

Queries

- A filter maps a collection of sets of ground atoms to a subset of the collection
- A entailment mode maps a collection of sets of ground atoms to a subset of the union of the collection
- *query* maps an atomic query $q \in \mathcal{A}$, an entailment mode μ, a filter ν, and a system state $S = (R, I, i, j)$ to the set $\{yes, no\}$:

$$query(q, (\mu, \nu), S) = \begin{cases} yes & \text{if } \; q \in \mu \circ \nu(AS(P(S))) \\ no & \text{otherwise} \end{cases}$$

- Boolean queries: Boolean expression in negation normal form
- Non-ground conjunctive queries: conjunction of (non-ground) literals

Queries

- A filter maps a collection of sets of ground atoms to a subset of the collection
- A entailment mode maps a collection of sets of ground atoms to a subset of the union of the collection
- *query* maps an atomic query $q \in \mathcal{A}$, an entailment mode μ, a filter ν, and a system state $S = (R, I, i, j)$ to the set $\{yes, no\}$:

$$query(q, (\mu, \nu), S) = \begin{cases} yes & \text{if } \; q \in \mu \circ \nu(AS(P(S))) \\ no & \text{otherwise} \end{cases}$$

- Boolean queries: Boolean expression in negation normal form
- Non-ground conjunctive queries: conjunction of (non-ground) literals

Outline

Demo

Outline

Summary

- Release forseen for late summer