# Generating Answer Set Justifications with xclingo

Pedro Cabalar, Jorge Fandinno, Brais Muñiz

September 7, 2022

# ASP lack of human-readable explanations

- In ASP, we obtain solutions to logic programs called answer sets.

```
Answer:  1
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0)
h(s1,open,0) h(s2,open,0) h(s2,open,1) h(s1,open,1) h(relayline,off,1)
h(protect,on,1) h(light,off,1) h(protect,on,2) h(s1,closed,2) h(s2,closed,2)
Answer:  2
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0)
(...)
```

# ASP lack of human-readable explanations

- In ASP, we obtain solutions to logic programs called answer sets.

```
Answer:  1
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0)
h(s1,open,0) h(s2,open,0) h(s2,open,1) h(s1,open,1) h(relayline,off,1)
h(protect,on,1) h(light,off,1) h(protect,on,2) h(s1,closed,2) h(s2,closed,2)
Answer:  2
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0)
(...)
```

- Declarativeness: we lose HOW (not true for very simple programs).

# ASP lack of human-readable explanations

- In ASP, we obtain solutions to logic programs called answer sets.

```
Answer:  1
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0)
h(s1,open,0) h(s2,open,0) h(s2,open,1) h(s1,open,1) h(relayline,off,1)
h(protect,on,1) h(light,off,1) h(protect,on,2) h(s1,closed,2) h(s2,closed,2)
Answer:  2
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0)
(...)
```

- Declarativeness: we lose HOW (not true for very simple programs).
- Which leads to a lack of explainability.

# ASP lack of human-readable explanations

- In ASP, we obtain solutions to logic programs called answer sets.

```
Answer:  1
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0)
h(s1,open,0) h(s2,open,0) h(s2,open,1) h(s1,open,1) h(relayline,off,1)
h(protect,on,1) h(light,off,1) h(protect,on,2) h(s1,closed,2) h(s2,closed,2)
Answer:  2
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0)
(...)
```

- Declarativeness: we lose HOW (not true for very simple programs).
- Which leads to a lack of explainability.
- Not only explainability: debugging and visualization of solutions can be tricky sometimes.

# xclingo: eXplainable clingo

- Generates explanations from ASP programs
  - Provides text-based, human readable explanations.

# xclingo: eXplainable clingo

- Generates explanations from ASP programs
  - Provides text-based, human readable explanations.
  - User-defined text labels (*annotations*) or automatic ones

# `xclingo`: eXplainable clingo

- Generates explanations from ASP programs
  - Provides text-based, human readable explanations.
  - User-defined text labels (*annotations*) or automatic ones
  - User chooses the detail level, enabling both debugging and causal explanation.

# xclingo: eXplainable clingo

- Generates explanations from ASP programs
  - Provides text-based, human readable explanations.
  - User-defined text labels (*annotations*) or automatic ones
  - User chooses the detail level, enabling both debugging and causal explanation.
  - Annotations do not affect the original semantics. They are ASP comments (start with %).

# xclingo: eXplainable clingo

- Generates explanations from ASP programs
  - Provides text-based, human readable explanations.
  - User-defined text labels (*annotations*) or automatic ones
  - User chooses the detail level, enabling both debugging and causal explanation.
  - Annotations do not affect the original semantics. They are ASP comments (start with %).
  - Constraints can also be explained.

# xclingo: eXplainable clingo

- Generates explanations from ASP programs
  - Provides text-based, human readable explanations.
  - User-defined text labels (*annotations*) or automatic ones
  - User chooses the detail level, enabling both debugging and causal explanation.
  - Annotations do not affect the original semantics. They are ASP comments (start with %).
  - Constraints can also be explained.

```
$clingo prog.lp
```

```
Solving...
Answer: 1
o(toggle(s1),2) o(toggle(s1),5) h(light,off,0) h(protect,on,0) h(relayline,off,0) h(s1,open,0) h(s2,open,0) h(s2,ope
n,1) h(s1,open,1) h(relayline,off,1) h(protect,on,1) h(light,off,1) h(protect,on,2) h(s1,closed,2) h(s2,closed,2) h(
light,on,2) h(relayline,on,2) h(relayline,on,3) h(light,on,3) h(s2,closed,3) h(s1,closed,3) h(protect,on,3) h(protec
t,on,4) h(s1,closed,4) h(s2,closed,4) h(light,on,4) h(relayline,on,4) h(light,on,5) h(s2,closed,5) h(s1,open,5) h(re
layline,off,5) h(protect,on,5) h(protect,on,6) h(relayline,off,6) h(s1,open,6) h(s2,closed,6) h(light,on,6) h(light,
on,7) h(s2,closed,7) h(s1,open,7) h(relayline,off,7) h(protect,on,7) h(protect,on,8) h(relayline,off,8) h(s1,open,8)
 h(s2,closed,8) h(light,on,8)
SATISFIABLE
```

# xclingo: eXplainable clingo

- Generates explanations from ASP programs
  - Provides text-based, human readable explanations.
  - User-defined text labels (*annotations*) or automatic ones
  - User chooses the detail level, enabling both debugging and causal explanation.
  - Annotations do not affect the original semantics. They are ASP comments (start with %).
  - Explains fired constraints.

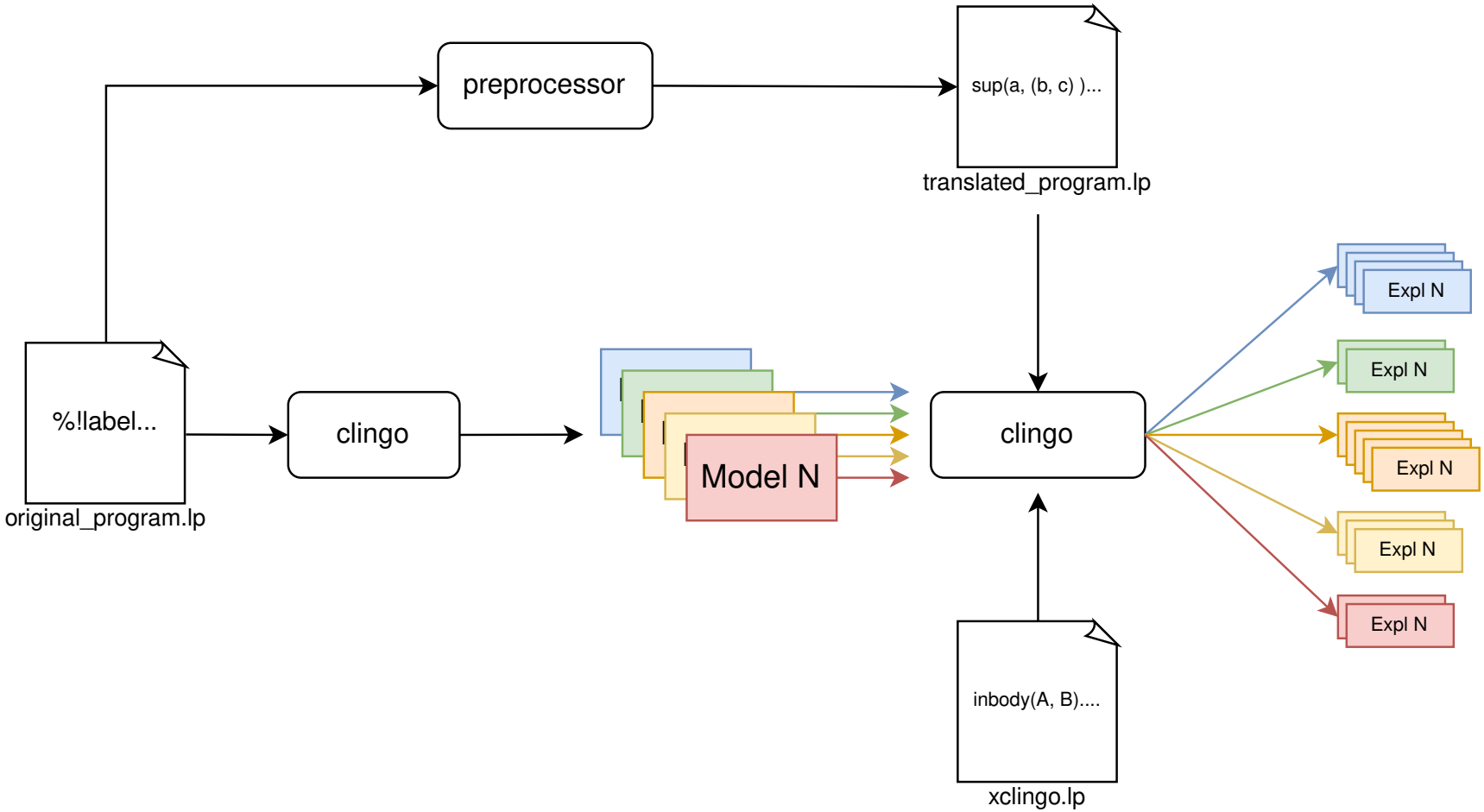
```
$xclingo prog.lp
```

```
Answer 1
  *
  |__The relay stopped working at 5
  |  |__s1 was open at 5
  |  |  |__s1 was closed at 2
  |  |  |  |__s1 was initially open
  |  |  |  |__The agent toggled s1 at 2
  |  |  |__The agent toggled s1 at 5
```

# Contents

# N Explanations for each Answer Set

Note that one answer set may have more than one explanation.



# Labelled logic program

A labelled logic program is a set of labelled rules of the form:

$$\ell : H \leftarrow B \wedge N \tag{1}$$

# Labelled logic program

A labelled logic program is a set of labelled rules of the form:

$$\ell : H \leftarrow B \wedge N \tag{1}$$

If $r$ is a rule of the form (1):

- $Lb(r) \stackrel{\text{df}}{=} \ell$
- $H(r) \stackrel{\text{df}}{=} H$
- $Body(r) \stackrel{\text{df}}{=} B \wedge N$
- $B^+(r) \stackrel{\text{df}}{=} B$
- $B^-(r) \stackrel{\text{df}}{=} N$

# Explanation: formal definition

Let $P$ be a labelled program and $I \models P$ a model of $P$.

# Explanation: formal definition

Let $P$ be a labelled program and $I \models P$ a model of $P$.
An explanation $G$ of $I$ under $P$ is a labelled directed graph $G = \langle I, E, \lambda \rangle$

# Explanation: formal definition

Let $P$ be a labelled program and $I \models P$ a model of $P$.
An explanation $G$ of $I$ under $P$ is a labelled directed graph $G = \langle I, E, \lambda \rangle$

- whose vertices are the atoms in $I$
- the edges in $E \subseteq I \times I$ connect pairs of atoms
- the function $\lambda : I \to \Lambda_P$ assigns a label to each atom

# Explanation: formal definition

Let $P$ be a labelled program and $I \models P$ a model of $P$.
An explanation $G$ of $I$ under $P$ is a labelled directed graph $G = \langle I, E, \lambda \rangle$

- whose vertices are the atoms in $I$
- the edges in $E \subseteq I \times I$ connect pairs of atoms
- the function $\lambda : I \to \Lambda_P$ assigns a label to each atom

An explanation $G = \langle I, E, \lambda \rangle$, satisfies:

# Explanation: formal definition

Let $P$ be a labelled program and $I \models P$ a model of $P$.
An explanation $G$ of $I$ under $P$ is a labelled directed graph $G = \langle I, E, \lambda \rangle$

- whose vertices are the atoms in $I$
- the edges in $E \subseteq I \times I$ connect pairs of atoms
- the function $\lambda : I \to \Lambda_P$ assigns a label to each atom

An explanation $G = \langle I, E, \lambda \rangle$, satisfies:

1. $G$ is acyclic

# Explanation: formal definition

Let $P$ be a labelled program and $I \models P$ a model of $P$.
An explanation $G$ of $I$ under $P$ is a labelled directed graph $G = \langle I, E, \lambda \rangle$

- whose vertices are the atoms in $I$
- the edges in $E \subseteq I \times I$ connect pairs of atoms
- the function $\lambda : I \to \Lambda_P$ assigns a label to each atom

An explanation $G = \langle I, E, \lambda \rangle$, satisfies:

1. $G$ is acyclic
2. It contains no repeated labels: $\lambda(p) \neq \lambda(q)$ for every pair $p, q \in I$

# Explanation: formal definition

Let $P$ be a labelled program and $I \models P$ a model of $P$.
An explanation $G$ of $I$ under $P$ is a labelled directed graph $G = \langle I, E, \lambda \rangle$

- whose vertices are the atoms in $I$
- the edges in $E \subseteq I \times I$ connect pairs of atoms
- the function $\lambda : I \to \Lambda_P$ assigns a label to each atom

An explanation $G = \langle I, E, \lambda \rangle$, satisfies:

1. $G$ is acyclic
2. It contains no repeated labels: $\lambda(p) \neq \lambda(q)$ for every pair $p, q \in I$
3. for every $p \in I$, the rule $r$ such that $Lb(r) = \lambda(p)$ satisfies:
   $r \in Sup_I(P, p)$ and $B^+(r) = \{q \mid (q, p) \in E\}$.

# Examples (1)

Consider the program

$$p \tag{2}$$
$$q \leftarrow p \tag{3}$$
$$r \leftarrow p, q \tag{4}$$

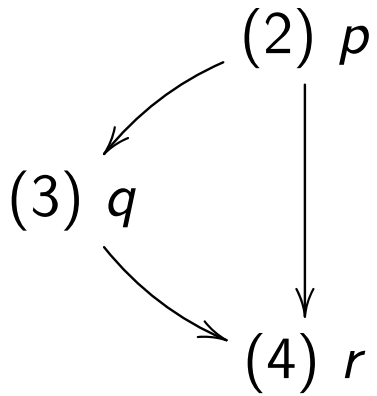
One answer set: $\{p, q, r\}$.

# Examples (1)

Consider the program

$$p \tag{2}$$
$$q \leftarrow p \tag{3}$$
$$r \leftarrow p, q \tag{4}$$

One answer set: $\{p, q, r\}$.One explanation:



# Examples (2)

Consider the program

$$p \vee q \tag{5}$$

$$q \leftarrow p \tag{6}$$

$$p \leftarrow q \tag{7}$$

One answer set: $\{p, q\}$.

# Examples (2)

Consider the program

$$p \vee q \tag{5}$$

$$q \leftarrow p \tag{6}$$

$$p \leftarrow q \tag{7}$$

One answer set: $\{p, q\}$.Two explanations:

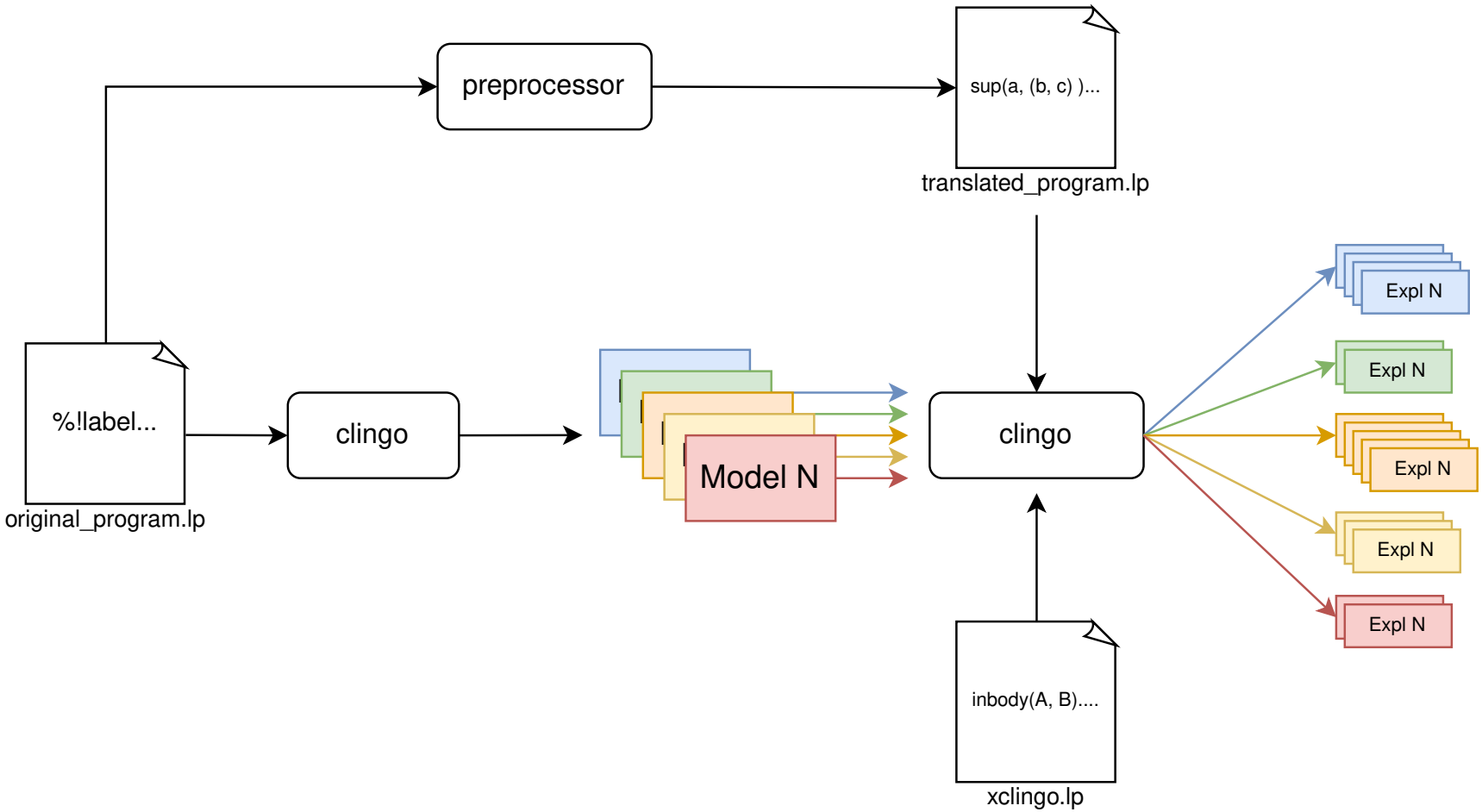
# Contents

# Time for a demo

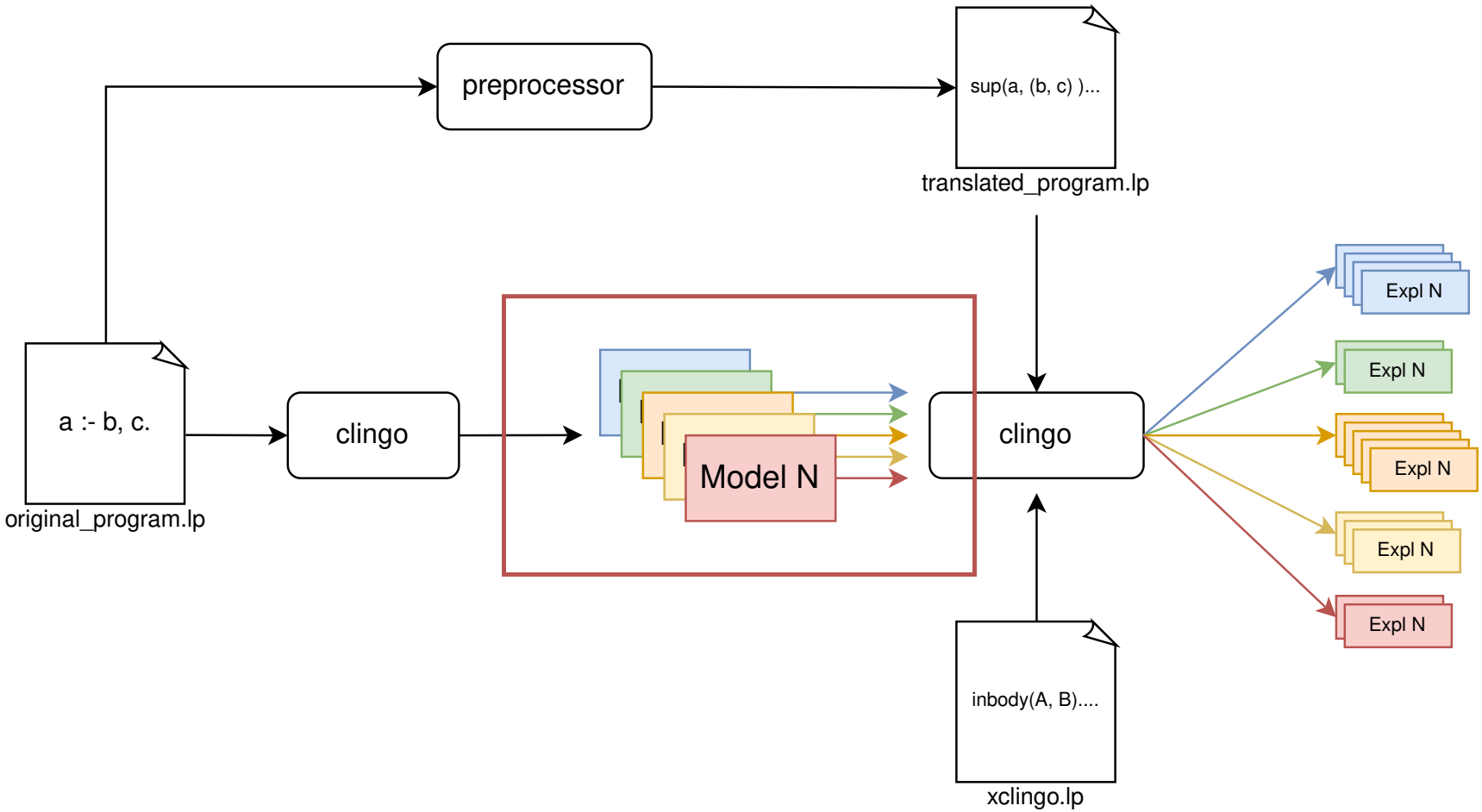
Time for a demo

# Contents

# General view



# General view



# Model Input

- Output from the original program:

```
Answer:  1
person(gabriel) person(clare) alcohol(gabriel,40) alcohol(clare,5) drive(gabriel)
drive(clare) punish(gabriel) resist(gabriel) sentence(clare,innocent)
sentence(gabriel,prison)
```

# Model Input

- Output from the original program:

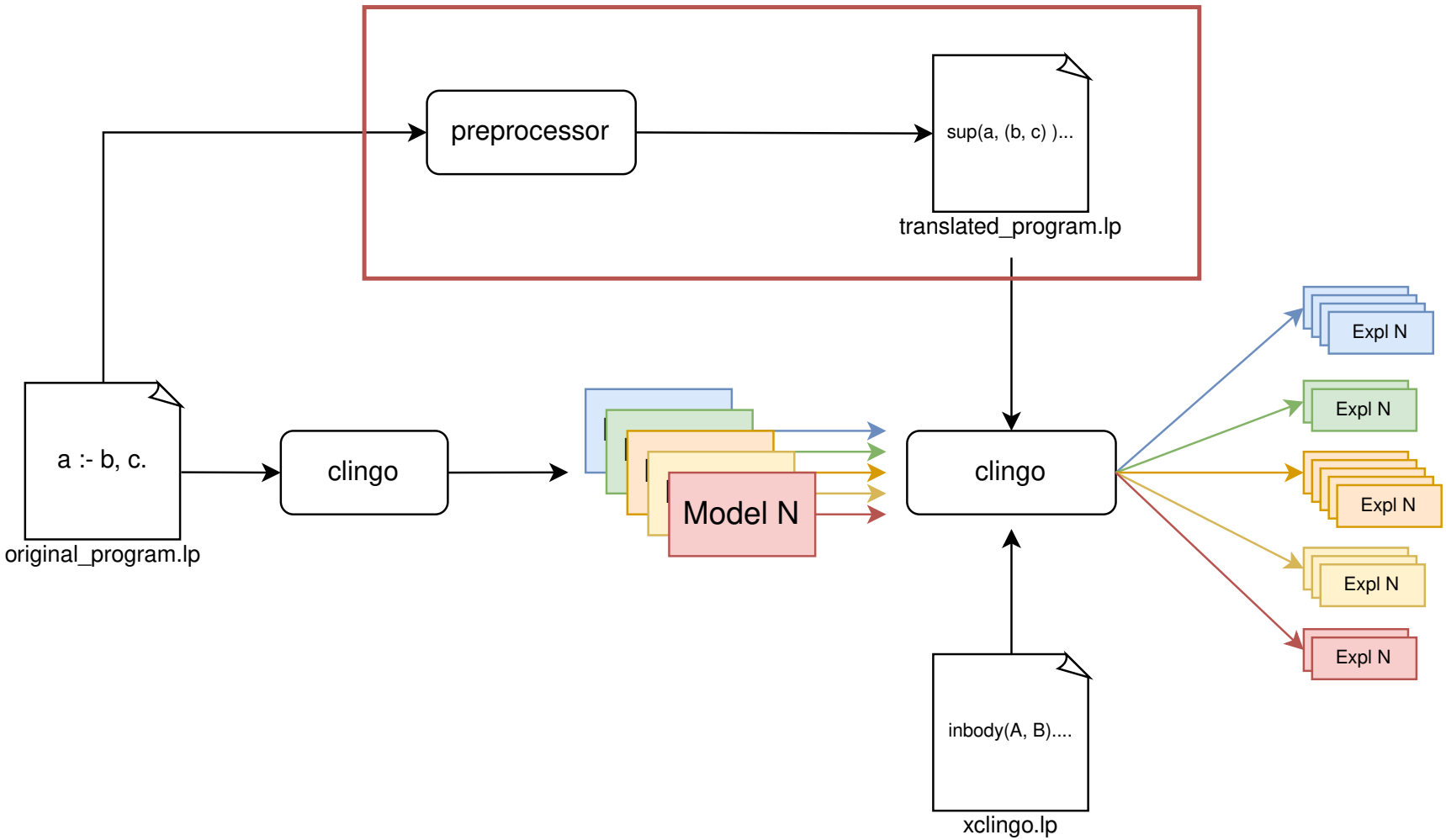
```
Answer:  1
person(gabriel) person(clare) alcohol(gabriel,40) alcohol(clare,5) drive(gabriel)
drive(clare) punish(gabriel) resist(gabriel) sentence(clare,innocent)
sentence(gabriel,prison)
```

- Input for `xclingo`:

```
model(person(gabriel)) model(person(clare)) model(alcohol(gabriel,40))
model(alcohol(clare,5)) model(drive(gabriel)) model(drive(clare))
model(punish(gabriel)) model(resist(gabriel)) model(sentence(clare,innocent))
model(sentence(gabriel,prison))
```

- This is done for every answer set.

# General view



# Translation

- Rules

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
sup(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   model(drive(P));
   model(alcohol(P,A));
   A > 30;
   model(person(P)).
```

# Translation

- Rules to sup/3: rule supports the atom, given the model and the program.

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
sup(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   model(drive(P));
   model(alcohol(P,A));
   A > 30;
   model(person(P)).
```

# Translation

- Rules to sup/3: rule supports the atom, given the model and the program.

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
sup(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   model(drive(P));
   model(alcohol(P,A));
   A > 30;
   model(person(P)).
```

- Facts to sup/3: the atom is always supported, regardless of the model.

```
alcohol(gabriel, 40).
sup(3,alcohol(gabriel,40),()).
```

# Translation

- Rules to sup/3: rule supports the atom, given the model and the program.

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
sup(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   model(drive(P));
   model(alcohol(P,A));
   A > 30;
   model(person(P)).
```

- Facts to sup/3: the atom is always supported, regardless of the model.

```
alcohol(gabriel, 40).
sup(3,alcohol(gabriel,40),()).
```

- %!show_trace

```
%!show_trace sentence(P,S) : alcohol(P,A), not resist(P).
show_trace(sentence(P,S)) :-
   model(sentence(P,S));
   model(alcohol(P,A));
   not model(resist(P)).
```

# Translation

- Rules to sup/3: rule supports the atom, given the model and the program.

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
sup(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   model(drive(P));
   model(alcohol(P,A));
   A > 30;
   model(person(P)).
```

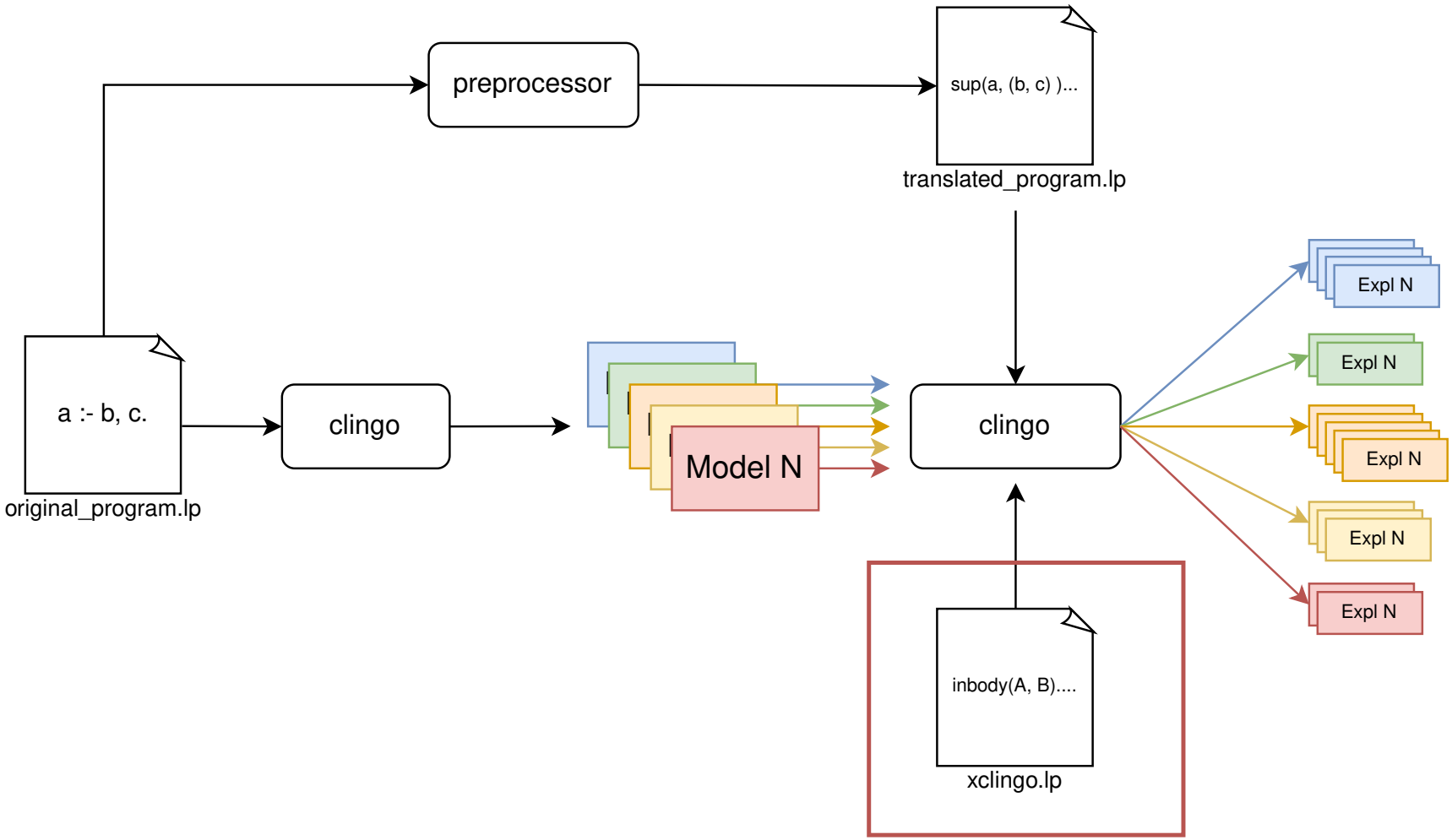
- Facts to sup/3: the atom is always supported, regardless of the model.

```
alcohol(gabriel, 40).
sup(3,alcohol(gabriel,40),()).
```

- %!show_trace (%!mute works in the same way with muted/1. )

```
%!show_trace sentence(P,S) : alcohol(P,A), not resist(P).
show_trace(sentence(P,S)) :-
   model(sentence(P,S));
   model(alcohol(P,A));
   not model(resist(P)).
```

# General view



# fbody/3, f/3 and f_atom/3

- Rules to fbody/3: the body of this rule have been fired.

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
fbody(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   f_atom(drive(P));
   f_atom(alcohol(P,A));
   A > 30;
   f_atom(person(P)).
```

# fbody/3, f/3 and f_atom/3

- Rules to fbody/3: the body of this rule have been fired.

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
fbody(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   f_atom(drive(P));
   f_atom(alcohol(P,A));
   A > 30;
   f_atom(person(P)).
```

- Facts to fbody/3.

```
alcohol(gabriel, 40).
fbody(3,alcohol(gabriel,40),()).
```

# Alternative fbody predicates

```
*
|__gabriel goes to prison
|   |__gabriel drove drunk
|   |   |__gabriel alcohol's level was 40
|   |   |__gabriel was driving

*
|__gabriel goes to prison
|   |__gabriel resisted to authority
```

# fbody/3, f/3 and f_atom/3

- Rules to fbody/3: ~~body of this rule have been fired.~~ the body is true considering the rules that have been selected for the current graph.

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
fbody(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   f_atom(drive(P));
   f_atom(alcohol(P,A));
   A > 30;
   f_atom(person(P)).
```

- Facts to fbody/3.

```
alcohol(gabriel, 40).
fbody(3,alcohol(gabriel,40),()).
```

# fbody/3, f/3 and f_atom/3

- Rules to fbody/3: ~~body of this rule have been fired.~~ the body is true considering the rules that have been selected for the current graph.

```
punish(P) :- drive(P), alcohol(P,A), A>30, person(P).
fbody(7,punish(P),(drive(P),alcohol(P,A),person(P))) :-
   f_atom(drive(P));
   f_atom(alcohol(P,A));
   A > 30;
   f_atom(person(P)).
```

- Facts to fbody/3.

```
alcohol(gabriel, 40).
fbody(3,alcohol(gabriel,40),()).
```

- %!trace and %!trace_rule

```
%!trace_rule {"% resisted to authority", P}
label(Head,@label("% drove drunk",(P,))) :-
   f(7,Head,(drive(P),alcohol(P,A),person(P))).
```

# Contents

# Conclusions and future work

- xclingo: a tool for generating explanations of ASP programs.
  - Text-based, human readable explanations for ASP programs.
  - Meaning of the original program is not modified by annotations.
  - Able to explain why a program is UNSAT.

- Future work:
  - Proper explaining of body aggregates.
    - Now: body aggregates are allowed but not explained.
    - Future: atoms giving support to aggregates will participate as causes for the rules.
  - Performance improvements.
  - Enhancement of the python API.

# The end

Thank you!

```
python -m pip install xclingo
```

https://github.com/bramucas/xclingo2

Warninig: ongoing development!

# Generating Answer Set Justifications with xclingo

Pedro Cabalar, Jorge Fandinno, Brais Muñiz

September 7, 2022