Answer Set Programming

(Partially based on slides from K. Chvalovsky and T. Eiter)

A Problem with Negation as Failure

- Negation in Prolog: “not A” is true if we fail to prove A.

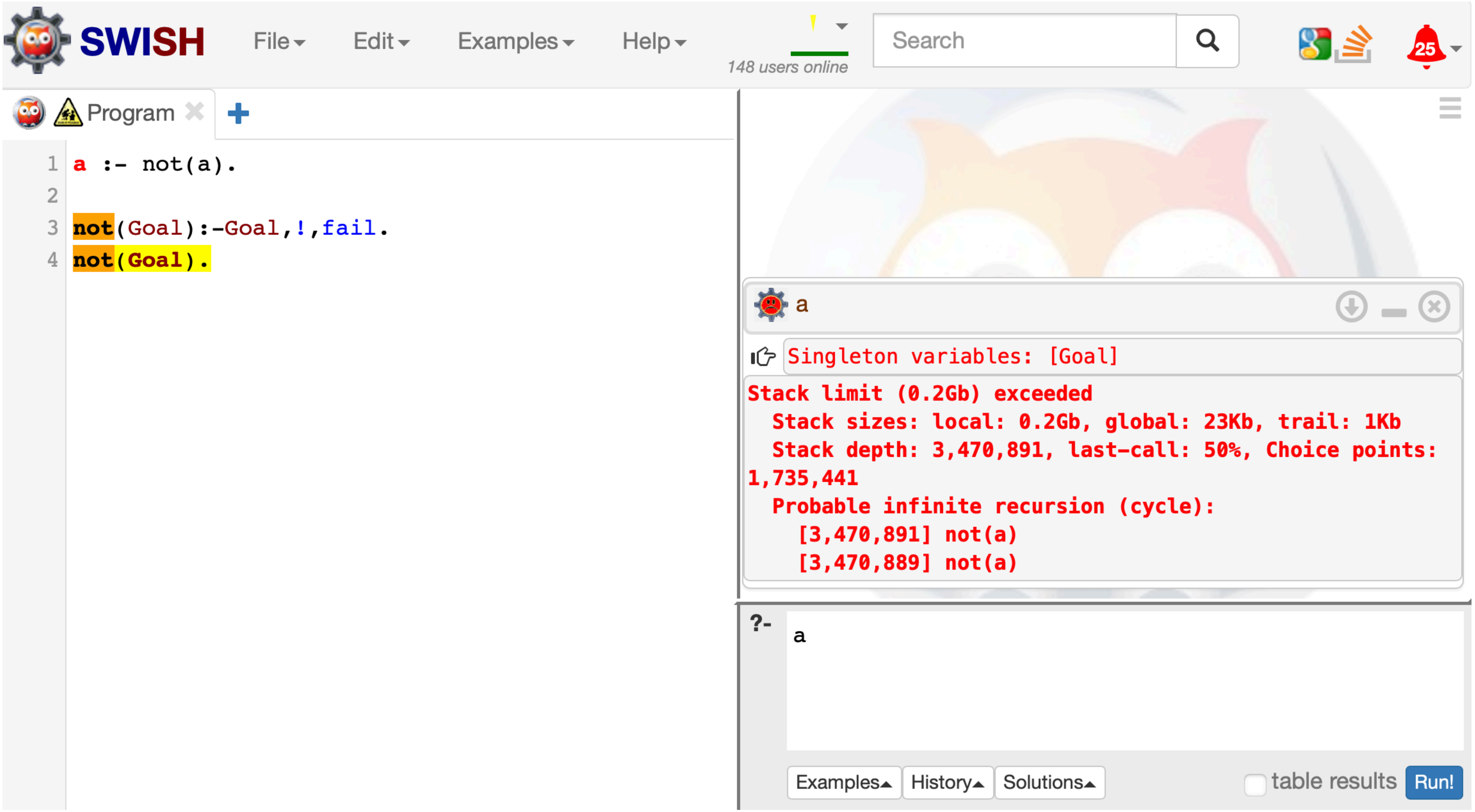
- How would you interpret a rule such as:

```
a :- not a.
```

- If we fail to prove `a` then `a` is true but that means it can be proven so the rule should not have fired but then it a would not be proven…. chicken and egg problem.

- (If the above were a classical logic formula with a classical negation then we would have: $a \Leftarrow \neg a \equiv a \vee a \equiv a$, so its model would be {*a*} but that is not what we want from negation as failure.)

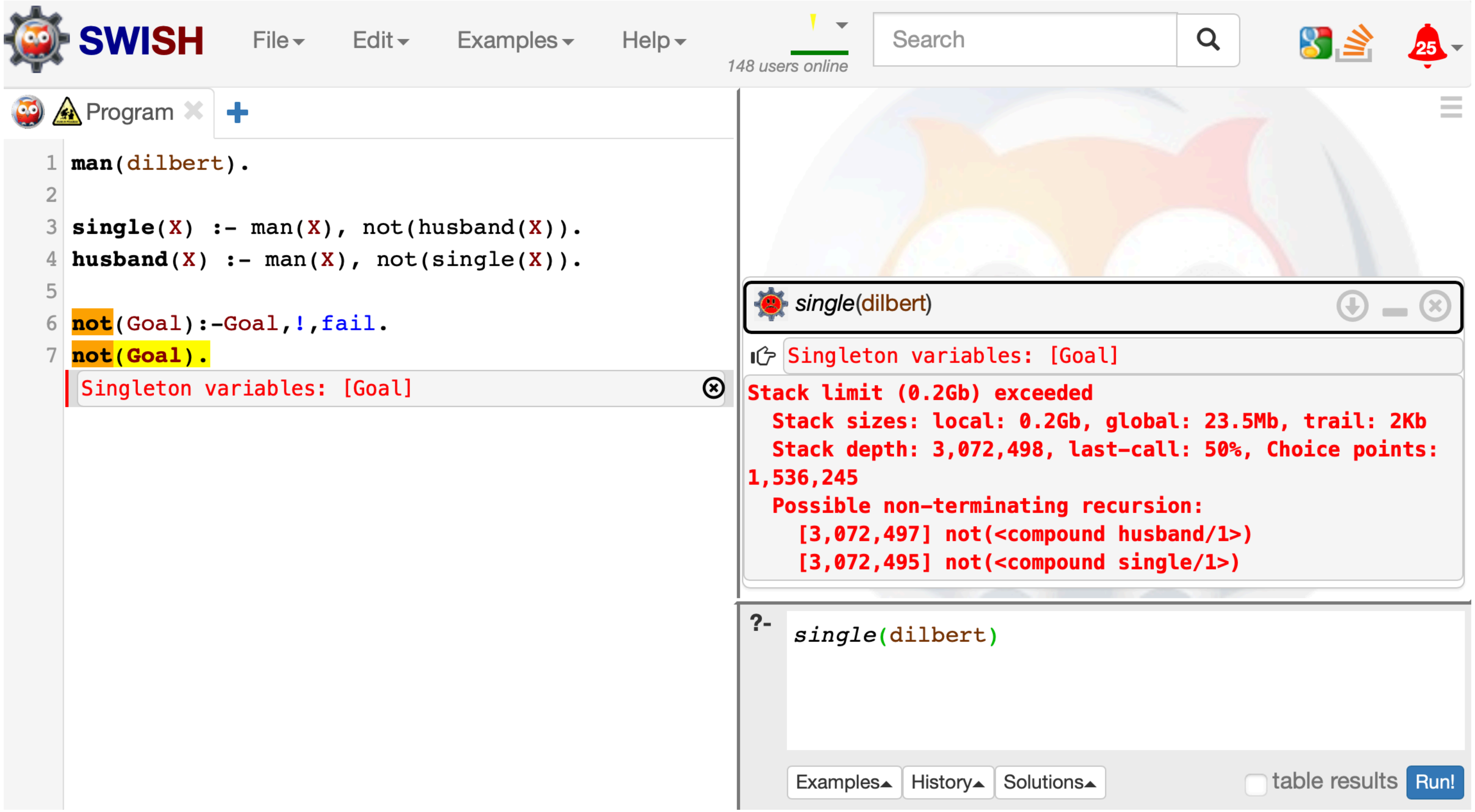
What will Prolog do?



A More Complex Example

```
man(dilbert).
single(X) :- man(X), not husband(X).
husband(X) :- man(X), not single(X).
```

What will Prolog do?



In Classical Logic

If we interpreted the previous program using classical negation, instead of negation as failure, we would get the following FOL sentence:

$$\text{man}(\text{dilbert}) \wedge \forall x : \text{single}(x) \vee \text{married} \vee \neg\text{man}(x).$$

This sentence has two minimal models over the domain:

$$\{\text{man}(\text{dilbert}), \text{single}(\text{dilbert})\} \text{ and } \{\text{man}(\text{dilbert}), \text{married}(\text{dilbert})\}.$$

None of these models is the *least* model. **There is no *least model* in this case.** So the minimum model semantics will not help us.

We need different semantics

- Prolog is a nice programming language but it is not fully declarative: the order of rules matters, the order of literals in rules matters, cut is not declarative at all, negation implemented using cut is tricky from the knowledge-representation perspective…

Stable Model Semantics

- By Gelfond and Lifschitz [1988, 1991].
- Based on the idea of stable models, which agree with minimal model semantics for programs without negation (and also for so-called stratified programs - not too important here).
- *(There are other types of semantics for logic programming, such as well-founded semantics [van Gelder er al, 1991]… but we will not be dealing with them here.)*

Caution!

- **In most of what follows we will focus on ground programs, that is programs without variables, e.g. `a :- b, c.`**

RECAP

Small Recap: Minimal Model Semantics

- **Proposition:** Let $\mathscr{M}$ be the set of all models of a given definite program $\mathscr{P}$. Let us define $\omega_{least} = \bigcap_{\omega \in \mathscr{M}} \omega$. Then ω_{least} is a model of $\mathscr{P}$ (and hence $\omega_{least} \in \mathscr{M}$). We call ω_{least} the least model of ω .

RECAP

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \{h \mid h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P}$ and $b_1, \ldots, b_m \in \omega\}$.

RECAP

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{ h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega \right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\varnothing)))$.

RECAP

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
   1. $\omega_0 = \emptyset$

RECAP

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{ h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega \right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\varnothing)))$.

- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have

  1. $\omega_0 = \varnothing$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.

RECAP

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
  1. $\omega_0 = \emptyset$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.
  3. $\omega_2 = T_P(\omega_1) = \{a, b\}$.

RECAP

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \{h \mid h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
  1. $\omega_0 = \emptyset$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.
  3. $\omega_2 = T_P(\omega_1) = \{a, b\}$.
  4. $\omega_3 = T_P(\omega_2) = \{a, b, c\}$.

RECAP

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
  1. $\omega_0 = \emptyset$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.
  3. $\omega_2 = T_P(\omega_1) = \{a, b\}$.
  4. $\omega_3 = T_P(\omega_2) = \{a, b, c\}$.
  5. $\omega_4 = T_P(\omega_3) = \{a, b, c\} = \omega_3$. We have reached fix-point, ω_3 is the least model of $\mathscr{P}$.

RECAP

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
  1. $\omega_0 = \emptyset$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.
  3. $\omega_2 = T_P(\omega_1) = \{a, b\}$.
  4. $\omega_3 = T_P(\omega_2) = \{a, b, c\}$.
  5. $\omega_4 = T_P(\omega_3) = \{a, b, c\} = \omega_3$. We have reached fix-point, ω_3 is the least model of $\mathscr{P}$.

Back to Stable Model Semantics…

Positive and Normal LPs

- **Positive logic program:** a logic program containing no negations, i.e. all rules have the form:

```
h :- b_1, …, b_m.
```

- **Normal logic program:** a logic program containing rules of the form:

```
h :- b_1, …, b_m, not c_1, …, not c_n.
```

Reduct P^M

- **Reduct of a program P relative to a set of atoms M:** Given a normal logic program P and a set of atoms M, we define the reduct **P^M** as

$$P^M = \{h \text{ :- } b_1, \ldots, b_m \,|\, h \text{ :- } b_1, \ldots. b_m, \text{not}\, c_1, \ldots, \text{not}\, c_n \in P \text{ and } \forall i : c_i \notin M\}.$$

- **That is P^M is obtained from P by:**
  1. removing all rules $h \text{ :- } b_1, \ldots. b_m, \text{not}\, c_1, \ldots, \text{not}\, c_n$ where some of the atoms $c_1, \ldots, c_n$ are contained in M (intuition: such a rule would not "fire" under the context of M),
  2. removing all negative literals from all the other rules.

Stable Models (1)

- A set of atoms *M* is a **stable model** (aka **answer set**) of *P* if *M* is the minimal model of $\mathbf{P^M}$.

- **Example 1:** $P_1 = \{a \leftarrow a.\ b \leftarrow \text{not}\, a.\}$ has one answer set $\{b\}$.

M	P_1^M	$\min_{\subseteq}(P_1^M)$
$\emptyset$	$\{a \leftarrow a.\ b \leftarrow .\}$	$\{b\}$
$\{a\}$	$\{a \leftarrow a.\}$	$\emptyset$
$\{b\}$	$\{a \leftarrow a.\ b \leftarrow .\}$	$\{b\}$
$\{a, b\}$	$\{a \leftarrow a.\}$	$\emptyset$

Stable Models (2)

- A set of atoms *M* is a **stable model** (aka **answer set**) of *P* if *M* is the minimal model of **P^M**.

- **Example 2:** $P_2 = \{a \leftarrow \text{not}\, b.\ b \leftarrow \text{not}\, a.\}$ has two answer sets $\{a\}$ and $\{b\}$.

M	P_2^M	$\min_{\subseteq}(P_2^M)$
$\emptyset$	$\{a \leftarrow .\ b \leftarrow .\}$	$\{a, b\}$
$\{a\}$	$\{a \leftarrow .\}$	$\{a\}$
$\{b\}$	$\{b \leftarrow .\}$	$\{b\}$
$\{a, b\}$	$\emptyset$	$\emptyset$

Stable Models (3)

- A set of atoms *M* is a **stable model** (aka **answer set**) of *P* if *M* is the minimal model of $\mathbf{P^M}$.

- **Example 2:** $P_3 = \{a \leftarrow \text{not}\, a.\}$ has no answer set.

M	P_3^M	$\min_{\subseteq}(P_3^M)$
$\emptyset$	$\{a \leftarrow .\}$	$\{a\}$
$\{a\}$	$\emptyset$	$\emptyset$

Complexity

- **Theorem:** Deciding whether a normal logic program has an answer set is:

  1. NP-complete if the logic program is ground (i.e. has no variables).
     Intuition for membership in NP: Guess an answer set M and check if M is a minimal model of P^M - this can be done in polynomial time using the immediate consequence operator until fixpoint.
     Intuition for hardness (in a moment).

  2. NEXPTIME-complete if the logic program is function-free.
     Intuition for membership in NEXPTIME: Same as above but we first need to ground the program which may lead to an exponential blow-up in its size.

Constraints

- Constraints have the form
  `:- a_1, …, a_m, not c_1, …, not c_n.`

- Such a constraint removes all minimal models that contain all of `a_1, …, a_m` are true and none of `c_1, …, c_n`.

SAT in ASP (1)

- We will now show how to encode an arbitrary SAT problem as an ASP problem (thus also proving NP-hardness).
- **An observation:**

For any ground atom, say `a`, we can introduce a new ground atom `not_a` (we could call it differently, the prefix `not_` has no special meaning in ASP) and add the following two rules + a constraint:

```
a :- not not_a.
not_a :- not a.
:- a, not_a.
```

Then the answer sets will be: `{a}, {not_a}`.

SAT in ASP (2)

- We will now show how to encode an arbitrary SAT problem as an ASP problem (thus also proving NP-hardness).

- **An observation:**

  For any ground atom, say `a`, we can introduce a new ground atom `not_a` (we could call it differently, the prefix `not_` has no special meaning in ASP) and add the following two rules + a constrain:

```
a :- not not_a.
not_a :- not a.
:- a, not_a.
```

  Then the answer sets will be: `{a}, {not_a}`.

SAT in ASP (3)

```
a :- not not_a.
not_a :- not a.
:- a, not_a.
```

Then the answer sets will be: `{a}, {not_a}`.

Why?

M	**P^M**	**min model of P^M**
`{}`	`a.` `not_a.`	`{a, not_a}`
`{a}`	`a.`	`{a}`
`{not_a}`	`not_a.`	`{not_a}`
`{a,not_a}`		`{}`

SAT in ASP (4)

- Now we will encode a SAT problem (the construction can be easily generalized):

- **Example:** $\varphi = (a \vee b) \wedge (\neg a \vee \neg b)$

```
a :- not not_a.
not_a :- not a.
:- a, not_a.
b :- not not_b.
not_b :- not b.
:- b, not_b.
```

`:- not_a, not_b.` $(a \vee b) \equiv \neg(\neg a \wedge \neg b) \equiv \bot \Leftarrow (\neg a \wedge \neg b)$

`:- a, b.` $(\neg a \vee \neg b) \equiv \neg(a \wedge b) \equiv \bot \Leftarrow (a \wedge b)$

SAT in ASP (5)

- **Example:** $\varphi = (a \vee b) \wedge (\neg a \vee \neg b)$

Running clingo

Examples: Harry and Sally

```
a :- not not_a.
not_a :- not a.
:- a, not_a.
b :- not not_b.
not_b :- not b.
:- b, not_b.

:- not_a, not_b.
:- a, b.
```

Configuration: reasoning mode enumerate all ☐ project ☐ statistics

▶ Run!

```
clingo version 5.5.0
Reading from stdin
Solving...
Answer: 1
b not_a
Answer: 2
not_b a
SATISFIABLE

Models       : 2
Calls        : 1
Time         : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Disjunctive Rules

- A **disjunctive logic rule *r*** is a rule of the form:

```
h_1 | h_2 | … | h_k :- b_1, …, b_m, not c_1, …, not c_n.
```

- We define $H(r) = \{$`h_1, h_2, …, h_k`$\}$, $B(r) = \{$`b_1,…,b_m, c_1, …, c_n`$\}$, $B^+(r) = \{$`b_1,…,b_m`$\}$, $B^-(r) = \{$`c_1, …, c_n`$\}$.

- A set of atoms *M* is a model of a disjunctive program *P* if for all rules $r \in P$ it holds: if $B^+(r) \subseteq M$ and $B^-(r) \cap M = \emptyset$ then $H(r) \cap M \neq \emptyset$.

Minimal Models

- Unlike normal logic programs without negation as failure, which have a single minimal (least) model, disjunctive logic programs can have multiple minimal models (even without negation).

- **Example:**

```
c.
a | b :- c.
```

This program has two minimal models: `{a,c}`, `{b,c}`.

Reduct of a Disjunctive Program

- Similar to reduct of a normal program…

- **P^M is obtained from P by:**

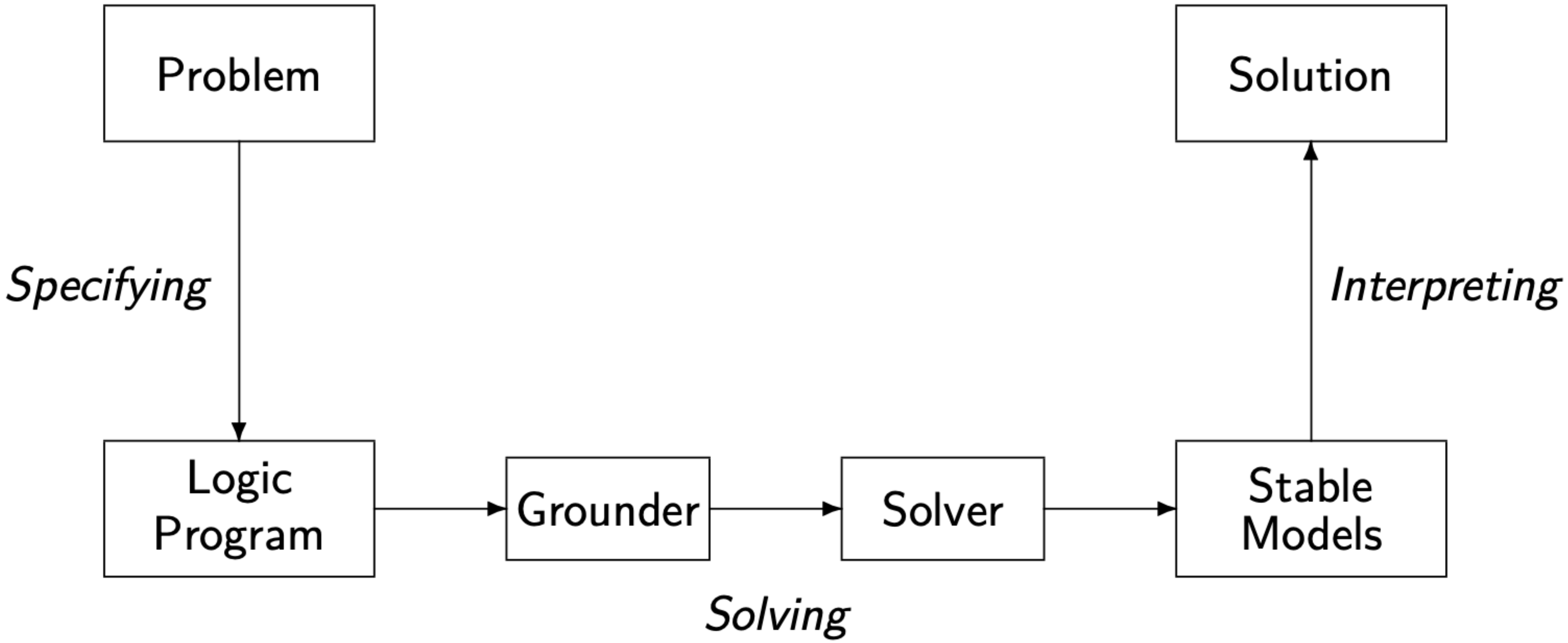
  1. removing all rules $h_1 \,|\, \ldots \,|\, h_k$:- $b_1, \ldots . \, b_m, \text{not}\, c_1, \ldots, \text{not}\, c_n$ where some of the atoms $c_1, \ldots, c_n$ are contained in M (intuition: such a rule would not "fire" under the context of *M*),
  2. removing all negative literals from all the other rules

- *M* is an answer set of a program *P* if *M* is a (non-unique) minimal model of P^M (same as for normal programs).

Complexity

- Deciding whether a disjunctive logic *P* has an answer set is
  - NP^{NP}-complete if *P* is ground.
  - $NEXPTIME^{NP}$-complete if *P* is function-free (but not ground).

Intermezzo: Grounding (1)



source: Gebser et al. 2012

Intermezzo: Grounding (2)

- **Naive grounding:** Use all substitutions.

```
giant(john).
elf(bob).
tall(X) :- giant(X).
small(X) :- elf(X).

  taller(X,Y) :- tall(X),
                 small(Y).
```

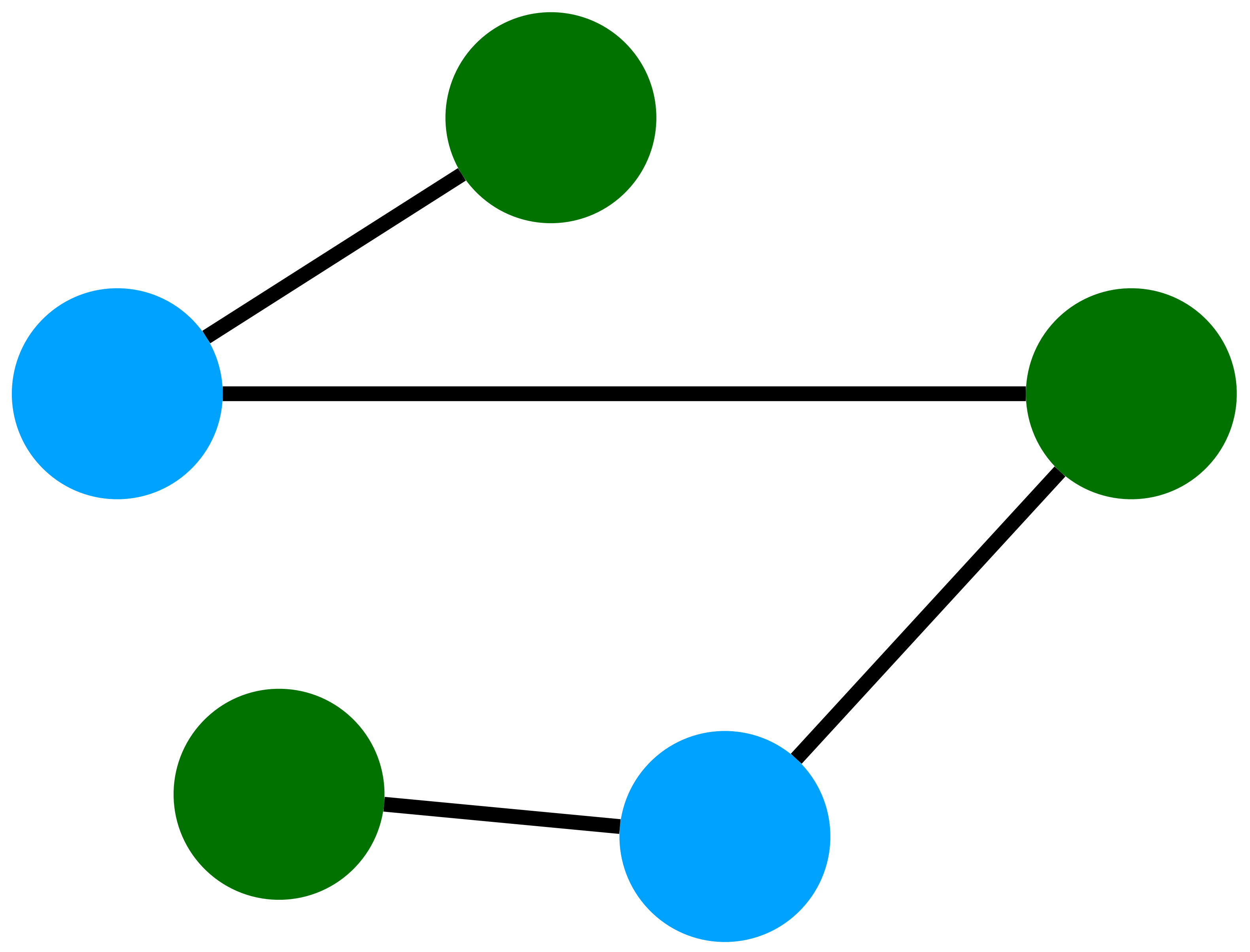
```
giant(john).
elf(bob).
tall(john) :- giant(john).
tall(bob) :- giant(bob).
small(john) :- elf(john).
  small(bob) :- elf(bob).

  taller(john,john) :- tall(john),
                       small(john).
  taller(john,bob) :- tall(john),
                      small(bob).
  ….
```

- There are also more intelligent grounding mechanisms (see, e.g., the GrinGo grounder).

Graph Coloring using ASP (1)

- Recall that a coloring of a given graph is an assignment of colors to its vertices in such a way that no two vertices connected by an edge have the same color. We are typically interested in coloring a graph with a given set of colors.



Graph Coloring using ASP (2)

- Recall that a coloring of a given graph is an assignment of colors to its vertices in such a way that no two vertices connected by an edge have the same color. We are typically interested in coloring a graph with a given set of colors.

```
node(1). node(2). node(3). node(4).
edge(1,2). edge(1,4). edge(2,3). edge(3,4).
edge(X,Y) :- edge(Y,X).

red(X) | blue(X) :- node(X).
:- red(X), blue(X).
:- red(X), edge(X,Y), red(Y).
:- blue(X), edge(X,Y), blue(Y).
```

Graph Coloring using ASP (3)

```
node(1). node(2). node(3). node(4).
edge(1,2). edge(1,4). edge(2,3). edge(3,4).
edge(X,Y) :- edge(Y,X).

red(X) | blue(X) :- node(X).

:- red(X), blue(X).
:- red(X), edge(X,Y), red(Y).
:- blue(X), edge(X,Y), blue(Y).
```

Configuration: reasoning mode enumerate all ☐ project ☐ statistics

▶ Run!

```
edge(4,3) edge(3,2) edge(4,1) edge(2,1) node(1) node(2) node(3) node(4) blue(1) red(2) blue(3) red(4)

edge(4,3) edge(3,2) edge(4,1) edge(2,1) node(1) node(2) node(3) node(4) red(1) blue(2) red(3) blue(4)

t Model: 0.00s Unsat: 0.00s)
```

Graph Coloring using ASP (4)

- Recall that a coloring of a given graph is an assignment of colors to its vertices in such a way that no two vertices connected by an edge have the same color. We are typically interested in coloring a graph with a given set of colors.

```
node(1). node(2). node(3). node(4).
edge(1,2). edge(1,4). edge(2,3). edge(3,4).
edge(X,Y) :- edge(Y,X).

red(X) | blue(X) :- node(X).
:- red(X), blue(X).  ← this can be removed… do you see why?
:- red(X), edge(X,Y), red(Y).
:- blue(X), edge(X,Y), blue(Y).
```

Note

- Graph coloring can be also expressed and solved using normal logic programs (not disjunctive) since the ASP problem with normal programs is NP-complete.

- Disjunctive programs can express more complex problems from the class NP^{NP}.

Modelling

Choice Rules

- **A choice rule:**

  {h_1;…;h_k} :- b_1, …, b_m, not c_1, …, not c_n.

  The meaning is that any subset of {h_1,…,h_k} can be added to the answer set if the body is satisfied.

- **Example:**

```
a.
{b} :- a.
```

  This program has two answer sets: `{a}, {a,b}.`

Choice Rules (Meaning)

```
{h_1;…;h_k} :- b_1, …, b_n, not c_1, …, not c_m.
```

A choice rule r can be replaced by normal rules

$$h' \leftarrow b_1, \ldots, b_n, \text{not}\, c_1, \ldots, \text{not}\, c_m.$$

$$h_i \leftarrow h', \text{not}\, \overline{h_i}.$$

$$\overline{h_i} \leftarrow \text{not}\, h_i.$$

for $1 \leq i \leq k$ if we introduce new atoms $h', \overline{h_1}, \ldots, \overline{h_k}$. The resulting program has the same stable models if we ignore the newly introduced atoms.

Cardinality Constraints

- Cardinality constraints have the form:

```
l{b_1; …, b_n; not c_1; …; not c_m}u
```

The meaning is that at least l and at most u atoms from {b_1; …, b_n; not c_1; …; not c_m} are true in a stable model. If l or u is missing, respectively, then there is no lower or upper bound, respectively. Cardinality constraints can be used in heads and bodies.

There are also other modelling constructs that we have not covered, such as weight constraints and aggregate atoms, weak constraints…

Consequence Relations for Answer Sets

Brave and Cautious Consequence

- An atom `a` is a **brave consequence** of P if $M \models a$ for some answer set M of P.

- An atom `a` is a **cautious consequence** of P if $M \models a$ for all answer sets M of P.

Non-Monotonicity

- Both brave and cautious consequence relations are non-monotonic.
- In classical logic when we add more rules to a theory T, everything that could have been derived from T can also be derived from the new larger theory - this is called **monotonicity.**
- Consider the following program P:

```
bird(tweety).
flies(X) :- bird(X), not penguin(X).
```

Then `flies(tweety)` is both a brave and cautious consequence of P.

However, if we add `penguin(tweety)` to P then `flies(tweety)` will no longer be a consequence of P (neither brave not cautious).