# Answer Set Programming

Włodzimierz Drabent

IPI PAN, seminarium zespołu, styczeń 2021
Wersja 0.5, 21 stycznia 2021

# Światowy Dzień Logiki

14 stycznia
Światowy Dzień Logiki / World Logic Day
(UNESCO, International Council for Philosophy and Human Sciences)

Data śmierci Kurta Gëdla i urodzin Alfreda Tarskiego

`https://en.unesco.org/commemorations/worldlogicday`
`https://en.wikipedia.org/wiki/World_Logic_Day`

UNESCO:
The ability to think is one of the most defining features of humankind. ...

Vienna World Logic Day Lecture, Prof. Georg Gottlob
on the future of logic in the world shaped by AI, 2021-01-14 17:00 CET
`https://logicday.vcla.at/vienna-logic-day-lecture/`

# Światowy Dzień Logiki

We overlook it at our own peril, as Logic is even more inexorable than taxes and death.

Valeria de Paiva

There's nothing you can do that can't be done
Nothing you can sing that can't be sung
Nothing you can say, but you can learn how to play the game
It's easy
All you need is logic
All you need is logic
All you need is logic, logic
Logic is all you need!

Arnon Avron

Wielu najpotężniejszych czarodziejów nie ma pojęcia o logice.
Dla nich to pułapka bez wyjścia.

Harry Potter, tom 1

# Plan

- Programowanie w logice – wprowadzenie
- Potrzeba negacji. CWA, NAF
- Programy z negacją

Tytuł: ASP – brak polskiego terminu(?)

Przepraszam za dwujęzyczność slajdów

# Programowanie w logice

inne slajdy

**Prz.:**

$$P: \quad \begin{array}{l} high\_salary \leftarrow employed, educated. \\ educated \leftarrow high\_salary. \\ employed \leftarrow motivated. \\ motivated. \end{array}$$

Zbiór logicznych konsekwencji (bez zmiennych),
czyli najmniejszy (względem $\subseteq$) model Herbranda programu $P$

$$\mathbf{M}_P = \{\, motivated,\ employed \,\}$$

# The need for negation (or for non-monotonic reasoning)

No negative consequences of definite clause programs.

Ex.: $\neg salary(a, 99)$ does not follow from program

$$P_s = \{\, salary(a, 3000). \quad salary(b, 4000). \quad salary(c, 5000). \,\}$$

But we (sometimes) would like to derive it.

So we derive $\neg A$ from $P$ when $P \not\models A$ and $A$ is ground.

This is called Closed World Assumption (CWA).

Założenie o zamkniętym świecie

# The need for negation (or for non-monotonic reasoning)

No negative consequences of definite clause programs.

Ex.: $\neg salary(a, 99)$ does not follow from program

$$P_s = \{\, salary(a, 3000). \quad salary(b, 4000). \quad salary(c, 5000). \,\}$$

But we (sometimes) would like to derive it.

So we derive $\neg A$ from $P$ when $P \not\models A$ and $A$ is ground.

This is called Closed World Assumption (CWA).

Założenie o zamkniętym świecie

# Closed World Assumption (CWA)

**Df.** (CWA): $P \vdash_{\mathrm{CWA}} \neg A$ when $P \not\models A$ and $A$ is ground.
(derive $\neg A$ from $P$)

**Fact** (about definite clause LP)

☞ Logic programs can compute whatever is computable.
(Any r.e. set of terms can be defined by a definite clause program.)

☹ CWA may be not recursively enumerable (i.e. not computable).
As $\{ A \mid P \models A,\ A \text{ is ground} \}$ is r.e. but may be not recursive.

# Negation as Finite Failure (NAF, NAFF ?)

What we usually compute in Prolog is Negation as Finite Failure:

**Df.:** $P \vdash_{\mathrm{NAF}} \neg A$ when an SLD-tree for $A$ is failed (better: finitely failed)

$A$ - atom, maybe nonground

i.e. is finite and without answers

The difference CWA – NAF sometimes not understood.

**Ex.:** $P = \{\, p \leftarrow p \,\}$ $P \vdash_{\mathrm{CWA}} \neg p$ $P \nvdash_{\mathrm{NAF}} \neg p$

NAF compatible with CWA:

If $P \vdash_{\mathrm{NAF}} \neg A$ then $P \vdash_{\mathrm{CWA}} \neg A$

(If $A$ fails then $P \not\models A$)

NAF can be defined logically, by program completion

Roughly: $\leftrightarrow$ instead of $\leftarrow$

# Negation as Finite Failure (NAF, NAFF ?)

What we usually compute in Prolog is Negation as Finite Failure:

**Df.:** $P \vdash_{\mathrm{NAF}} \neg A$ when an SLD-tree for $A$ is failed (better: finitely failed)

$A$ - atom, maybe nonground

i.e. is finite and without answers

The difference CWA – NAF sometimes not understood.

**Ex.:** $P = \{ p \leftarrow p \}$ $P \vdash_{\mathrm{CWA}} \neg p$ $P \nvdash_{\mathrm{NAF}} \neg p$

NAF compatible with CWA:

If $P \vdash_{\mathrm{NAF}} \neg A$ then $P \vdash_{\mathrm{CWA}} \neg A$

(If $A$ fails then $P \not\models A$)

NAF can be defined logically, by program completion

Roughly: $\leftrightarrow$ instead of $\leftarrow$

# Negation as Finite Failure (NAF, NAFF ?)

What we usually compute in Prolog is Negation as Finite Failure:

**Df.:** $P \vdash_{\mathrm{NAF}} \neg A$ when an SLD-tree for $A$ is failed (better: finitely failed)

$A$ - atom, maybe nonground

i.e. is finite and without answers

The difference CWA – NAF sometimes not understood.

**Ex.:** $P = \{\, p \leftarrow p \,\}$ $P \vdash_{\mathrm{CWA}} \neg p$ $P \nvdash_{\mathrm{NAF}} \neg p$

NAF compatible with CWA:

If $P \vdash_{\mathrm{NAF}} \neg A$ then $P \vdash_{\mathrm{CWA}} \neg A$

(If $A$ fails then $P \not\models A$)

NAF can be defined logically, by program completion

Roughly: $\leftrightarrow$ instead of $\leftarrow$

# Negation as Finite Failure (NAF, NAFF ?)

What we usually compute in Prolog is Negation as Finite Failure:

**Df.:** $P \vdash_{\mathrm{NAF}} \neg A$ when an SLD-tree for $A$ is failed (better: finitely failed)

$A$ - atom, maybe nonground

i.e. is finite and without answers

The difference CWA – NAF sometimes not understood.

**Ex.:** $P = \{\, p \leftarrow p \,\}$ $\quad P \vdash_{\mathrm{CWA}} \neg p \quad$ $P \nvdash_{\mathrm{NAF}} \neg p$

NAF compatible with CWA:

If $P \vdash_{\mathrm{NAF}} \neg A$ then $P \vdash_{\mathrm{CWA}} \neg A$

(If $A$ fails then $P \not\models A$)

NAF can be defined logically, by program completion

Roughly: $\leftrightarrow$ instead of $\leftarrow$

# Programy z negacją

Oczywiście chcielibyśmy użyć negacji w programach.

**Prz.:** % $listd(X)$ – $X$ jest listą parami różnych elementów

$listd([\,]).$
$listd([H|T]) \leftarrow listd(T), \neg\, member(H, T).$

$member(X, [X|L]).$
$member(X, [Y|L]) \leftarrow member(X, L).$

**Prz.:** % $subset([x_1, \ldots, x_n], [y_1, \ldots, y_m])$ – $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_m\}$

$P$: $subset(L, M) \leftarrow \neg\, notsubset(L, M).$
$notsubset(L, M) \leftarrow member(X, L), \neg\, member(X, M).$

# Programy z negacją

Oczywiście chcielibyśmy użyć negacji w programach.

**Prz.:** % $listd(X)$ – $X$ jest listą parami różnych elementów

$$listd([\,]).$$
$$listd([H|T]) \leftarrow listd(T), \neg\, member(H,T).$$

$$member(X,[X|L]).$$
$$member(X,[Y|L]) \leftarrow member(X,L).$$

**Prz.:** % $subset([x_1,\ldots,x_n],[y_1,\ldots,y_m])$ – $\{x_1,\ldots,x_n\} \subseteq \{y_1,\ldots,y_m\}$

$P$: $subset(L,M) \leftarrow \neg\, notsubset(L,M).$
$notsubset(L,M) \leftarrow member(X,L), \neg\, member(X,M).$

# Programy z negacją – składnia

**Df.:**

Literal – an atom (positive literal) or a negated atom (negative literal).

Clause (general clause)

$$H \leftarrow L_1, \ldots, L_n$$

where $n \geqslant 0$, $H$ – atom, $L_1, \ldots, L_n$ – literals, (comma stands for $\wedge$).

Program (general program, normal program) – a set of clauses.

Query (general query) – $L_1, \ldots, L_n$.

Note: No negative literal is a logical consequence of a general program.

# Programy z negacją – kłopoty z semantyką

Ale jakie ma być właściwie znaczenie takich programów ?

Prolog dostarcza naiwnej implementacji rozszerzenia NAF,
która jest często bezsensowna.

Sprawy nieoczywiste i trudne.

Np. błędna główna definicja w 1. wyd. głównego podręcznika 1987 r.

Temat intensywnych badań, głównie przełom lat '80 i '90.

Dopracowano się "właściwych" rozszerzeń CWA i NAF.

# Uogólnienie CWA na programy warstwowe

oczywiste, dość łatwe

Trudności, gdy mamy “rekursję przez negację”

# Semantyka programów z negacją

WFS (well-founded semantics)

general programs

$\uparrow$

CWA $\longrightarrow$ standard model

definite programs — stratified programs

$\downarrow$

stable model semantics

general programs

$\downarrow$

ASP (answer set programming)

disjunctive programs
(with rules $H_1 \vee \cdots \vee H_m \leftarrow B_1, \ldots, B_n$)

NAF

definite programs

$\downarrow$

3v completion sem.,

(SLDNF-resolution)

general programs

# Stable Model Semantics (s.m.s.)

S.m.s. given by 2-valued stable Herbrand models.

Also called answer sets

A program may have 0, 1, or more stable models.

Pragmatics: Problem solutions = stable models.

Other branches of LP – problems solutions = program answers

# Stable Model Semantics (s.m.s.)

Main idea:

Guess a (2-valued, Herbrand) interpretation $M$.
Use $M$ to evaluate negative literals of $P \rightsquigarrow P^M$ (reduct).
$P^M$ – positive program.

(the least Herbrand model of $P^M$)
If the ground atomic consequences of $P^M$ are $M$

$$\underbrace{\{ A \in \mathbf{B}_{\mathcal{A}} \mid P^M \models A \}}_{\mathbf{M}_{P^M}} = M$$

then $M$ is a stable model of $P$.

Details: $P^M$ is $ground(P)$ without

- each rule with some $\neg A$, s.t. $M \models A$
- each $\neg A$, s.t. $M \models \neg A$

# Stable Model Semantics (s.m.s.)

Main idea:

Guess a (2-valued, Herbrand) interpretation $M$.
Use $M$ to evaluate negative literals of $P \rightsquigarrow P^M$ (reduct).
$P^M$ – positive program.

(the least Herbrand model of $P^M$)
If the ground atomic consequences of $P^M$ are $M$

$$\underbrace{\{ A \in \mathbf{B}_{\mathcal{A}} \mid P^M \models A \}}_{\mathbf{M}_{P^M}} = M$$

then $M$ is a stable model of $P$.

Details: $P^M$ is $ground(P)$ without

- each rule with some $\neg A$, s.t. $M \models A$
- each $\neg A$, s.t. $M \models \neg A$

# Stable Model Semantics

Ex.: $P_1 = \{ p \leftarrow \neg q \}$ – one stable model $\{ p \}$

$P_1^{\{p\}} = \{ p \leftarrow \}$ $\quad\quad$ $\{ A \in \mathbf{B}_\mathcal{A} \mid P_1^{\{p\}} \models A \} = \{p\}$

$P_2 = \{ p \leftarrow \neg p \}$ – no stable models

$P_3 = \{ p \leftarrow \neg q;\ q \leftarrow \neg p \}$ – two stable models $\{p\}$ and $\{q\}$

$P_3^{\{p\}} = \{ p \leftarrow \}$ $\quad\quad$ $P_3^{\{q\}} = \{ q \leftarrow \}$

$P_4 = \{ p \leftarrow q;\ q \leftarrow \neg p \}$ – no stable models

In s.m.s. we are usually not interested in answers/consequences, but in whole stable models.

# Stable Model Semantics

Ex.: $P_1 = \{ p \leftarrow \neg q \}$ – one stable model $\{ p \}$

$P_1^{\{p\}} = \{ p \leftarrow \}$ $\quad\quad$ $\{ A \in \mathbf{B}_\mathcal{A} \mid P_1^{\{p\}} \models A \} = \{p\}$

$P_2 = \{ p \leftarrow \neg p \}$ – no stable models

$P_3 = \{ p \leftarrow \neg q;\ q \leftarrow \neg p \}$ – two stable models $\{p\}$ and $\{q\}$

$P_3^{\{p\}} = \{ p \leftarrow \}$ $\quad\quad$ $P_3^{\{q\}} = \{ q \leftarrow \}$

$P_4 = \{ p \leftarrow q;\ q \leftarrow \neg p \}$ – no stable models

In s.m.s. we are usually not interested in answers/consequences, but in whole stable models.

# Example, s.m.s., subsets

Notation: $p^M$ – predicate $p$ in interpretation $M$

A program fragment defining $p^M$ to be a subset of $q^M$ (and $np^M$ to be $q^M \setminus p^M$):

$$p(X) \leftarrow q(X), \neg np(X). \qquad np(X) \leftarrow q(X), \neg p(X).$$

$q^M$ defined by the rest of the program, e.g. by facts $q(c_1).\ \ldots\ q(c_n).$

A stable model for each subset of $q^M$. $2^n$ stable models.

Note: Without the clause for $np$ we get $p^M = q^M$.

# Example, s.m.s., subsets (2)

Notes:

The same task in "standard" logic programming:
most likely by defining a predicate $subs(Subset, Set)$,
with a suitable representation of sets (e.g. by lists).

FOL (first order logic) description of $p^M \subseteq q^M$ is $q(X) \leftarrow p(X)$.
Does not work as a logic program.

# Ograniczenia pragmatyczne

1.
Wymaganie – wykluczamy z języka symbole funkcyjne
(bo potrzebujemy skończonego zbioru obiektów)

Dopuszczalne tylko stałe. ☜

Nie tylko definicje, ale i implementacje operują na instancjach bez zmiennych klauzul programu itp.

Grounding

$$P \quad \rightsquigarrow \quad ground(P)$$

2.
Reguły winne być "safe" – każda zmienna występuje w niezanegowanym literale po prawej stronie.

# Ograniczenia pragmatyczne

1.
Wymaganie – wykluczamy z języka symbole funkcyjne
(bo potrzebujemy skończonego zbioru obiektów)

Dopuszczalne tylko stałe. ☜

Nie tylko definicje, ale i implementacje operują na instancjach bez zmiennych klauzul programu itp.

Grounding

$$P \quad \rightsquigarrow \quad ground(P)$$

2.
Reguły winne być "safe" – każda zmienna występuje w niezanegowanym literale po prawej stronie.

# Example, s.m.s., expressing integrity constraints

To state that some of $L_1, \ldots, L_n$ must be false,
take a new symbol $f$, use rule:

$$f \leftarrow \neg f, L_1, \ldots, L_n$$

$f$ false in each stable model (as $f$ true $\rightsquigarrow$ contradiction)

so in each stable model some $L_i$ must be false.

Abbreviated as

$$\leftarrow L_1, \ldots, L_n \qquad (\text{or} \quad :\!- L_1, \ldots, L_n)$$

# Example, s.m.s., expressing integrity constraints

To state that some of $L_1, \ldots, L_n$ must be false,
take a new symbol $f$, use rule:

$$f \leftarrow \neg f, L_1, \ldots, L_n$$

$f$ false in each stable model (as $f$ true $\rightsquigarrow$ contradiction)

so in each stable model some $L_i$ must be false.

Abbreviated as

$$\leftarrow L_1, \ldots, L_n \qquad \text{(or } :- L_1, \ldots, L_n \text{)}$$

# Example, s.m.s., integrity constraints

3-colouring of a graph, described by predicates $node/1$, $edge/2$.
(From Eiter *et al.* "ASP: A primer")

% $b(X)$ – node $X$ coloured blue, $r(X)$ – ... red, $g(X)$ – ... green.

$$b(X) \leftarrow node(X), \neg r(X), \neg g(X).$$
$$r(X) \leftarrow node(X), \neg b(X), \neg g(X).$$
$$g(X) \leftarrow node(X), \neg r(X), \neg b(X).$$

Each node gets exactly one colour

$$f \leftarrow \neg f, b(X), b(Y), edge(X,Y).$$
$$f \leftarrow \neg f, r(X), r(Y), edge(X,Y).$$
$$f \leftarrow \neg f, g(X), g(Y), edge(X,Y).$$

No neighbours are blue
... red
... green

# Przykłady

## Cykle Hamiltona

`hamilton.dlv`

Dane (instancja problemu) i logika (opis rozwiązania)
– oddzielne części programu

## Największy wspólny podzielnik

`gcd.dlv`

Użycie podwójnej negacji $\neg\neg$

# Krótko o implementacji

1. Przekształcenie do programu bez zmiennych (*grounding*)

$$P \rightsquigarrow grndP$$

2. Znajdowanie modelu dla programów j.w.
   Dwa podejścia

2a. Bezpośrednio. Podobnie jak DPLL lub CDCL dla SAT.

2b. Tłumaczenie do SAT

$$grndP \rightsquigarrow comp(grndP) \cup loop\_formulas(grndP)$$

$comp(P)$ – dopełnienie (*completion*) programu

# Dopełnienie programu

Idea: zamienić $\leftarrow$ na $\leftrightarrow$

Np.

| $P$ | $comp(P)$ |
|---|---|
| $p \leftarrow q$ | $p \leftrightarrow q \vee (r \wedge s)$ |
| $p \leftarrow r, s$ | $r \leftrightarrow \ldots$ |
| $r \leftarrow \ldots$ | $\neg s$ if no rule $s \leftarrow \ldots$ in $P$ |

## Loop formulas

by wykluczyć modele, które nie są stabilne

Np. $p \leftarrow p$ $p \leftrightarrow p$ loop formulas: $\neg p$

$p \leftarrow \neg q$ $p \leftrightarrow \neg q$
$q \leftarrow \neg p$ $q \leftrightarrow \neg p$
loop formulas: $\emptyset$

# Dopełnienie programu

Idea: zamienić $\leftarrow$ na $\leftrightarrow$

Np.

| $P$ | $comp(P)$ |
|---|---|
| $p \leftarrow q$ | $p \leftrightarrow q \vee (r \wedge s)$ |
| $p \leftarrow r, s$ | $r \leftrightarrow \ldots$ |
| $r \leftarrow \ldots$ | $\neg s$ if no rule $s \leftarrow \ldots$ in $P$ |

# Loop formulas

by wykluczyć modele, które nie są stabilne

Np. $p \leftarrow p$ $\quad$ $p \leftrightarrow p$ $\quad$ loop formulas: $\neg p$

$p \leftarrow \neg q$ $\quad$ $p \leftrightarrow \neg q$

$q \leftarrow \neg p$ $\quad$ $q \leftrightarrow \neg p$ $\quad$ loop formulas: $\emptyset$

# Mocna negacja

Rozważana negacja $\neg$ (w programach $not$) zw. z CWA (domyślna negacja, *default negation*)

$$\ldots \vdash \neg p \quad \text{tylko wtedy gdy} \quad \ldots \nvdash p.$$

Czasami potrzebujemy określić jawnie,
kiedy $nie\, p$ ma zachodzić

Np. predykaty $p$ i $np$ w przykładzie z podzbiorami

# Mocna negacja

Aby uniknąć "ręcznego" tworzenia nowych symboli
specjalna konstrukcja wbudowana w jęz. programowania:
dla każdego predykatu $p$ nowy symbol $-p$

i reguła $\leftarrow -p(\vec{X}),\ p(\vec{X})$

Można powiedzieć, że (z punktu widzenia silnej negacji $-$)
mamy logikę 3-wartościową

$$\left.\begin{array}{ll} A \text{ prawdziwe w } I & A \in I \\ A \text{ fałszywe w } I & -A \in I \\ A \ \mathbf{u} \text{ w } I & A \notin I,\ -A \notin I \end{array}\right\} \neg A \text{ prawdziwe w } I$$

(Pewnie lepiej stosować $not$ zamiast $\neg$, ale...)

# Mocna negacja, przykład

Porównajmy

$$walk \leftarrow at(A, L),\ crossing(L),\ \neg\, train\_approaches(L).$$

i

$$walk \leftarrow at(A, L),\ crossing(L),\ -train\_approaches(L).$$

# Reguły dysjunktywne

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \neg\, c_1, \ldots, \neg\, c_n, \qquad (1)$$

Reguła (1) bez zmiennych jest spełniona w modelu Herbranda $I$, gdy

$$\{b_1, \ldots, b_m\} \subseteq I \quad \text{i} \quad I \cap \{c_1, \ldots, c_n\} = \emptyset$$
$$\text{implikuje} \quad I \cap \{a_1, \ldots, a_k\} \neq \emptyset$$

Redukt $P^I$ zdefiniowany jak poprzednio

Uogólnienie modelu stabilnego – answer set

Df.: $M$ jest zbiorem odpowiedzi (AS, modelem stabilnym?) dla $P$, gdy $M$ jest $\subseteq$-minimalnym modelem $P^M$

ASP (Answer Set Programming) – programy
z reg. dysjunktywnymi, mocną negacją i modelami j.w.

# Reguły dysjunktywne, przykłady, programy bez negacji

$broken(left\_hand, tom) \vee broken(right\_hand, tom) \leftarrow$

Dwa modele minimalne (=stabilne)

---

$a \vee b \leftarrow$
$a \vee c \leftarrow$

Modele minimalne $\{a\}$ i $\{b, c\}$

---

$a \vee b \leftarrow$
$a \leftarrow b$

Jeden model minimalny $\{a\}$

---

$a \vee b \leftarrow$
$b \vee c \leftarrow$
$a \vee c \leftarrow$

Trzy modele minimalne $\{a, b\}$, $\{a, c\}$, $\{b, c\}$

# Reguły dysjunktywne, przykłady, programy bez negacji 2

$$male(X) \vee female(X) \leftarrow person(X)$$

---

Podzbiór zbioru $q$ $\quad p(X) \vee -p(X) \leftarrow q(X).$

zamiast

$$p(X) \leftarrow q(X), \neg -p(X). \qquad -p(X) \leftarrow q(X), \neg p(X).$$

---

$$b(X) \vee r(X) \vee g(X) \leftarrow node(X).$$

zamiast

$$b(X) \leftarrow node(X), \neg r(X), \neg g(X).$$
$$r(X) \leftarrow node(X), \neg b(X), \neg g(X).$$
$$g(X) \leftarrow node(X), \neg r(X), \neg b(X).$$

# Reguły dysjunktywne, przykłady, programy bez negacji 2

$$male(X) \vee female(X) \leftarrow person(X)$$

---

Podzbiór zbioru $q$ $\quad p(X) \vee -p(X) \leftarrow q(X).$

zamiast

$$p(X) \leftarrow q(X), \neg -p(X). \qquad -p(X) \leftarrow q(X), \neg p(X).$$

---

$$b(X) \vee r(X) \vee g(X) \leftarrow node(X).$$

zamiast

$$b(X) \leftarrow node(X), \neg r(X), \neg g(X).$$
$$r(X) \leftarrow node(X), \neg b(X), \neg g(X).$$
$$g(X) \leftarrow node(X), \neg r(X), \neg b(X).$$

# Reguły dysjunktywne, przykłady, programy bez negacji 2

$$male(X) \vee female(X) \leftarrow person(X)$$

---

Podzbiór zbioru $q$ $\quad p(X) \vee -p(X) \leftarrow q(X).$

zamiast

$$p(X) \leftarrow q(X), \neg -p(X). \qquad -p(X) \leftarrow q(X), \neg p(X).$$

---

$$b(X) \vee r(X) \vee g(X) \leftarrow node(X).$$

zamiast

$$b(X) \leftarrow node(X), \neg r(X), \neg g(X).$$
$$r(X) \leftarrow node(X), \neg b(X), \neg g(X).$$
$$g(X) \leftarrow node(X), \neg r(X), \neg b(X).$$

# ASP Przykład

$$male(X) \vee female(X) \leftarrow person(X).$$
$$bachleor(X) \leftarrow male(X), \neg married(X).$$
$$person(joey).$$

Modele stabilne:

$M_1 = \{person(joey), female(joey)\}$

$M_2 = \{person(joey), male(joey), bachelor(joey)\}$

$M_3 = \{person(joey), male(joey), married(joey)\}$ nie jest m. stabilnym

# ASP Przykłady

$a \vee b \leftarrow$
$a \leftarrow b$

Jeden model minimalny $\{a\}$

---

$a \vee b \leftarrow$
$a \leftarrow b$
$b \leftarrow a$

Jeden model minimalny $\{a, b\}$

---

$a \vee b \leftarrow$
$\leftarrow a, \neg\, b$

Jeden model stabilny $\{b\}$

---

$a \vee b \leftarrow$
$\leftarrow a, \neg\, b$
$\leftarrow b, \neg\, a$

Nie ma modeli stabilnych

# ASP Przykłady

$a \vee b \leftarrow$
$a \leftarrow b$

Jeden model minimalny $\{a\}$

---

$a \vee b \leftarrow$
$a \leftarrow b$
$b \leftarrow a$

Jeden model minimalny $\{a, b\}$

---

$a \vee b \leftarrow$
$\leftarrow a, \neg b$

Jeden model stabilny $\{b\}$

---

$a \vee b \leftarrow$
$\leftarrow a, \neg b$
$\leftarrow b, \neg a$

Nie ma modeli stabilnych

# Łamigłówka programistyczna

Nieistnienie rozwiązania problemu – brak modelu stabilnego.

Ale jak przekazać informację o nieistnieniu rozwiązania pod-problemu do innej części programu?

Przykład `3color+.dlv`

# Inne rozszerzenia

## Słabe więzy (weak constraints)

$$:\sim \textit{Conjunction} \quad [\textit{Weight}:\texttt{Level}]$$

## Aggregates

Cardinality atom

$$l\{a_1, \ldots, a_n\}k$$

pośród $\{a_1, \ldots, a_n\}$ atomów prawdziwych jest $\geqslant l$, $\leqslant k$

# Uwagi

Porównania z SAT

ASP – wnioskowanie niemonotoniczne, CWA, zamknięty świat

SAT – monotoniczne, otwarty świat

"the unavailability of modeling languages forces users of SAT-based technology to embed their problem encodings in compilers"

"many concepts frequently encountered in KRR problems [can be modelled in ASP] in a fairly easy way.
This includes (un)reachability (and transitive closures), inertia (for solving frame problems), defaults, etc."

# Podsumowanie

# Dziękuję !

a

# Złożoność

Każdą relację rekurencyjnie przeliczalną można zdefiniować za pom. programu (bez negacji).

Dla programów bez zmiennych

Istnienie modelu stabilnego NP-zupełne

Z regułami dysjunktywnymi $\mathrm{NP}^{NP}$-zupełne

Programy pierwszego rzędu (skończony zbiór termów "ustalonych") – eksponencjalny wzrost (NEXP-zupełne)