Beyond Basic Logic Programming

Basic Logic Programming

- Datasets
- Queries
- Updates
- View Definitions
- Operations

Beyond Basic Logic Programming

- View definitions
  - No disjunctions in the head
  - Safe and stratified
- Efficiency of computation
  - Constraint logic programs
  - Existential rules
- Updates
  - Updates to the logic program
  - Constraint checking

Beyond Basic Logic Programming

- View definitions
  - No disjunctions in the dataset (and rule heads)
  - Safe and stratified
- Efficiency of computation
  - Constraint logic programs
  - Existential rules
- Updates
  - Updates to the logic program
  - Constraint checking

Disjunctive Logic Programs

male(joe) | female (joe)

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe:

Herbrand Base:

Herbrand Interpretations:

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Base:

Herbrand Interpretations:

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Base: {male(joe), female(joe)}

Herbrand Interpretations:

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Base: {male(joe), female(joe)}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Semantics

- An interpretation Γ satisfies a ground atom ϕ, if $\phi \in \Gamma$
- An interpretation Γ satisfies a ground negation $\sim\phi$, if $\phi \notin \Gamma$

Semantics

- An interpretation Γ satisfies a ground atom ϕ, if $\phi \in \Gamma$
- An interpretation Γ satisfies a ground negation $\sim\phi$, if $\phi \notin \Gamma$

Closed World Assumption

Semantics

- An interpretation Γ satisfies a ground atom ϕ, if $\phi \in \Gamma$
- An interpretation Γ satisfies a ground negation $\sim\phi$, if $\phi \notin \Gamma$

….

- An interpretation Γ satisfies an arbitrary logic program Ω if and only if Γ satisfies every ground instance of every sentence in Ω.

Semantics

- An interpretation Γ satisfies a ground atom ϕ, if $\phi \in \Gamma$
- An interpretation Γ satisfies a ground negation $\sim\phi$, if $\phi \notin \Gamma$

….

- An interpretation Γ satisfies an arbitrary logic program Ω if and only if Γ satisfies every ground instance of every sentence in Ω.
- A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program, i.e., the set of conclusions is the intersection of all models of the program.

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Models

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Minimal Models

A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program,

Disjunctive Logic Programs

male(joe) | female (joe)

Is male(joe) true?
Is female(joe) true?

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Minimal Models

A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program,

Disjunctive Logic Programs

male(joe) | female (joe)

Is male(joe) true? No

Is female(joe) true? No

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

Minimal Models

{male(joe),female(joe)}

{}

A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program,

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Is male(joe) true? No

Is female(joe) true? No

Is ~male(joe) true?

Is ~female(joe) true?

Minimal Models

A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program,

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Is male(joe) true? No

Is female(joe) true? No

Is ~male(joe) true? Yes

Is ~female(joe) true? Yes

Minimal Models

A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program,

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Is male(joe) true? No

Is female(joe) true? No

Is ~male(joe) true? Yes

Is ~female(joe) true? Yes

Inconsistent

Minimal Models

A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program,

Semantics

- An interpretation Γ satisfies a ground atom ϕ, if $\phi \in \Gamma$
- An interpretation Γ satisfies a ground negation $\sim\phi$, if $\phi \notin \Gamma$

Generalized Closed World Assumptions

- H: Herbrand Base
- D: Definite facts are a union of
  - Set of all facts that are true in all the models
  - Set of all facts that are false in all the models
- I : Indefinite facts are H-D

Generalized Closed World Assumption

- An interpretation Γ satisfies a ground atom ϕ, if $\phi \in \Gamma$
- An interpretation Γ satisfies a ground negation $\sim\phi$, if $\phi \notin \Gamma$
- An interpretation Γ satisfies a ground disjunction $\phi_1, \ldots, \phi_n$, if Γ satisfies at least one ϕ_i.

Generalized Closed World Assumption

- An interpretation Γ satisfies a ground atom ϕ, if $\phi \in \Gamma$
- An interpretation Γ satisfies a ground negation $\sim\phi$, if $\phi \notin \Gamma$
- An interpretation Γ satisfies a ground disjunction $\phi_1, \ldots, \phi_n$, if Γ satisfies at least one ϕ_i.
- An interpretation Γ satisfies an arbitrary logic program Ω if and only if Γ satisfies every ground instance of every sentence in Ω.
- A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program, i.e., the set of conclusions is the intersection of all models of the program.

Generalized Closed World Assumption

- An interpretation Γ satisfies a ground atom ϕ, if $\phi \in \Gamma$
- An interpretation Γ satisfies a ground negation $\sim\phi$, if $\phi \notin \Gamma$
- An interpretation Γ satisfies a ground disjunction $\phi_1, \ldots, \phi_n$, if Γ satisfies at least one ϕ_i.
- An interpretation Γ satisfies an arbitrary logic program Ω if and only if Γ satisfies every ground instance of every sentence in Ω.
- A factoid is *logically entailed* by a closed logic program if and only if it is true in every model of the program, i.e., the set of conclusions is the intersection of all models of the program.

only if it appears in the definite set

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Definite facts: Facts that are true or false in all the minimal models
Indefinite facts: remaining facts

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Definite facts: {}
Indefinite facts: {male(joe), female(joe)}

Disjunctive Logic Programs

male(joe) | female (joe)

Is male(joe) true?
Is female(joe) true?

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Definite facts: {}
Indefinite facts: {male(joe), female(joe)}

Disjunctive Logic Programs

male(joe) | female (joe)

Is male(joe) true? No
Is female(joe) true? No

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Definite facts: {}
Indefinite facts: {male(joe), female(joe)}

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Is male(joe) true? No

Is female(joe) true? No

Is ~male(joe) true?

Is ~female(joe) true?

Definite facts: {}

Indefinite facts: {male(joe), female(joe)}

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Is male(joe) true? No

Is female(joe) true? No

Is ~male(joe) true? No

Is ~female(joe) true? No

Definite facts: {}
Indefinite facts: {male(joe), female(joe)}

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Is male(joe) true? No

Is female(joe) true? No

Is ~male(joe) true? No

Is ~female(joe) true? No

Is male(joe) | female(joe) true?

Definite facts: {}

Indefinite facts: {male(joe), female(joe)}

Disjunctive Logic Programs

male(joe) | female (joe)

Herbrand Universe: {joe}

Herbrand Interpretations:

{male(joe)}

{female(joe)}

{male(joe),female(joe)}

{}

Is male(joe) true? No

Is female(joe) true? No

Is ~male(joe) true? No

Is ~female(joe) true? No

Is male(joe) | female(joe) true? Yes

Definite facts: {}

Indefinite facts: {male(joe), female(joe)}

Disjunctive Logic Programs

- Model intersection property breaks down
  - ie, intersection of all the minimal models is not a model
- Generalized Closed World Assumption is a possible solution
  - Explicitly keep record of definite and indefinite facts

Beyond Basic Logic Programming

- Limitations on view definitions
  - No disjunctions in the dataset
  - Safe and stratified
- Efficiency of computation
  - Constraint logic programs
  - Existential rules
- Updates
  - Updates to the logic program
  - Constraint checking

Constraint Logic Programs

- Consider Peano Arithmetic (Section 10.2 of the textbook)

```
number(0)
number(s(X)) :- number(X)

add(0,Y,Y) :- number(Y)
add(s(X),Y,s(Z)) :- add(X,Y,Z)
```

Constraint Logic Programs

- Consider Peano Arithmetic

```
number(0)
number(s(X)) :- number(X)

add(0,Y,Y) :- number(Y)
add(s(X),Y,s(Z)) :- add(X,Y,Z)
```

```
number(L) & number(M) & add(L,M,N) & add(L,M,s(N))
```

Constraint Logic Programs

- Consider Peano Arithmetic

```
number(0)
number(s(X)) :- number(X)

add(0,Y,Y) :- number(Y)
add(s(X),Y,s(Z)) :- add(X,Y,Z)
```

number(L) & number(M) & add(L,M,N) & add(L,M,s(N))

Runs forever in the standard LP evaluation algorithm

Constraint Logic Programs

- Consider Peano Arithmetic

```
number(0)
number(s(X)) :- number(X)

add(0,Y,Y) :- number(Y)
add(s(X),Y,s(Z)) :- add(X,Y,Z)
```

number(L) & number(M) & add(L,M,N) & add(L,M,s(N))

Runs forever in the standard LP evaluation algorithm

Solution: Check satisfaction of constraints at each step

Constraint Logic Programs

- Direct expression of constraints

sumto(0,0) 0

sumto(1,1) 0+1

sumto(2,3) 0+1+2

sumto(3,6) 0+1+2+3

Constraint Logic Programs

- Direct expression of constraints

```
sumto(0,0)
sumto(N,S) :- N ≥ 1 & N ≤ S & sumto(N-1,S-1)
```

Constraint Logic Programs

- Direct expression of constraints

sumto(0,0)
sumto(N,S) :- N ≥ 1 & N ≤ S & sumto(N-1,S-1)

Prove: S <= 1

$N = N_1 \ \& \ S = S_1 \ \& \ N_1 \geq 1 \ \& \ N_1 \leq S_1 \ \& \ \text{sumto}(N_1\text{-}1, S_1\text{-}1)$

Constraint Logic Programs

- Many problems can be naturally expressed as constraints
  - Map coloring
  - SEND MORE MONEY
- Constraints with floating point numbers
- Distributed constraints
- Constraint optimization (Assignment 4.3)

Beyond Basic Logic Programming

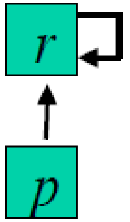
- Limitations on view definitions
  - No disjunctions in the dataset
  - Safe and stratified
- Efficiency of computation
  - Constraint logic programs
  - Existential rules
- Updates
  - Updates to the logic program
  - Constraint checking

Stratified Negation

A set of rules is said to be *stratified* if and only if there is no recursive cycle in the dependency graph involving a negation.

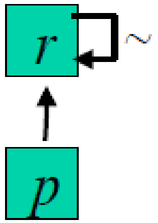
Stratified Negation:

```
r(X,Z) :- p(X,Y)
r(X,Z) :- r(X,Y) & r(Y,Z)
```



Negation that is not stratified:

```
r(X,Z) :- p(X,Y)
r(X,Z) :- p(X,Y) & ~r(Y,Z)
```



*All negations **must** be stratified.*

Minimal Models

If a program has just one minimal model, then every factoid true in that model is trivially true in *every* model of the program.

A logic program that does not contain any negations has a *unique minimal model*.

A logic program with negations can have *more than one minimal model* (in addition to multiple non-minimal models).

If a program is *stratified* (as defined below), then once again there is only one minimal model.

Multiple *Minimal* Models

Dataset

```
p(a,b)
p(b,a)
```

Ruleset

```
r(X) :- p(X,Y) & ~r(Y)
```

Interpretations

p(a,b)	p(a,b)	p(a,b)	p(a,b)
p(b,a)	p(b,a)	p(b,a)	p(b,a)
	r(a)	r(b)	r(a)
			r(b)

Is r(a) true or not? What about r(b)?

The intersection of all models is not necessarily a model!

Answer Set Semantics

- Defining semantics for programs that *may not* be stratified

Answer Set Semantics

- Defining semantics for programs that *may not* be stratified
  - To check if a set S of atoms is an answer set of a program, compute the reduct of the grounded program as follows:
    - For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms
    - We drop rest of the rules
    - We compute the extension of the rules
    - If the extension is the same as S, then S is the answer set of the program

Example

Data Set

p(a,b) p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) & ~r(b)

r(b) :- p(b,a) & ~r(a)

Is p(a,b) an answer set?
p(b,a)

Example

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) & ~r(b)

r(b) :- p(b,a) & ~r(a)

Is p(a,b) an answer set?
p(b,a)

Example

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

Data Set

p(a,b) p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) ~~& ~r(b)~~

r(b) :- p(b,a) ~~& ~r(a)~~

Is p(a,b) an answer set?

p(b,a)

Example

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) ~~& ~r(b)~~

r(b) :- p(b,a) ~~& ~r(a)~~

Is p(a,b) an answer set?

p(b,a)

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

Example

Data Set

p(a,b) p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) ~~& ~r(b)~~

r(b) :- p(b,a) ~~& ~r(a)~~

Is p(a,b) an answer set?
p(b,a)

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

If the extension is the same as S, then S is the answer set of the program

Extension = p(a,b)
p(b,a)
r(a)
r(b)

Example

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) ~~& ~r(b)~~

r(b) :- p(b,a) ~~& ~r(a)~~

Is p(a,b) an answer set? No
p(b,a)

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

If the extension is the same as S, then S is the answer set of the program

Extension = p(a,b)
p(b,a)
r(a)
r(b)

Example

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) & ~r(b)

r(b) :- p(b,a) & ~r(a)

Is p(a,b) an answer set?

p(b,a)

r(a)

Example

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) & ~r(b)

r(b) :- p(b,a) & ~r(a)

Is p(a,b) an answer set?

p(b,a)

r(a)

Example

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) ~~& ~r(b)~~

r(b) :- p(b,a) & ~r(a)

Is p(a,b) an answer set?

p(b,a)

r(a)

Example

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) ~~& ~r(b)~~

~~r(b) :- p(b,a) & ~r(a)~~

Is p(a,b) an answer set?

p(b,a)

r(a)

Example

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) ~~& ~r(b)~~

~~r(b) :- p(b,a) & ~r(a)~~

Is p(a,b) an answer set?
p(b,a)
r(a)

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

If the extension is the same as S, then S is the answer set of the program

Extension = p(a,b)
p(b,a)
r(a)

Example

Data Set

p(a,b) p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) ~~& ~r(b)~~

~~r(b) :- p(b,a) & ~r(a)~~

Is p(a,b) an answer set? Yes

p(b,a)

r(a)

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

If the extension is the same as S, then S is the answer set of the program

Extension = p(a,b)
p(b,a)
r(a)

Example

Data Set

p(a,b)   p(b,a)

Rules

r(X)   :-   p(X,Y)   &   ~r(Y)

Grounded program:

r(a)   :-   p(a,b)   &   ~r(b)

r(b)   :-   p(b,a)   &   ~r(a)

Is p(a,b) an answer set?

p(b,a)

r(a), r(b)

Example

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) & ~r(b)

r(b) :- p(b,a) & ~r(a)

Is p(a,b) an answer set?

p(b,a)

r(a), r(b)

Example

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

r(a) :- p(a,b) & ~r(b)

r(b) :- p(b,a) & ~r(a)

Is p(a,b) an answer set?

p(b,a)

r(a), r(b)

Example

Data Set

p(a,b) p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

~~r(a) :- p(a,b) & ~r(b)~~

~~r(b) :- p(b,a) & ~r(a)~~

Is p(a,b) an answer set?

p(b,a)

r(a), r(b)

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

Example

Data Set

p(a,b)   p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

~~r(a) :- p(a,b) & ~r(b)~~

~~r(b) :- p(b,a) & ~r(a)~~

Is p(a,b) an answer set?

p(b,a)

r(a), r(b)

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

If the extension is the same as S, then S is the answer set of the program

Extension = p(a,b)
p(b,a)

Example

Data Set

p(a,b) p(b,a)

Rules

r(X) :- p(X,Y) & ~r(Y)

Grounded program:

~~r(a) :- p(a,b) & ~r(b)~~

~~r(b) :- p(b,a) & ~r(a)~~

Is p(a,b) an answer set? No
p(b,a)
r(a), r(b)

For any rule that contains negative atoms in the body that do not appear in S, we drop those atoms from the rule, and retain only its positive atoms

We drop the rest of the rules

We compute the extension of the rules

If the extension is the same as S, then S is the answer set of the program

Extension = p(a,b)
p(b,a)

Multiple *Minimal* Models

Dataset

```
p(a,b)
p(b,a)
```

Ruleset

```
r(X) :- p(X,Y) & ~r(Y)
```

Interpretations

p(a,b)	p(a,b)	p(a,b)	p(a,b)
p(b,a)	p(b,a)	p(b,a)	p(b,a)
	r(a)	r(b)	r(a)
			r(b)

Is r(a) true or not? What about r(b)?

The intersection of all models is not necessarily a model!

Implementing an Answer Set Solver

- Start with an empty answer set
- Add one atom at a time to the answer set
- Compute all the atoms that can be derived
  - If a contradiction is obtained abandon that answer set
- Repeat

Implementing an Answer Set Solver

- For a rule r
  - head(r): atom in the head of the rule r
  - positive(r): set of positive atoms in the body of the rule r
  - negative(r): set of the negative atoms in the body of the rule r

Implementing an Answer Set Solver

- For a rule r
  - head(r): atom in the head of the rule r
  - positive(r): set of positive atoms in the body of the rule r
  - negative(r): set of the negative atoms in the body of the rule r

If an atom does not appear in the head of any rule, it cannot appear in any answer set

Implementing an Answer Set Solver

- For a rule r
  - head(r): atom in the head of the rule r
  - positive(r): set of positive atoms in the body of the rule r
  - negative(r): set of the negative atoms in the body of the rule r

If an atom does not appear in the head of any rule, it cannot appear in any answer set

If an atom appears in the answer set S, then there must exist a rule r such that

positive(r) $\subseteq S$

negative(r) $\nsubseteq S$

Implementing an Answer Set Solver

```
compute_answer_sets(P)
    return solve(P, ∅, ∅)

solve(P, CS, CN)
    if expand(P, CS, CN) = false then return ∅
    ⟨CS,CN⟩ ← expand(P,CS,CN)
    Select an atom a ∉ CS ∪ CN
    return solve(P,CS∪{a},CN) ∪ solve(P,CS,CN∪{a})
```

Implementing an Answer Set Solver

```
expand(P, CS, CN)
  repeat
    change ← false
    find all rules r such that
      positive(r) ⊆ CS and negative(r) ⊆ CN
      add head(r) to CS
      change ← true
    if all rules r with same head satisfy that
      positive(r) ∩ CN ≠ Ø or negative(r) ∩ CS ≠ Ø
      add head(r) to CN
      change ← true
  until change is false
  if CS ∩ CN = Ø return ⟨CS,CN⟩ else return false
```

Implementing an Answer Set Solver

p(a,b)
p(b,a)
r(a) :- p(a,b) & ~r(b)
r(b) :- p(b,a) & ~r(a)

```
expand(P, CS, CN)
  repeat
    change ← false
    find all rules r such that
      positive(r) ⊆ CS and negative(r) ⊆ CN
      add head(r) to CS
      change ← true
  if all rules r with same head satisfy that
      positive(r) ∩ CN ≠ ∅ or negative(r) ∩ CS ≠ ∅
      add head(r) to CN
      change ← true
  until change is false
  if CS ∩ CN = ∅ return ⟨CS,CN⟩ else return false
```

CS = ∅ CN = ∅

Implementing an Answer Set Solver

p(a,b)
p(b,a)
r(a) :- p(a,b) & ~r(b)
r(b) :- p(b,a) & ~r(a)

expand(P, CS, CN)

repeat

change ← **false**

find all rules r such that

positive(r) ⊆ CS and negative(r) ⊆ CN

add head(r) to CS

change ← **true**

if all rules r with same head satisfy that

positive(r) ∩ CN ≠ ∅ or negative(r) ∩ CS ≠ ∅

add head(r) to CN

change ← **true**

until change is **false**

if CS ∩ CN = ∅ **return** ⟨CS,CN⟩ **else** return **false**

CS = ∅ CN = ∅

CS={p(a,b),p(b,a)}

Implementing an Answer Set Solver

p(a,b)
p(b,a)
r(a) :- p(a,b) & ~r(b)
r(b) :- p(b,a) & ~r(a)

expand(P, CS, CN)
  repeat
   change ← **false**
   find all rules r such that
     positive(r) ⊆ CS and negative(r) ⊆ CN
     add head(r) to CS
     change ← **true**
   if all rules r with same head satisfy that
     positive(r) ∩ CN ≠ ∅ or negative(r) ∩ CS ≠ ∅
     add head(r) to CN
     change ← **true**
   until change is **false**
  if CS ∩ CN = ∅ **return** ⟨CS,CN⟩ **else** return **false**

CS = ∅ CN = ∅

CS={p(a,b),p(b,a)}

CS={p(a,b),p(b,a)} CN = ∅

Implementing an Answer Set Solver

p(a,b)
p(b,a)
r(a) :- p(a,b) & ~r(b)
r(b) :- p(b,a) & ~r(a)

expand(P, CS, CN)

repeat

change ← **false**

find all rules r such that

positive(r) ⊆ CS and negative(r) ⊆ CN

add head(r) to CS

change ← **true**

if all rules r with same head satisfy that

positive(r) ∩ CN ≠ ∅ or negative(r) ∩ CS ≠ ∅

add head(r) to CN

change ← **true**

until change is **false**

if CS ∩ CN = ∅ **return** ⟨CS,CN⟩ **else** return **false**

CS={p(a,b),p(b,a),r(a)} CN = ∅

CS={p(a,b),p(b,a),r(a)} CN = {r(b)}

CS={p(a,b),p(b,a),r(a)} CN = {r(b)}

Implementing an Answer Set Solver

p(a,b)
p(b,a)
r(a) :- p(a,b) & ~r(b)
r(b) :- p(b,a) & ~r(a)

expand(P, CS, CN)

repeat

change ← **false**

find all rules r such that

positive(r) ⊆ CS and negative(r) ⊆ CN

add head(r) to CS

change ← **true**

if all rules r with same head satisfy that

positive(r) ∩ CN ≠ ∅ or negative(r) ∩ CS ≠ ∅

add head(r) to CN

change ← **true**

until change is **false**

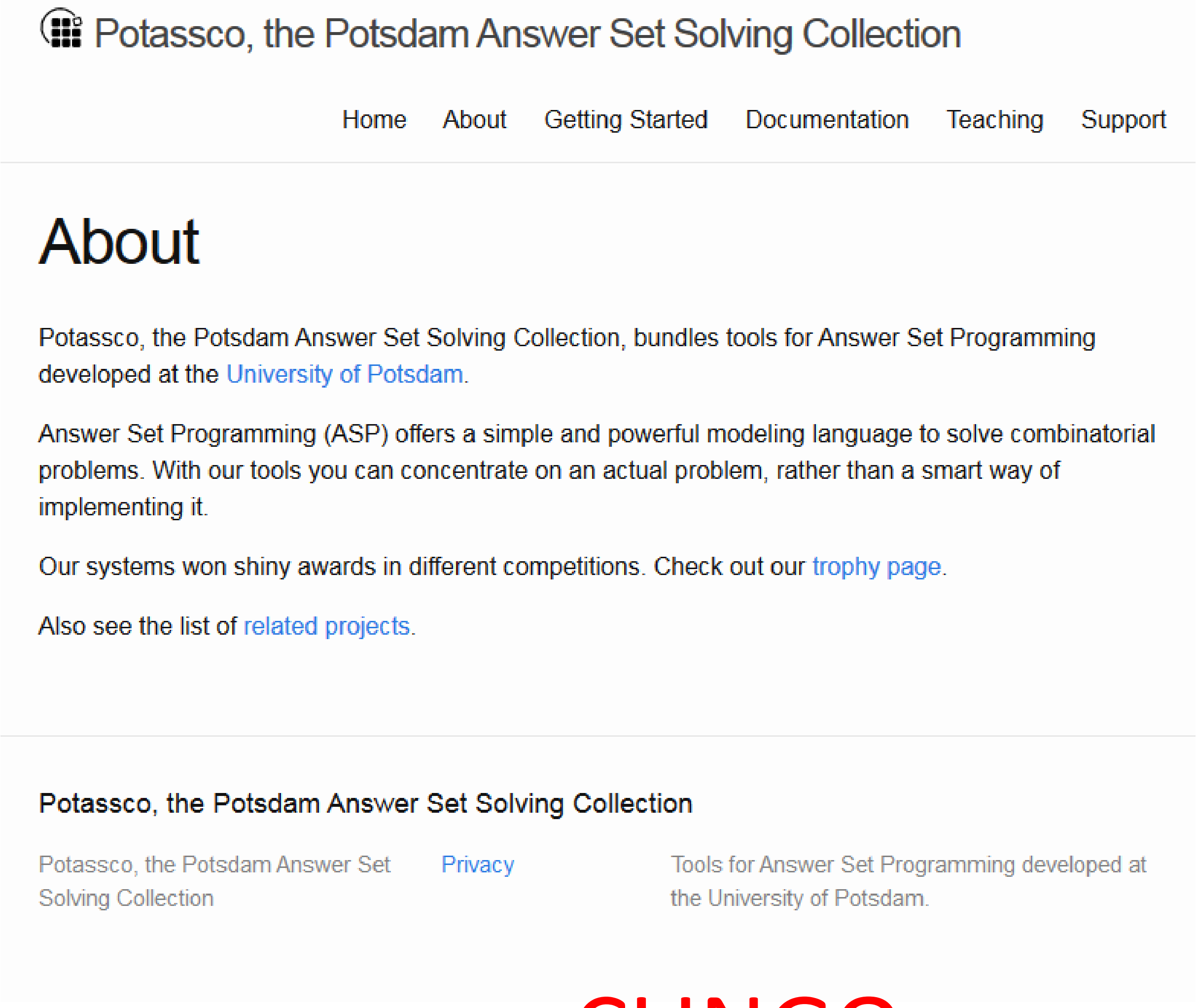
if CS ∩ CN = ∅ **return** ⟨CS,CN⟩ **else** return **false**

CS={p(a,b),p(b,a)} CN = r(a)

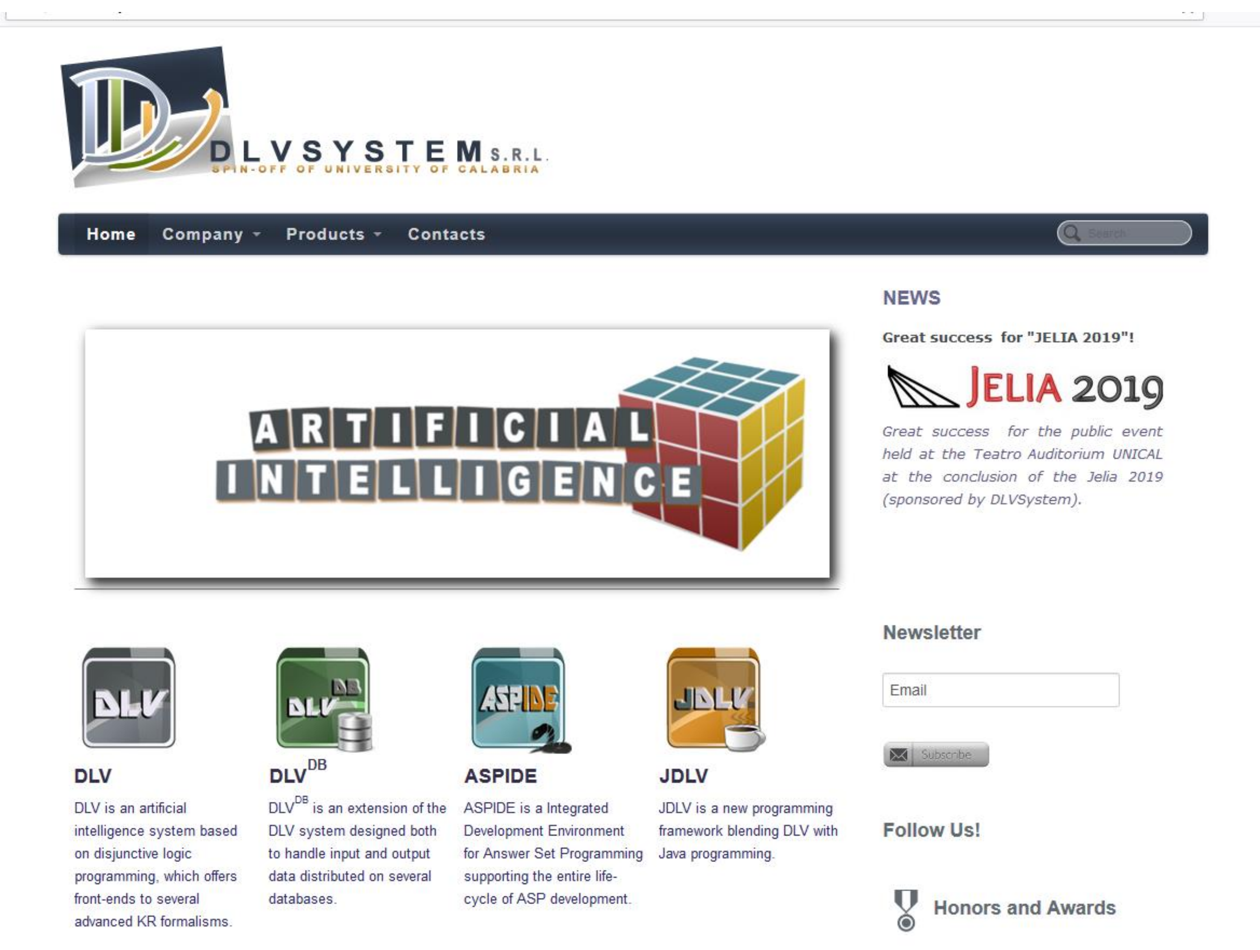
CS={p(a,b),p(b,a), r(b)} CN = $\{\mathrm{r(a)}\}$

CS={p(a,b),p(b,a), r(b)} CN = $\{\mathrm{r(a)}\}$

Available Answer Set Solvers



CLINGO



DLV

Extensions to ASP

- Choice rule
  - Disjunctions
- Constraints
- Classical negation

Choice Rule

- Enclose a set of atoms in curly braces
  - Choose in all possible ways which atoms will be included in the answer set

{ p(1), p(2) }

Possible answer sets are ∅,{p(1)}, {p(2)}, {p(1), p(2)}

Choice Rule

- Enclose a set of atoms in curly braces
  - Choose in all possible ways which atoms will be included in the answer set
  - Can also indicate bounds on the number of atoms to be included

{ p(1), p(2) }
Possible answer sets are $\emptyset$,{p(1)}, {p(2)}, {p(1), p(2)}

1 { p(1), p(2) } 1
Possible answer sets are {p(1)}, {p(2)}

Constraint

- A rule with an empty head

{ p(1), p(2) }
Possible answer sets are Ø,{p(1)}, {p(2)}, {p(1), p(2)}

:- p(1), ~p(2)
Possible answer sets are Ø, {p(2)}, {p(1), p(2)}

Constraint

- A rule with an empty head
  - A constraint is an unstratified rule
  - Stratification is defined only for rules with a head
    - Therefore, we have to convert a constraint to a rule with a head

```
:- p
q :- p, ~q
```

Classical Negation

- The predicates can have a classical negation symbol in front of them
    - -p(a) indicates that we know for sure that p(a) is false
    - ~p(a) indicates that p(a) could be true or false
- Two negation operators can be related
    - -p :- ~p

Beyond Basic Logic Programming

- Limitations on view definitions
  - No disjunctions in the dataset (and rule heads)
  - Safe and stratified
- Efficiency of computation
  - Constraint logic programs
  - Existential rules
- Updates
  - Updates to the logic program
  - Constraint checking

Existential Rules

- A rule that has a functional term in its head

```
owns(X,house(X)) :- instance_of(X,person)
has_parent(X,f(X)) :- instance_of(X,person)
has_parent(X,g(X)) :- instance_of(X,male)
```

Existential Rules

- In the context of database systems

has parent	
john	peter
sue	peter
peter	??
…	…

Also known as:

Tuple generating dependencies (in relational databases)

Existential Rules

- In the context of description logic systems

Person $\sqcap$ ($\exists$has_parent.Person)

Also known as:

Existential rules

Problems with Existential Rules

- Termination

has_parent(X,f(X)) :- instance_of(X,person)

Unrestricted application of this rule leads to infinite recursion

Problems with Existential Rules

- Under-specification when used with a class hierarchy

has_parent(X,f(X)) :- instance_of(X,person)

subclass_of(male,person)

has_parent(X,g(X)) :- instance_of(X,male)

What is the relationship between f(X) and g(X)?

Solutions for Existential Rules

- Ensure termination by design
- Limit depth of reasoning
- Rule strengthening

Beyond Basic Logic Programming

- Limitations on view definitions
  - No disjunctions in the dataset (and rule heads)
  - Safe and stratified
- Efficiency of computation
  - Constraint logic programs
  - Existential rules
- Updates
  - Updates to the logic program
  - Constraint checking

Updates

- What if the view definitions themselves need to be updated?
  - Naturally happens during rule authoring
    - Dropping a relation used in multiple rules
- What if an update to the dataset violates some constraint?
  - For example, asserting two fathers of a person using a dynamic rule

Beyond Basic Logic Programming

- Limitations on view definitions
  - No disjunctions in the dataset (and rule heads)
  - Safe and stratified
- Efficiency of computation
  - Constraint logic programs
  - Existential rules
- Updates
  - Updates to the logic program
  - Constraint checking