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

\[ \text{male}(joe) \mid \text{female} \ (joe) \]

**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

\[ \text{male}(\text{joe}) \mid \text{female} (\text{joe}) \]

**Herbrand Universe:** \{joe\}

**Herbrand Base:** \{\text{male}(\text{joe}), \text{female}(\text{joe})\}

**Herbrand Interpretations:**
- \{\text{male}(\text{joe})\}
- \{\text{female}(\text{joe})\}
- \{\text{male}(\text{joe}), \text{female}(\text{joe})\}
- \{\}\
Semantics

• An interpretation $\Gamma$ satisfies a ground atom $\phi$, if $\phi \in \Gamma$
• An interpretation $\Gamma$ satisfies a ground negation $\neg \phi$, if $\phi \notin \Gamma$
Semantics

• An interpretation $\Gamma$ satisfies a ground atom $\phi$, if $\phi \in \Gamma$
• An interpretation $\Gamma$ satisfies a ground negation $\neg \phi$, if $\phi \notin \Gamma$

Closed World Assumption
Semantics

• An interpretation $\Gamma$ satisfies a ground atom $\phi$, if $\phi \in \Gamma$
• An interpretation $\Gamma$ satisfies a ground negation $\neg \phi$, if $\phi \notin \Gamma$

....

• An interpretation $\Gamma$ satisfies an arbitrary logic program $\Omega$ if and only if $\Gamma$ satisfies every ground instance of every sentence in $\Omega$. 
Semantics

• An interpretation \( \Gamma \) satisfies a ground atom \( \phi \), if \( \phi \in \Gamma \)
• An interpretation \( \Gamma \) satisfies a ground negation \( \neg \phi \), if \( \phi \notin \Gamma \)

....

• An interpretation \( \Gamma \) satisfies an arbitrary logic program \( \Omega \) if and only if \( \Gamma \) satisfies every ground instance of every sentence in \( \Omega \).
• 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

\[ \text{male}(joe) \lor \text{female}(joe) \]

Herbrand Universe: \{joe\}

Herbrand Interpretations:
- \{\text{male}(joe)\}
- \{\text{female}(joe)\}
- \{\text{male}(joe),\text{female}(joe)\}
- \{\}

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

\[
\text{male}(joe) \lor \text{female}(joe)
\]

Is \text{male}(joe) true?  
Is \text{female}(joe) true?

Herbrand Universe: \{joe\}

Herbrand Interpretations:
- \{\text{male}(joe)\}
- \{\text{female}(joe)\}
- \{\text{male}(joe), \text{female}(joe)\}
- \{

Minimal Models

A factoid is \textit{logically entailed} by a closed logic program if and only if it is true in every model of the program,
Disjunctive Logic Programs

\[
\text{male(joe)} \mid \text{female (joe)}
\]

Herbrand Universe: \{joe\}

Herbrand Interpretations:
\[
\{\text{male(joe)}\}
\{\text{female(joe)}\}
\{\text{male(joe),female(joe)}\}
\{\}
\]

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

\[ \text{male(joe)} | \text{female (joe)} \]

Herbrand Universe: \{joe\}

Herbrand Interpretations:
\[ \{\text{male(joe)}\} \]
\[ \{\text{female(joe)}\} \]
\[ \{\text{male(joe)},\text{female(joe)}\} \]
\[ \{\} \]

Minimal Models

Is \text{male(joe)} true? No
Is \text{female(joe)} true? No
Is ~\text{male(joe)} true?  
Is ~\text{female(joe)} true?

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

\[
\text{male(joe) | female(joe)}
\]

Herbrand Universe: \{joe\}

Herbrand Interpretations:
\[
\{\text{male(joe)}\} \\
\{\text{female(joe)}\} \\
\{\text{male(joe),female(joe)}\} \\
\{\}
\]

Minimal Models

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

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)}
{}

Minimal Models

Is male(joe) true? No
Is female(joe) true? No
Is ¬male(joe) true? Yes
Is ¬female(joe) true? Yes
Inconsistent

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 $\Gamma$ satisfies a ground atom $\phi$, if $\phi \in \Gamma$
• An interpretation $\Gamma$ satisfies a ground negation $\neg \phi$, 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 $\Gamma$ satisfies a ground atom $\phi$, if $\phi \in \Gamma$
• An interpretation $\Gamma$ satisfies a ground negation $\sim \phi$, if $\phi \notin \Gamma$
• An interpretation $\Gamma$ satisfies a ground disjunction $\phi_1, \ldots, \phi_n$, if $\Gamma$ satisfies at least one $\phi_i$. 

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 $\Gamma$ satisfies a ground atom $\phi$, if $\phi \in \Gamma$
• An interpretation $\Gamma$ satisfies a ground negation $\neg \phi$, if $\phi \notin \Gamma$
• An interpretation $\Gamma$ satisfies a ground disjunction $\phi_1, \ldots, \phi_n$, if $\Gamma$ satisfies at least one $\phi_i$.
• An interpretation $\Gamma$ satisfies an arbitrary logic program $\Omega$ if and only if $\Gamma$ satisfies every ground instance of every sentence in $\Omega$.
• 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 $\Gamma$ satisfies a ground atom $\phi$, if $\phi \in \Gamma$
- An interpretation $\Gamma$ satisfies a ground negation $\neg\phi$, if $\phi \notin \Gamma$
- An interpretation $\Gamma$ satisfies a ground disjunction $\phi_1, \ldots, \phi_n$, if $\Gamma$ satisfies at least one $\phi_i$.
- An interpretation $\Gamma$ satisfies an arbitrary logic program $\Omega$ if and only if $\Gamma$ satisfies every ground instance of every sentence in $\Omega$.
- 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

\[ \text{male}(joe) \mid \text{female}(joe) \]

Herbrand Universe: \{joe\}

Herbrand Interpretations:
\[
\{\text{male}(joe)\}
\{\text{female}(joe)\}
\{\text{male}(joe),\text{female}(joe)\}
\{\}
\]

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

\[
\text{male(joe)} \mid \text{female (joe)}
\]

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

Herbrand Universe: \{joe\}

Herbrand Interpretations:
\begin{align*}
\{\text{male(joe)}\} \\
\{\text{female(joe)}\} \\
\{\text{male(joe)},\text{female(joe)}\} \\
\{\}
\end{align*}

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

\[ \text{male}(joe) \lor \text{female}(joe) \]

Herbrand Universe: \{joe\}

Herbrand Interpretations:
\[
\begin{align*}
\{ & \text{male}(joe) \} \\
\{ & \text{female}(joe) \} \\
\{ & \text{male}(joe), \text{female}(joe) \} \\
\{ & \}
\end{align*}
\]

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

Is male(joe) true? No
Is female(joe) true? No
Is \neg\text{male}(joe) true? No
Is \neg\text{female}(joe) true?
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)}

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

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?
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)}

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
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

\[
\text{number}(0) \\
\text{number}(\text{s}(X)) :\text{- number}(X) \\
\text{add}(0,Y,Y) :\text{- number}(Y) \\
\text{add}(\text{s}(X),Y,\text{s}(Z)) :\text{- add}(X,Y,Z) \\
\text{number}(L) \& \text{number}(M) \& \text{add}(L,M,N) \& \text{add}(L,M,\text{s}(N))
\]
Constraint Logic Programs

• Consider Peano Arithmetic

\[
\begin{align*}
\text{number}(0) \\
\text{number}(\text{s}(X)) & : - \text{number}(X) \\
\text{add}(0,Y,Y) & : - \text{number}(Y) \\
\text{add}(s(X),Y,s(Z)) & : - \text{add}(X,Y,Z)
\end{align*}
\]

\[
\text{number}(L) \& \text{number}(M) \& \text{add}(L,M,N) \& \text{add}(L,M,s(N))
\]

Runs forever in the standard LP evaluation algorithm
Constraint Logic Programs

• Consider Peano Arithmetic

\[
\begin{align*}
\text{number}(0) \\
\text{number}(s(X)) & :\text{-} \text{number}(X) \\
\text{add}(0,Y,Y) & :\text{-} \text{number}(Y) \\
\text{add}(s(X),Y,s(Z)) & :\text{-} \text{add}(X,Y,Z)
\end{align*}
\]

\[
\text{number}(L) \& \text{number}(M) \& \text{add}(L,M,N) \& \text{add}(L,M,s(N))
\]

Runs forever in the standard LP evaluation algorithm
Solution: Check satisfaction of constraints at each step
Constraint Logic Programs

- Direct expression of constraints
  
  \[
  \text{sumto}(0,0) \quad 0 \\
  \text{sumto}(1,1) \quad 0+1 \\
  \text{sumto}(2,3) \quad 0+1+2 \\
  \text{sumto}(3,6) \quad 0+1+2+3 
  \]
Constraint Logic Programs

• Direct expression of constraints

\[
\text{sumto}(0,0) \\
\text{sumto}(N,S) \leftarrow N \geq 1 \land N \leq S \land \text{sumto}(N-1,S-1)
\]
Constraint Logic Programs

• Direct expression of constraints

\[
\text{sumto}(0,0) \\
\text{sumto}(N,S) :\text{\textit{-} } N \geq 1 & N \leq S & \text{sumto}(N-1,S-1)
\]

Prove: \( S \leq 1 \)

\[
N = N_1 & S = S_1 & N_1 \geq 1 & N_1 \leq S_1 & \text{sumto}(N_1-1,S_1-1)
\]
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:
\[
\begin{align*}
  r(X,Z) & : - p(X,Y) \\
  r(X,Z) & : - r(X,Y) \& r(Y,Z)
\end{align*}
\]

Negation that is not stratified:
\[
\begin{align*}
  r(X,Z) & : - p(X,Y) \\
  r(X,Z) & : - p(X,Y) \& \neg r(Y,Z)
\end{align*}
\]

*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**

\[ r(X) :- p(X,Y) \land \neg r(Y) \]

**Interpretations**

<table>
<thead>
<tr>
<th>p(a,b)</th>
<th>p(a,b)</th>
<th>p(a,b)</th>
<th>p(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(b,a)</td>
<td>p(b,a)</td>
<td>p(b,a)</td>
<td>p(b,a)</td>
</tr>
<tr>
<td>r(a)</td>
<td>r(b)</td>
<td>r(a)</td>
<td>r(b)</td>
</tr>
</tbody>
</table>

*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

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) \quad :\quad p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) \quad :\quad p(a,b) \quad \& \quad \neg r(b) \]
\[ r(b) \quad :\quad p(b,a) \quad \& \quad \neg r(a) \]

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

\[ p(b,a) \]
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) \quad :\quad p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) \quad :\quad p(a,b) \quad \& \quad \neg r(b) \]

\[ r(b) \quad :\quad p(b,a) \quad \& \quad \neg r(a) \]

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.
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) \,:-\, p(X,Y) \quad &\quad \neg r(Y) \]

Grounded program:

\[ r(a) \,:-\, p(a,b) \quad &\quad \neg r(b) \]
\[ r(b) \,:-\, p(b,a) \quad &\quad \neg r(a) \]

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.
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) \quad : \quad p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) \quad : \quad p(a,b) \quad \& \quad \neg r(b) \]
\[ r(b) \quad : \quad p(b,a) \quad \& \quad \neg r(a) \]

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

\( p(b,a) \)
Example

Data Set
\[ p(a,b) \quad p(b,a) \]

Rules
\[ r(X) : \quad -p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:
\[ r(a) : \quad -p(a,b) \quad \& \quad \neg r(b) \]
\[ r(b) : \quad -p(b,a) \quad \& \quad \neg r(a) \]

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

\[ p(a, b) \quad p(b, a) \]

Rules

\[ r(X) : - p(X, Y) \quad \& \quad \sim r(Y) \]

Grounded program:

\[ r(a) : - p(a, b) \quad \& \quad \sim r(b) \]
\[ r(b) : - p(b, a) \quad \& \quad \sim r(a) \]

Is \( p(a, b) \) an answer set? \( \text{No} \)

\[ p(b, a) \]

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 =

\[ p(a, b) \quad p(b, a) \quad r(a) \quad r(b) \]
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) \quad : \quad \neg p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) \quad : \quad p(a,b) \quad \& \quad \neg r(b) \]
\[ r(b) \quad : \quad p(b,a) \quad \& \quad \neg r(a) \]

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

\[ p(b,a)\]
\[ r(a) \]
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) \quad :\quad \neg p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) \quad :\quad p(a,b) \quad \& \quad \neg r(b) \]
\[ r(b) \quad :\quad p(b,a) \quad \& \quad \neg r(a) \]

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

\[ p(b,a) \quad r(a) \]

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.
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) \quad :\quad p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) \quad :\quad p(a,b) \quad \& \quad \neg r(b) \]
\[ r(b) \quad :\quad p(b,a) \quad \& \quad \neg r(a) \]

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

\[ p(b,a) \quad r(a) \]

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.
Example

Data Set

\[ p(a, b) \quad p(b, a) \]

Rules

\[ r(X) : \neg p(X, Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) : \neg p(a, b) \quad \& \quad \neg r(b) \]

\[ r(b) : \neg p(b, a) \quad \& \quad \neg r(a) \]

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

\[ p(b, a) \quad r(a) \]

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.
Example

Data Set
\[ p(a, b) \quad p(b, a) \]

Rules
\[ r(X) \quad : \quad p(X,Y) \quad & \quad \neg r(Y) \]

Grounded program:
\[ r(a) \quad : \quad p(a, b) \quad & \quad \neg r(b) \]
\[ r(b) \quad : \quad p(b, a) \quad & \quad \neg r(a) \]

Is \( p(a, b) \) an answer set?
\[ p(b, a) \quad r(a) \]

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.

\[ \text{Extension} = \quad \begin{align*} p(a, b) \\ p(b, a) \\ r(a) \end{align*} \]
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) : - \quad p(X,Y) \quad & \quad \neg r(Y) \]

Grounded program:

\[ r(a) : - \quad p(a,b) \quad & \quad \neg r(b) \]

\[ r(b) : - \quad p(b,a) \quad & \quad \neg r(a) \]

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

Extension = \[ p(a,b) \quad p(b,a) \quad r(a) \]

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.
Example

Data Set

\[ p(a, b) \quad p(b, a) \]

Rules

\[ r(X) \quad : \quad p(X, Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) \quad : \quad p(a, b) \quad \& \quad \neg r(b) \]
\[ r(b) \quad : \quad p(b, a) \quad \& \quad \neg r(a) \]

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

\[ p(b, a) \]
\[ r(a), r(b) \]
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) : \quad p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) : \quad p(a,b) \quad \& \quad \neg r(b) \]

\[ r(b) : \quad p(b,a) \quad \& \quad \neg r(a) \]

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

\[ p(b,a) \]

\[ r(a), \ r(b) \]
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) \quad :\quad \neg p(X,Y) \quad \& \quad \neg r(Y) \]

Grounded program:

\[ r(a) \quad :\quad \neg p(a,b) \quad \& \quad \neg r(b) \]

\[ r(b) \quad :\quad \neg p(b,a) \quad \& \quad \neg r(a) \]

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

\[ p(b,a) \]
\[ r(a), r(b) \]
Example

Data Set

\[ p(a,b) \quad p(b,a) \]

Rules

\[ r(X) : \quad p(X,Y) \quad & \quad \neg r(Y) \]

Grounded program:

\[ r(a) : \quad p(a,b) \quad & \quad \neg r(b) \]

\[ r(b) : \quad p(b,a) \quad & \quad \neg r(a) \]

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

\[ p(b,a) \quad r(a), \quad r(b) \]

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

\[ \text{p(a,b) \quad p(b,a)} \]

Rules

\[ \text{r(X) :- \quad p(X,Y) \quad & \quad \neg r(Y)} \]

Grounded program:

\[ \text{r(a) :- \quad p(a,b) \quad & \quad \neg r(b)} \]

\[ \text{r(b) :- \quad p(b,a) \quad & \quad \neg r(a)} \]

Is \( \text{p(a,b)} \) an answer set?

\( \text{p(b,a)} \)

\( \text{r(a), r(b)} \)

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 = \( \text{p(a,b)} \)

\( \text{p(b,a)} \)
Example

Data Set
\[ p(a, b) \quad p(b, a) \]

Rules
\[ r(X) \colon \neg p(X, Y) \land \neg r(Y) \]

Grounded program:
\[ r(a) \colon \neg p(a, b) \land \neg r(b) \]
\[ r(b) \colon \neg p(b, a) \land \neg r(a) \]

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.

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

\[ p(a, b) \quad p(b, a) \]
\[ r(a), r(b) \]

Extension = \[ p(a, b) \]
\[ p(b, a) \]
Multiple Minimal Models

Dataset

\[ p(a,b) \]
\[ p(b,a) \]

Ruleset

\[ r(X) :- p(X,Y) \land \sim r(Y) \]

Interpretations

<table>
<thead>
<tr>
<th>p(a,b)</th>
<th>p(a,b)</th>
<th>p(a,b)</th>
<th>p(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(b,a)</td>
<td>p(b,a)</td>
<td>p(b,a)</td>
<td>p(b,a)</td>
</tr>
<tr>
<td>r(a)</td>
<td>r(b)</td>
<td>r(a)</td>
<td>r(b)</td>
</tr>
</tbody>
</table>

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 $r$
  • $\text{head}(r)$: atom in the head of the rule $r$
  • $\text{positive}(r)$: set of positive atoms in the body of the rule $r$
  • $\text{negative}(r)$: 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 $r$
  • $\text{head}(r)$: atom in the head of the rule $r$
  • $\text{positive}(r)$: set of positive atoms in the body of the rule $r$
  • $\text{negative}(r)$: 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
\[\text{positive}(r) \subseteq S\]
\[\text{negative}(r) \not\subseteq S\]
Implementing an Answer Set Solver

```plaintext
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 U CN
    return solve(P,CS∪{a},CN) U solve(P,CS,CNU{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 ≠ ∅ or negative(r) ∩ CS ≠ ∅
add head(r) to CN
change ← true
until change is false
if CS ∩ CN = ∅ return ⟨CS,CN⟩ else return false
Implementing an Answer Set Solver

\[
p(a,b) \\
p(b,a) \\
r(a) :\ p(a,b) \land \neg r(b) \\
r(b) :\ p(b,a) \land \neg r(a)
\]

\[
\text{CS} = \emptyset \quad \text{CN} = \emptyset
\]

\[
\text{expand}(P, \text{CS}, \text{CN}) \\
\text{repeat} \\
\quad \text{change} \leftarrow \text{false} \\
\quad \text{find all rules r such that} \\
\quad \quad \text{positive}(r) \subseteq \text{CS} \text{ and negative}(r) \subseteq \text{CN} \\
\quad \quad \text{add head}(r) \text{ to CS} \\
\quad \quad \text{change} \leftarrow \text{true} \\
\quad \text{if all rules r with same head satisfy that} \\
\quad \quad \text{positive}(r) \cap \text{CN} \neq \emptyset \text{ or negative}(r) \cap \text{CS} \neq \emptyset \\
\quad \quad \text{add head}(r) \text{ to CN} \\
\quad \quad \text{change} \leftarrow \text{true} \\
\text{until change is false} \\
\text{if} \ \text{CS} \cap \text{CN} = \emptyset \ \text{return} \ (\text{CS,CN}) \ \text{else return false}
\]
Implementing an Answer Set Solver

\begin{align*}
\text{p(a,b)} \\
p(b,a) \\
r(a) :&= \text{p(a,b)} \land \neg\text{r(b)} \\
r(b) :&= \text{p(b,a)} \land \neg\text{r(a)}
\end{align*}

\text{expand(P, CS, CN)}

\begin{verbatim}
repeat
    change \leftarrow \text{false}
    find all rules \(r\) such that
        \text{positive}(r) \subseteq \text{CS} \text{ and } \text{negative}(r) \subseteq \text{CN}
        add head(r) to \text{CS}
        change \leftarrow \text{true}
    if all rules \(r\) with same head satisfy that
        \text{positive}(r) \cap \text{CN} \neq \emptyset \text{ or } \text{negative}(r) \cap \text{CS} \neq \emptyset
        add head(r) to \text{CN}
        change \leftarrow \text{true}
until change is \text{false}
if CS \cap CN = \emptyset \text{ return } (\text{CS}, \text{CN}) \text{ else return false}
\end{verbatim}

\(\text{CS} = \emptyset \quad \text{CN} = \emptyset\)

\(\text{CS} = \{\text{p(a,b)}, \text{p(b,a)}\}\)
Implementing an Answer Set Solver

\[
\begin{align*}
p(a,b) \\
p(b,a) \\
r(a) &: p(a,b) \& \neg r(b) \\
r(b) &: p(b,a) \& \neg r(a)
\end{align*}
\]

\[
\begin{align*}
\text{CS} &= \emptyset & \text{CN} &= \emptyset \\
\text{expand}(P, \text{CS}, \text{CN}) \\
\text{repeat} \\
\quad \text{change} \leftarrow \text{false} \\
\quad \text{find all rules } r \text{ such that} \\
\quad \quad \text{positive}(r) \subseteq \text{CS} \text{ and negative}(r) \subseteq \text{CN} \\
\quad \quad \text{add head}(r) \text{ to } \text{CS} \\
\quad \quad \text{change} \leftarrow \text{true} \\
\quad \text{if all rules } r \text{ with same head satisfy that} \\
\quad \quad \text{positive}(r) \cap \text{CN} \neq \emptyset \text{ or negative}(r) \cap \text{CS} \neq \emptyset \\
\quad \quad \text{add head}(r) \text{ to } \text{CN} \\
\quad \quad \text{change} \leftarrow \text{true} \\
\text{until} \ \text{change} \ \text{is} \ \text{false} \\
\text{if } \text{CS} \cap \text{CN} = \emptyset \ \text{return} \ (\text{CS}, \text{CN}) \ \text{else} \ \text{return} \ \text{false} \\
\end{align*}
\]

\[
\begin{align*}
\text{CS} &= \{p(a,b),p(b,a)\} \\
\text{CN} &= \emptyset
\end{align*}
\]
Implementing an Answer Set Solver

\[
\text{expand}(P, CS, CN)
\]

\[
\begin{align*}
\text{repeat} \\
\text{change} & \leftarrow \text{false} \\
\text{find all rules } r \text{ such that} \\
& \text{positive}(r) \subseteq CS \text{ and negative}(r) \subseteq CN \\
& \text{add head}(r) \text{ to CS} \\
& \text{change} \leftarrow \text{true} \\
\text{if all rules } r \text{ with same head satisfy that} \\
& \text{positive}(r) \cap CN \neq \emptyset \text{ or negative}(r) \cap CS \neq \emptyset \\
& \text{add head}(r) \text{ to CN} \\
& \text{change} \leftarrow \text{true} \\
\text{until} \text{ change is } \text{false} \\
\text{if } CS \cap CN = \emptyset \text{ return } \langle CS, CN \rangle \text{ else return } \text{false}
\end{align*}
\]

\[
\begin{align*}
CS &= \{p(a,b), p(b,a), r(a)\} \quad CN = \emptyset \\
\text{repeat} \\
\text{change} & \leftarrow \text{false} \\
\text{find all rules } r \text{ such that} \\
& \text{positive}(r) \subseteq CS \text{ and negative}(r) \subseteq CN \\
& \text{add head}(r) \text{ to CS} \\
& \text{change} \leftarrow \text{true} \\
\text{if all rules } r \text{ with same head satisfy that} \\
& \text{positive}(r) \cap CN \neq \emptyset \text{ or negative}(r) \cap CS \neq \emptyset \\
& \text{add head}(r) \text{ to CN} \\
& \text{change} \leftarrow \text{true} \\
\text{until} \text{ change is } \text{false} \\
\text{if } CS \cap CN = \emptyset \text{ return } \langle CS, CN \rangle \text{ else return } \text{false}
\end{align*}
\]

\[
\begin{align*}
CS &= \{p(a,b), p(b,a), r(a)\} \quad CN = \{r(b)\} \\
\text{repeat} \\
\text{change} & \leftarrow \text{false} \\
\text{find all rules } r \text{ such that} \\
& \text{positive}(r) \subseteq CS \text{ and negative}(r) \subseteq CN \\
& \text{add head}(r) \text{ to CS} \\
& \text{change} \leftarrow \text{true} \\
\text{if all rules } r \text{ with same head satisfy that} \\
& \text{positive}(r) \cap CN \neq \emptyset \text{ or negative}(r) \cap CS \neq \emptyset \\
& \text{add head}(r) \text{ to CN} \\
& \text{change} \leftarrow \text{true} \\
\text{until} \text{ change is } \text{false} \\
\text{if } CS \cap CN = \emptyset \text{ return } \langle CS, CN \rangle \text{ else return } \text{false}
\end{align*}
\]
Implementing an Answer Set Solver

\[
\begin{align*}
\text{CS} &= \{p(a, b), p(b, a)\} & \text{CN} &= r(a) \\
\end{align*}
\]

\[
\begin{align*}
\text{expand}(P, \text{CS}, \text{CN}) \\
\text{repeat} \\
\text{change} \leftarrow \text{false} \\
\text{find all rules } r \text{ such that} \\
\quad \text{positive}(r) \subseteq \text{CS} \text{ and negative}(r) \subseteq \text{CN} \\
\quad \text{add head}(r) \text{ to } \text{CS} \\
\quad \text{change} \leftarrow \text{true} \\
\text{if all rules } r \text{ with same head satisfy that} \\
\quad \text{positive}(r) \cap \text{CN} \neq \emptyset \text{ or negative}(r) \cap \text{CS} \neq \emptyset \\
\quad \text{add head}(r) \text{ to } \text{CN} \\
\quad \text{change} \leftarrow \text{true} \\
\text{until} \text{ change is } \text{false} \\
\text{if } \text{CS} \cap \text{CN} = \emptyset \text{ return } \langle \text{CS}, \text{CN} \rangle \text{ else return false} \\
\end{align*}
\]
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  \( \emptyset, \{p(1)\}, \{p(2)\}, \{p(1), p(2)\} \)
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 $\emptyset, \{p(1)\}, \{p(2)\}, \{p(1), p(2)\}$

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

\[ :- p \]
\[ q :- p, \neg q \]
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
  - \(^\neg p(a)\) indicates that \(p(a)\) could be true or false

- Two negation operators can be related
  - \(-p : -^\neg p\)
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

Also known as:

Tuple generating dependencies (in relational databases)
Existential Rules

• In the context of description logic systems

   Person ⊓ (∃has_parent.Person)

Also known as:

   Existential rules
Problems with Existential Rules

• Termination

\[ \text{has\_parent}(X,f(X)) :- \text{instance\_of}(X,\text{person}) \]

Unrestricted application of this rule leads to infinite recursion.
Problems with Existential Rules

• Under-specification when used with a class hierarchy

\[
\text{has\_parent}(X,f(X)) \quad \text{-} \quad \text{instance\_of}(X,\text{person}) \\
\text{subclass\_of}(\text{male},\text{person}) \\
\text{has\_parent}(X,g(X)) \quad \text{-} \quad \text{instance\_of}(X,\text{male})
\]

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