Horn clauses and negations

An integrity constraint is a clause of the form

```
false :- a1,...,ak
```

where each `ai` is an atom and `false` is a special atom that is false in all interpretations.

Horn clauses and negations

An integrity constraint is a clause of the form

```
false :- a1,...,ak
```

where each `ai` is an atom and `false` is a special atom that is false in all interpretations.

A Horn clause is either a definite clause or an integrity constraint.

Horn clauses and negations

An integrity constraint is a clause of the form

```
false :- a1,...,ak
```

where each `ai` is an atom and `false` is a special atom that is false in all interpretations.

A Horn clause is either a definite clause or an integrity constraint.

The negation of a formula α, written $\neg\alpha$, is a formula that is true in an interpretation I iff α is false in I.

Horn clauses and negations

An integrity constraint is a clause of the form

```
false :- a1,...,ak
```

where each `ai` is an atom and `false` is a special atom that is false in all interpretations.

A Horn clause is either a definite clause or an integrity constraint.

The negation of a formula α, written $\neg\alpha$, is a formula that is true in an interpretation I iff α is false in I.

Example

$$KB = \begin{cases} \texttt{false :- a,b.} \\ \texttt{a :- c.} \\ \texttt{b :- c.} \end{cases}$$

$$KB \models \neg\texttt{c}$$

Disjunctions

Every set of definite clauses is satisfiable.
Not so with Horn clauses

$$KB \models \varphi \iff KB, \neg\varphi \text{ is not satisfiable}$$

Disjunctions

Every set of definite clauses is satisfiable.
Not so with Horn clauses

$$
\begin{aligned}
KB \models \varphi \quad &\iff \quad KB, \neg\varphi \text{ is not satisfiable} \\
&\iff \quad KB, \neg\varphi \models \texttt{false}.
\end{aligned}
$$

Disjunctions

Every set of definite clauses is satisfiable.
Not so with Horn clauses

$$\begin{aligned} KB \models \varphi &\iff KB, \neg\varphi \text{ is not satisfiable} \\ &\iff KB, \neg\varphi \models \texttt{false}. \end{aligned}$$

The disjunction $\alpha \vee \beta$ of α and β is a formula that is true in an interpretation I iff at least one of α or β is true in I.

Example

$$KB = \begin{cases} \texttt{false :- a,b.} \\ \texttt{a :- c.} \\ \texttt{b :- d.} \end{cases}$$

$$KB \models \neg\texttt{c} \vee \neg\texttt{d}$$

Disjunctions

Every set of definite clauses is satisfiable.
Not so with Horn clauses

$$\begin{aligned} KB \models \varphi \quad &\iff \quad KB, \neg\varphi \text{ is not satisfiable} \\ &\iff \quad KB, \neg\varphi \models \texttt{false}. \end{aligned}$$

The disjunction $\alpha \vee \beta$ of α and β is a formula that is true in an interpretation I iff at least one of α or β is true in I.

Example

$$KB = \begin{cases} \texttt{false :- a,b.} \\ \texttt{a :- c.} \\ \texttt{b :- d.} \end{cases}$$

$$KB \models \neg\texttt{c} \vee \neg\texttt{d}$$

Horn-SAT is feasible, whereas 3-SAT is likely not.

Non-monotonicity

Logical consequence is monotonic: adding clauses doesn't invalidate a previous conclusion

$$KB \models \varphi \text{ implies } KB, \psi \models \varphi.$$

Non-monotonicity

Logical consequence is monotonic: adding clauses doesn't invalidate a previous conclusion

$$KB \models \varphi \ \text{ implies } \ KB, \psi \models \varphi.$$

Negation-as-failure leads to non-monotonicity: a conclusion can be invalidated by adding more clauses.

```
empty-course(X) :- course(X),
                   \+enrolled(_,X).
```

Sometimes assume that a database of facts is complete.
Any fact not listed is false.

Non-monotonicity

Logical consequence is monotonic: adding clauses doesn't invalidate a previous conclusion

$$KB \models \varphi \text{ implies } KB, \psi \models \varphi.$$

Negation-as-failure leads to non-monotonicity: a conclusion can be invalidated by adding more clauses.

```
empty-course(X) :- course(X),
                   \+enrolled(_,X).
```

Sometimes assume that a database of facts is complete.
Any fact not listed is false.

Example: Assume a database of video segments is complete.

Rules

Encode *birds fly*

$$\texttt{fly}(X)\ \texttt{:-}\ \texttt{bird}(X).$$

to allow for exceptions.

Rules and defaults

Encode *birds fly*

$$\texttt{fly}(X) \ \texttt{:-}\ \texttt{bird}(X). \qquad\qquad \% \ \frac{\texttt{bird}(X)}{\texttt{fly}(X)}$$

to allow for exceptions.

Default rule (R. Reiter)

$$\frac{\texttt{bird}(X) \ : \ \texttt{fly}(X)}{\texttt{fly}(X)}$$

Rules and defaults

Encode *birds fly*

$$\texttt{fly}(X) \texttt{ :- } \texttt{bird}(X). \qquad \% \ \frac{\texttt{bird}(X)}{\texttt{fly}(X)}$$

to allow for exceptions.

Default rule (R. Reiter)

$$\frac{\texttt{bird}(X) \ : \ \texttt{fly}(X)}{\texttt{fly}(X)}$$

In general,

$$\frac{\text{prerequisite } p \ : \ \text{justification } j}{\text{conclusion } c}$$

applied to KB says:

$$\text{conclude } c \ \text{ if } \ KB \models p \text{ and } \underbrace{j \text{ is } KB\text{-consistent}}_{KB, j \not\models \texttt{false}}$$

Rules and defaults

Encode *birds fly*

$$\texttt{fly}(X) \;\texttt{:-}\; \texttt{bird}(X). \qquad\qquad \% \;\; \frac{\texttt{bird}(X)}{\texttt{fly}(X)}$$

to allow for exceptions.

Default rule (R. Reiter)

$$\frac{\texttt{bird}(X) \;:\; \texttt{fly}(X)}{\texttt{fly}(X)}$$

In general,

$$\frac{\text{prerequisite } p \;:\; \text{justification } j}{\text{conclusion } c}$$

applied to KB says:

$$\text{conclude } c \;\text{ if }\; KB \models p \text{ and } \underbrace{j \text{ is } KB\text{-consistent}}_{\substack{KB, j \not\models \texttt{false} \\ j \text{ is true in some model of } KB}}$$

Birds and bees

$$\frac{\texttt{bird}(X) \ : \ \texttt{fly}(X)}{\texttt{fly}(X)}$$

Let KB be

```
bird(robin).
bird(penguin).
false :- fly(penguin).
fly(bee).
```

Conclude:

Birds and bees

$$\color{red}{(\star)}\quad \frac{\texttt{bird}(X)\ :\ \texttt{fly}(X)}{\texttt{fly}(X)}$$

Let KB be

```
bird(robin).
bird(penguin).
false :- fly(penguin).
fly(bee).
```

Conclude:

`fly(robin)` by default rule $(\star)$

but *not* `fly(penguin)`.

Birds and bees

$$(\star) \quad \frac{\texttt{bird}(X) \ : \ \texttt{fly}(X)}{\texttt{fly}(X)}$$

Let KB be

```
bird(robin).
bird(penguin).
false :- fly(penguin).
fly(bee).
```

Conclude:

`fly(robin)` by default rule ($\star$)

but *not* `fly(penguin)`.

An explanation of `fly(bee)` using ($\star$) is

`bird(bee)`

Birds and bees

$$(\star)\quad \frac{\mathtt{bird}(X)\ :\ \mathtt{fly}(X)}{\mathtt{fly}(X)}$$

Let KB be

```
bird(robin).
bird(penguin).
false :- fly(penguin).
fly(bee).
```

Conclude:

`fly(robin)` by default rule $(\star)$

but *not* `fly(penguin)`.

An explanation of `fly(bee)` using $(\star)$ is

```
bird(bee)
```

which we can block by adding to KB the rule

```
false :- bird(bee).
```

Non-determinism

Conflicting defaults

$$\frac{\texttt{quaker}(X) : \texttt{pacifist}(X)}{\texttt{pacifist}(X)} \qquad \frac{\texttt{republican}(X) : \texttt{hawk}(X)}{\texttt{hawk}(X)}$$

Non-determinism

Conflicting defaults

$$\frac{\texttt{quaker}(X) : \texttt{pacifist}(X)}{\texttt{pacifist}(X)} \qquad \frac{\texttt{republican}(X) : \texttt{hawk}(X)}{\texttt{hawk}(X)}$$

Let *KB* be

```
quaker(nixon).
republican(nixon).
false :- pacifist(X), hawk(X).
```

Non-determinism

Conflicting defaults

$$\frac{\texttt{quaker}(X) : \texttt{pacifist}(X)}{\texttt{pacifist}(X)} \qquad \frac{\texttt{republican}(X) : \texttt{hawk}(X)}{\texttt{hawk}(X)}$$

Let KB be

```
quaker(nixon).
republican(nixon).
false :- pacifist(X), hawk(X).
```

Applying one default to Nixon makes the other inapplicable.

KB has two incompatible extensions, breaking

least fixed point (provability model) for Horn clauses.

Normal default rules and inferring negations

A default rule is *normal* if its justification is its conclusion $\quad \dfrac{p : c}{c}$

— infer c if it is consistent and p is provable

Normal default rules and inferring negations

A default rule is *normal* if its justification is its conclusion $\frac{p : c}{c}$

– infer c if it is consistent and p is provable

Closed World Assumption: any unprovable atom φ is false

$$\frac{\texttt{true} : \neg\varphi}{\neg\varphi}$$

Normal default rules and inferring negations

A default rule is *normal* if its justification is its conclusion $\dfrac{p : c}{c}$

– infer c if it is consistent and p is provable

Closed World Assumption: any unprovable atom φ is false

$$\frac{\mathtt{true} : \neg\varphi}{\neg\varphi}$$

Negation as failure: φ is false if attempting to prove φ fails finitely

```
naf(P) :- (P,!,fail); true.
```

Normal default rules and inferring negations

A default rule is *normal* if its justification is its conclusion $\dfrac{p : c}{c}$

– infer c if it is consistent and p is provable

Closed World Assumption: any unprovable atom φ is false

$$\frac{\texttt{true} : \neg\varphi}{\neg\varphi}$$

Negation as failure: φ is false if attempting to prove φ fails finitely

```
naf(P) :- (P,!,fail); true.
```

N.B. Checking finite failure can be as hard as the Halting Problem.

3 modes of inference (C.S. Peirce)

Deduction	deduce	modus ponens $\cong$ function app $f(a)$
Abduction	explain	choose input a from *assumables*
Induction	generalise/program	choose rule/function f

3 modes of inference (C.S. Peirce)

Deduction	deduce	modus ponens $\cong$ function app $f(a)$
Abduction	explain	choose input a from *assumables*
Induction	generalise/program	choose rule/function f

From $\models$ as inclusion $\subseteq$

$$\begin{aligned} KB \models g &\iff Mod(KB) \subseteq Mod(g) \\ KB \text{ satisfiable} &\iff Mod(KB) \not\subseteq Mod(\texttt{false}) \\ &\iff Mod(KB) \neq \emptyset \end{aligned}$$

to weighing alternatives $d \in D$ via probabilities given KB

$$prob(d|KB) \ = \ \text{conditional probability of } d \text{ given } KB$$

$\leadsto$ Bayesian networks, from next week on.