Declarative semantics for incomplete information: *completing incomplete programs*

semantics and proof theory for the not in a general clause will be discussed ~~later~~ NOW

problem

can no longer express

```
married(X); bachelor(X) :- man(X), adult(X).
man(john). adult(john).
```

characteristic of indefinite clauses

which had two minimal models

```
{man(john),adult(john),married(john)}
{man(john),adult(john),bachelor(john)}
{man(john),adult(john),married(john),bachelor(john)}
```

definite clause containing not

general clauses

first model is minimal model of **general** clause

```
married(X) :- man(X), adult(X), not bachelor(X).
```

second model is minimal model of **general** clause

```
bachelor(X) :- man(X), adult(X), not married(X).
```

to prove that someone is a bachelor, prove that he is a man and an adult, and prove that he is not a bachelor

Declarative semantics for incomplete information: *completing incomplete programs*

A program P is "complete" if for every (ground) fact f, either $P \models f$ or $P \models \neg f$

unique minimal model

Transform an incomplete program into a complete one, that captures the intended meaning of the original program.

possible transformations

closed world assumption	predicate completion
straightforward	ok for general clauses (with negation in body)
ok for definite clauses (without negation)	may lead to inconsistencies if the program is not stratified

Completing incomplete programs: *closed world assumption*

everything that is not known to be true, must be false

motivation: in general, there are more false statements that can be made than true statements

do not say something is not true, simply say nothing about it

Completing incomplete programs: *closed world assumption*

everything that is not known to be true, must be false

$CWA(P) = P \cup \{:\!-A \mid A \in B_P \wedge P \not\models A\}$

the clause "false :-A" is only true under interpretations in which A is false

CWA-complement of a program P (i.e, CWA(P)-P): explicitly assume that every ground atom A that does not follow from P is false

Completing incomplete programs:
closed world assumption - example

P
```
likes(peter,S) :- student_of(S,peter).
student_of(paul,peter).
```

only the black atoms are relevant for determining whether an interpretation is a model of every ground instance of every clause

B_P
```
{likes(peter,peter),likes(peter,paul),
 likes(paul,peter),likes(paul,paul),
 student_of(peter,peter),student_of(peter,paul),
 student_of(paul,peter),student_of(paul,paul)}
```

models
```
{student_of(paul,peter),likes(peter,paul)}
{student_of(paul,peter),likes(peter,paul),likes(peter,peter)}
{student_of(paul,peter),likes(peter,paul),
 student_of(peter,peter),likes(peter,peter)}
...
```

there are still 4 orange atoms remaining which can each be added (or not) freely to the above interpretations

in total: 3*2^4=48 models for such a simple program!

P ⊨ A
```
likes(peter,paul)
student_of(paul,peter)
```

Completing incomplete programs: *closed world assumption - example*

P
```
likes(peter,S) :- student_of(S,peter).
student_of(paul,peter).
```

B_P
```
{likes(peter,peter),likes(peter,paul),
 likes(paul,peter),likes(paul,paul),
 student_of(peter,peter),student_of(peter,paul),
 student_of(paul,peter),student_of(paul,paul)}
```

$P \models A$
```
likes(peter,paul)
student_of(paul,peter)
```

CWA(P)
```
likes(peter,S) :- student_of(S,peter).
student_of(paul,peter).
:- student(paul,paul).
:- student(peter,paul).
:- student(peter,peter).
:- likes(paul,paul).
:- likes(paul,peter).
:- likes(peter,peter).
```

is a complete program: every ground atom from B_P is assigned true or false

has only 1 model: {student_of(paul,peter),likes(peter,paul)} which is declared the intended model of the program (also obtained as the intersection of all models)

Completing incomplete programs: *closed world assumption - inconsistency*

when applied to indefinite and general clauses

P
```
bird(tweety).
flies(X);abnormal(X) :- bird(X).
```

B_P
```
{bird(tweety),abnormal(tweety),flies(tweety)}
```

models
```
{bird(tweety),flies(tweety)}
{bird(tweety),abnormal(tweety)}
{bird(tweety),abnormal(tweety),flies(tweety)}
```

$P \models A$
```
bird(tweety)
```

CWA(P)
```
bird(tweety).
flies(X);abnormal(X) :- bird(X).
:-abnormal(tweety).
:-flies(tweety)
```

CWA(P) is inconsistent

no longer has a model because, in order for the second clause to be true under an interpretation, its head needs to be true given that its body is already true due to the first clause

Completing incomplete programs: *predicate completion - idea*

turn implications (*if*) into equivalences (*iff*) by completing clauses (with their and-only-if part)

regard each clause as part of the complete definition of a predicate

only clause defining likes/2:

P
```
likes(peter,S) :- student(S,peter).
```

its completion:

$$\forall X \forall S\ likes(X,S) \leftrightarrow X = peter \wedge student(S,peter)$$

in clausal form:

Comp(P)
```
likes(peter,S) :- student(S,peter).
X=peter :- likes(X,S).
student(S,peter) :- likes(X,S)
```

Completing incomplete programs: *predicate completion - algorithm*

```
likes(peter,S) :- student_of(S,peter).
student_of(paul,peter).
```

1. ensure each argument of each clause head is a distinct variable

> add literals Var=Term to body

```
likes(X,S) :- X=peter,student_of(S,peter).
student_of(X,Y) :- X=paul,Y=peter
```

2. if there are several clauses for a predicate, combine them into a single formula

> use disjunction in implication's body if there are multiple clauses for a predicate

$\forall X \forall Y$ likes(X,Y)$\leftarrow$ X=peter$\wedge$student_of(Y,peter))

$\forall X \forall Y$ student_of(X,Y)$\leftarrow$ X=paul$\wedge$Y=peter

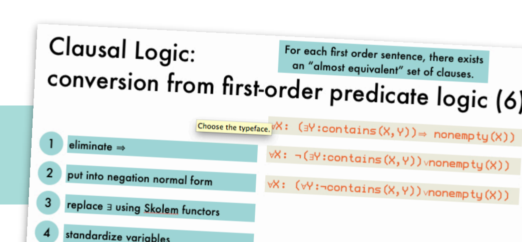
> if a predicate without definition is used in a body (e.g. p/1), add $\forall X\ \neg p(X)$

3. turn the implication into an equivalence

$\forall X \forall Y$ likes(X,Y)$\leftrightarrow$ X=peter$\wedge$student_of(Y,peter))

$\forall X \forall Y$ student_of(X,Y) $\leftrightarrow$ X=paul$\wedge$Y=peter

4. convert to clausal form



Completing incomplete programs: *predicate completion - algorithm*

```
likes(peter,S) :- student_of(S,peter).
student_of(paul,peter).
```

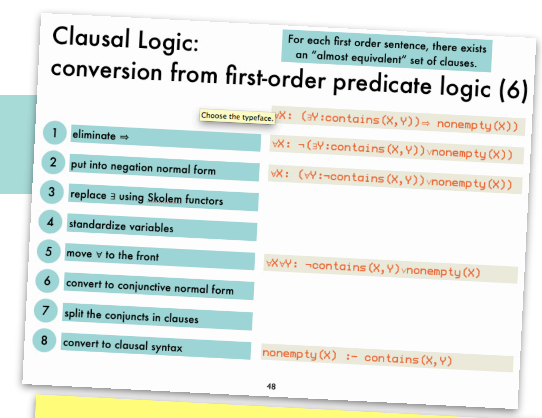
if a predicate without definition is used in a body (e.g. p/1), add ∀X ¬p(X)

3 turn the implication into an equivalence

$\forall X \forall Y$ likes(X,Y) $\leftrightarrow$ X=peter $\wedge$ student_of(Y,peter))

$\forall X \forall Y$ student_of(X,Y) $\leftrightarrow$ X=paul $\wedge$ Y=peter

4 convert to clausal form




```
likes(peter,S):-student_of(S,peter).
X=peter:-likes(X,S).
student_of(S,peter):-likes(X,S).
student_of(paul,peter).
X=paul:-student_of(X,Y).
Y=peter:-student_of(X,Y).
```

for definite clauses, CWA(P) and Comp(P) have same model

has the single model {student_of(paul,peter), likes(peter,paul)}

Completing incomplete programs:
predicate completion - existential variables

if a predicate without definition is used in a body (e.g. p/1), add $\forall X\ \neg p(X)$

3 turn the implication into an equivalence

careful with variables in a body that do not occur in the head

```
ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-parent(X,Z), ancestor(Z,Y).
```

$$\forall X \forall Y\ ancestor(X,Y) \leftrightarrow (parent(X,Y) \vee (\exists Z\ parent(X,Z) \wedge ancestor(Z,Y))))$$

$$\forall X \forall Y \forall Z\ ancestor(X,Y) \leftarrow parent(X,Z) \wedge ancestor(Z,Y)$$

$$\forall X \forall Y\ ancestor(X,Y) \leftarrow \exists Z\ parent(X,Z) \wedge ancestor(Z,Y))$$

use second form because all clauses must have the same head

$\forall Z{:}q \leftarrow p(Z)$
$\forall Z{:}q \vee \neg p(Z)$
$q \vee \forall Z{:}\neg p(Z)$
$q \vee \exists Z{:}p(Z)$

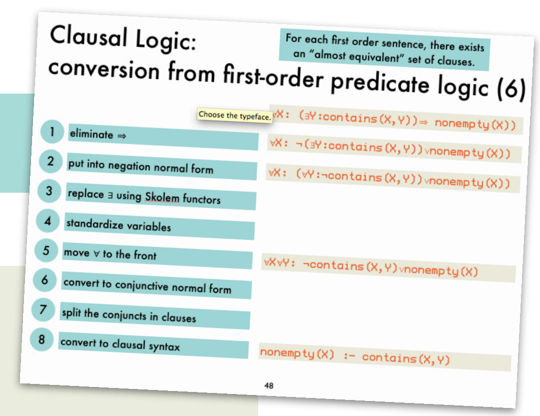
Completing incomplete programs:
predicate completion - existential variables

3 turn the implication into an equivalence

$$\forall X \forall Y \; ancestor(X,Y) \leftrightarrow (parent(X,Y) \lor (\exists Z \; parent(X,Z) \land ancestor(Z,Y)))$$

4 convert to clausal form

```
ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).
parent(X,Y);parent(X,pa(X,Y)):-ancestor(X,Y).
parent(X,Y);ancestor(pa(X,Y),Y):-ancestor(X,Y).
```



Skolem functor

$\forall X \exists Y$: loves(X,Y)

$\forall X$:loves(X,person_loved_by(X))

Completing incomplete programs: *predicate completion - negation*

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

1. ensure each argument of each clause head is a distinct variable

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

2. if there are several clauses for a predicate, combine them into a single formula

$\forall X\ bird(X) \leftarrow X=tweety.$

$\forall X\ flies(X) \leftarrow bird(X) \land \neg abnormal(X)$

3. turn the implication into an equivalence

$\forall X\ bird(X) \leftrightarrow X=tweety.$

$\forall X\ flies(X) \leftrightarrow bird(X) \land \neg abnormal(X).$

$\forall X\ \neg abnormal(X)$

if a predicate without definition is used in a body (e.g. p/1), add $\forall X\ \neg p(X)$

Completing incomplete programs: *predicate completion - negation*

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

if a predicate without definition is used in a body (e.g. p/1), add ∀X ¬p(X)

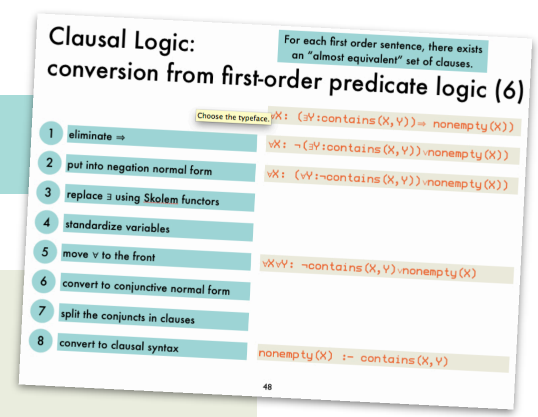
3 turn the implication into an equivalence

∀X bird(X) ↔ X=tweety.

∀X flies(X) ↔ bird(X)∧¬abnormal(X).

∀X ¬abnormal(X)

4 convert to clausal form




```
bird(tweety).
X=tweety:-bird(X).
flies(X);abnormal(X):-bird(X).
bird(X):-flies(X).
:-flies(X),abnormal(X).
:-abnormal(X).
```

has the single model {bird(tweety),flies(tweety)}

Completing incomplete programs: *predicate completion - inconsistency*

Comp(P) is inconsistent for certain **unstratified** P

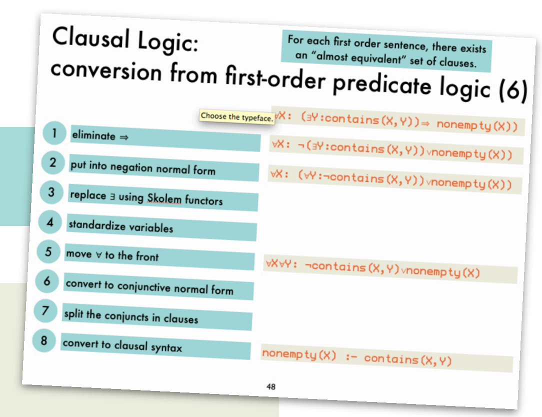
```
wise(X):-not(teacher(X)).
teacher(peter):-wise(peter).
```

if a predicate without definition is used in a body (e.g. p/1), add $\forall X\ \neg p(X)$

3 turn the implication into an equivalence

$\forall X\ wise(X) \leftrightarrow \neg teacher(X)$

$\forall X\ teacher(X) \leftrightarrow X = peter \wedge wise(peter)$



4 convert to clausal form

```
wise(X);teacher(X).
:-wise(X),teacher(X).
teacher(peter):-wise(peter).
X=peter:-teacher(X).
wise(peter):-teacher(X).
```

inconsistent!

Completing incomplete programs: *stratified programs*

if P is stratified then Comp(P) is consistent

sufficient but not necessary: there are non-stratified P's for which Comp(P) is consistent

organize the program in layers (strata);
do not allow the programmer to negate a predicate
that is not yet completely defined (in a lower stratum)

A program P is stratified if its predicate symbols can be partitioned into disjoint sets $S_0, \ldots, S_n$
such that for each clause $p(\ldots) \leftarrow L_1,\ldots,L_j$ where $p \in S_k$, any literal L_j is such that

if $L_j = q(\ldots)$ then $q \in S_0 \cup \ldots \cup S_k$

if $L_j = \neg q(\ldots)$ then $q \in S_0 \cup \ldots \cup S_{k-1}$

Completing incomplete programs: *soundness result for SLDNF-resolution*

$P \vdash_{SLDNF} q \Rightarrow Comp(P) \models q$

completeness result only holds for a subclass of programs