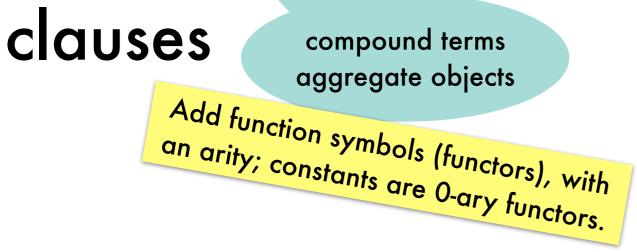
Full Clausal Logic - Syntax:



| | functor | : | single word starting with lower case |
|-------------|-----------|---|---|
| object | variable | • | single word starting with upper case |
| | term | • | <pre>variable functor[(term[,term]*)]</pre> |
| | predicate | • | single word starting with lower case |
| | | | <pre>predicate[(term[,term]*])]</pre> |
| proposition | clause | • | head [:- body] |
| | head | • | [atom[;atom]*] |
| | body | • | <pre>proposition[,proposition]*</pre> |

"adding two Peanoencoded naturals"

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).

Full Clausal Logic - Semantics: relational clausal logic Herbrand universe, base, interpretation

Herbrand universe of a program P

{ 0, s(0), s(s(0)), s(s(s(0))),... }

terms that can be constructed from the constants and functors

Herbrand base B_P of a program P

{ plus(0,0,0), plus(s(0),0,0),
 plus(0,s(0),0), plus(s(0),s(0),0),...}

set of all ground atoms that can be constructed using predicates in P and ground terms in the Herbrand universe of P

Herbrand interpretation I of P

is this a model?

infinite!

{ plus(0,0,0), plus(s(0),0,s(0)),plus(0,s(0),s(0)) }

possibly infinite subset of B_P consisting of ground atoms that are true

Full Clausal Logic - Semantics: infinite models are possible

Herbrand universe is infinite, therefore infinite number of grounding substitutions

An interpretation is a **model for a program** if it is a model for each ground instance of every clause in the program.

```
plus(0,0,0)
plus(s(0),0,s(0)):-plus(0,0,0)
plus(s(s(0)),0,s(s(0))):-plus(s(0),0,s(0))
...
plus(0,s(0),s(0))
plus(s(0),s(0),s(s(0))):-plus(0,s(0),s(s(0)))
plus(s(s(0)),s(0),s(s(s(0))):-plus(s(0),s(0),s(s(0)))
...
```

according to first ground clause, plus(0,0,0) has to be in any model but then the second clause requires the same of plus(s(0),0,s(0)) and the third clause of plus(s(s(0)),0,s(s(0))) ...

> all models of this program are necessarily infinite

Full Clausal Logic - Proof Theory: < computing the most general unifier

atoms

plus(s(0),X,s(X)) and plus(s(Y),s(0),s(s(Y)))

have most general unifier

 $\{Y/0, X/s(0)\}$

yields unified atom
plus(s(Y),s(0),s(s(Y)))

found by

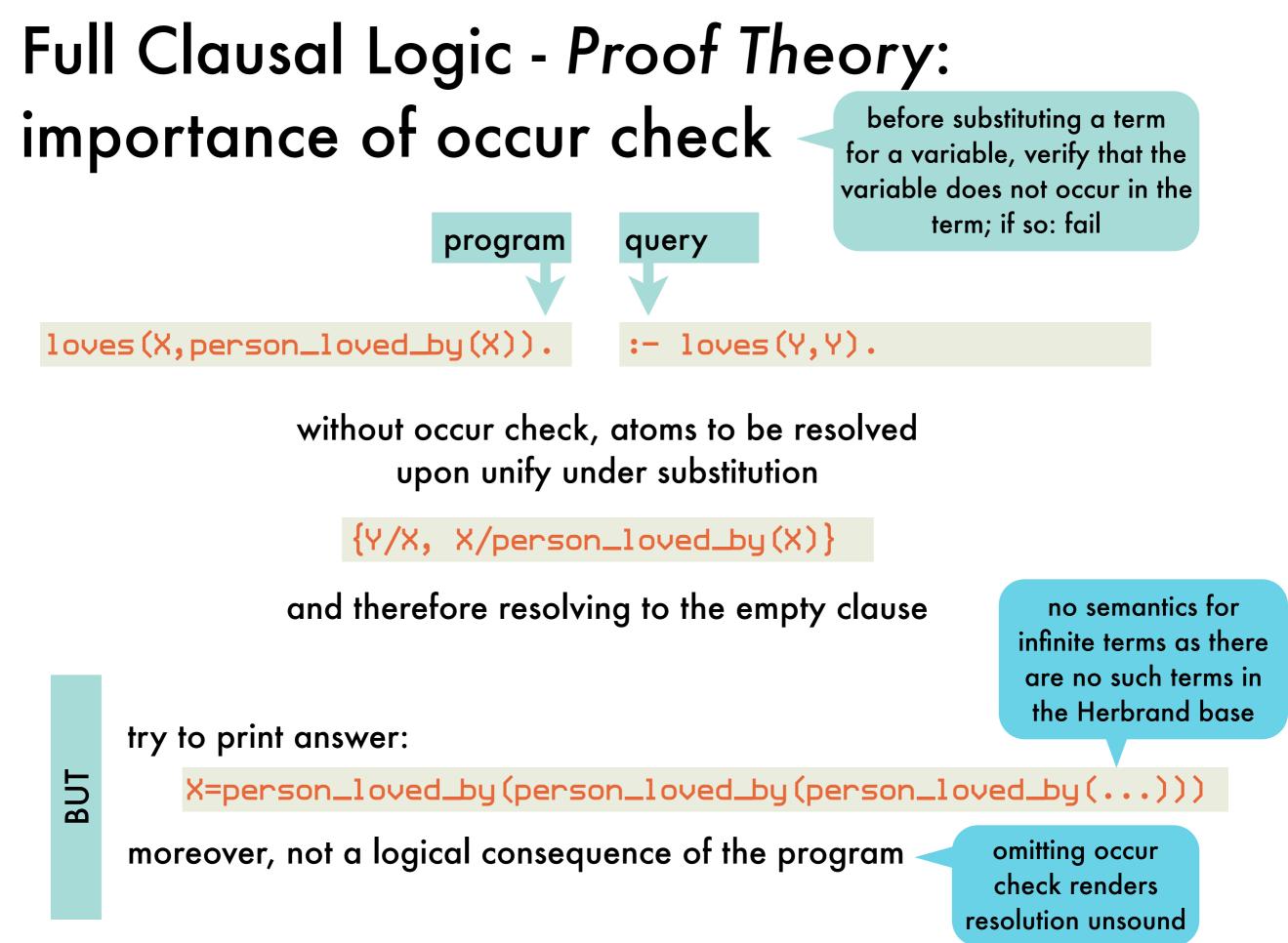
renaming variables so that the two atoms have none in common ensuring that the atoms' predicates and arity correspond scanning the subterms from left to right to find first pair of subterms where the two atoms differ; if neither subterm is a variable, unification fails; else substitute the other term for all occurrences of the variable and remember the partial substitution; repeat until no more differences found

analogous to relational clausal logic, but have to take compound terms into acount when computing the mgu of complementary atoms

Full Clausal Logic - Proof Theory: computing the most general unifier using the Martelli-Montanari algorithm

repeat operates on a finite set of equations s=t select $s = t \in \mathcal{E}$ case s = t of $f(s_1,...,s_n) = f(t_1,...,t_n) \ (n \ge 0)$: replace s = t by $\{s_1 = t_1, ..., s_n = t_n\}$ $f(s_1,\ldots,s_m)=g(t_1,\ldots,t_n) \ (f/m\neq g/n):$ fail X = Xremove X = X from \mathcal{E} t = X ($t \notin Var$): replace t = X by X = t $X = t \ (X \in Var \land X \neq t \land X \text{ occurs more than once in } \mathcal{E})$: if Xoccurs in t occur check then fail else replace all occurrences of X in \mathcal{E} (except in X = t) by t esac until no change

 ${f(X, g(Y)) = f(g(Z), Z)}$ $\Rightarrow \{X = g(Z), g(Y) = Z\}$ $\Rightarrow \{X = g(Z), Z = g(Y)\}$ $\Rightarrow \{X = g(g(Y)), Z = g(Y)\}$ $\Rightarrow \{X/g(g(Y)), Z/g(Y)\}$ resulting set = mgu ${f(X, g(X), b) = f(a, g(Z), Z)}$ $\Rightarrow \{X = a, g(X) = g(Z), b = Z\}$ $\Rightarrow \{\underline{X = a}, X = Z, b = Z\}$ $\Rightarrow \{X = a, \underline{a = Z}, b = Z\}$ $\Rightarrow \{X = a, \underline{Z} = a, b = Z\}$ $\Rightarrow \{X = a, Z = a, \underline{b} = a\}$ fail \Rightarrow



Full Clausal Logic - Proof Theory: occur check not performed in Prolog out of

performance considerations (e.g. unify X with a list of 1000 elements)

Martelli-Montanari algorithm

$$\{ \frac{I(Y, Y) = I(X, f(X))}{Y = X, Y = f(X)} \}$$

$$\Rightarrow \{ Y = X, \underline{X = f(X)} \}$$

$$\Rightarrow fail$$

SWI-Prolog

Full Clausal Logic - Meta-theory: soundness, completeness, decidability

full clausal logic is sound

full clausal

$$P \vdash C \Rightarrow P \models C$$

complete

full clausal logic is refutation-complete $P \cup \{C\}$ inconsistent $\Rightarrow P \cup \{C\} \vdash \Box$

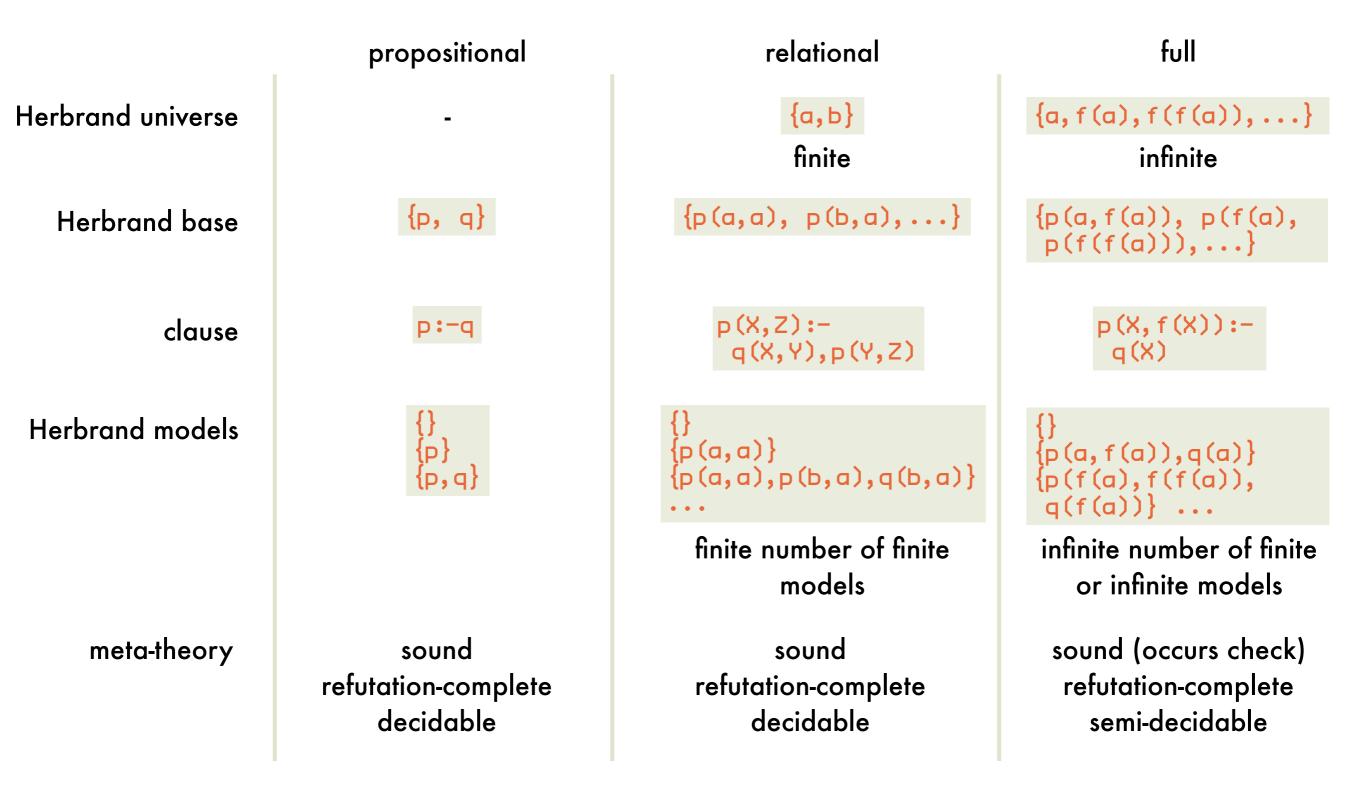
decidability

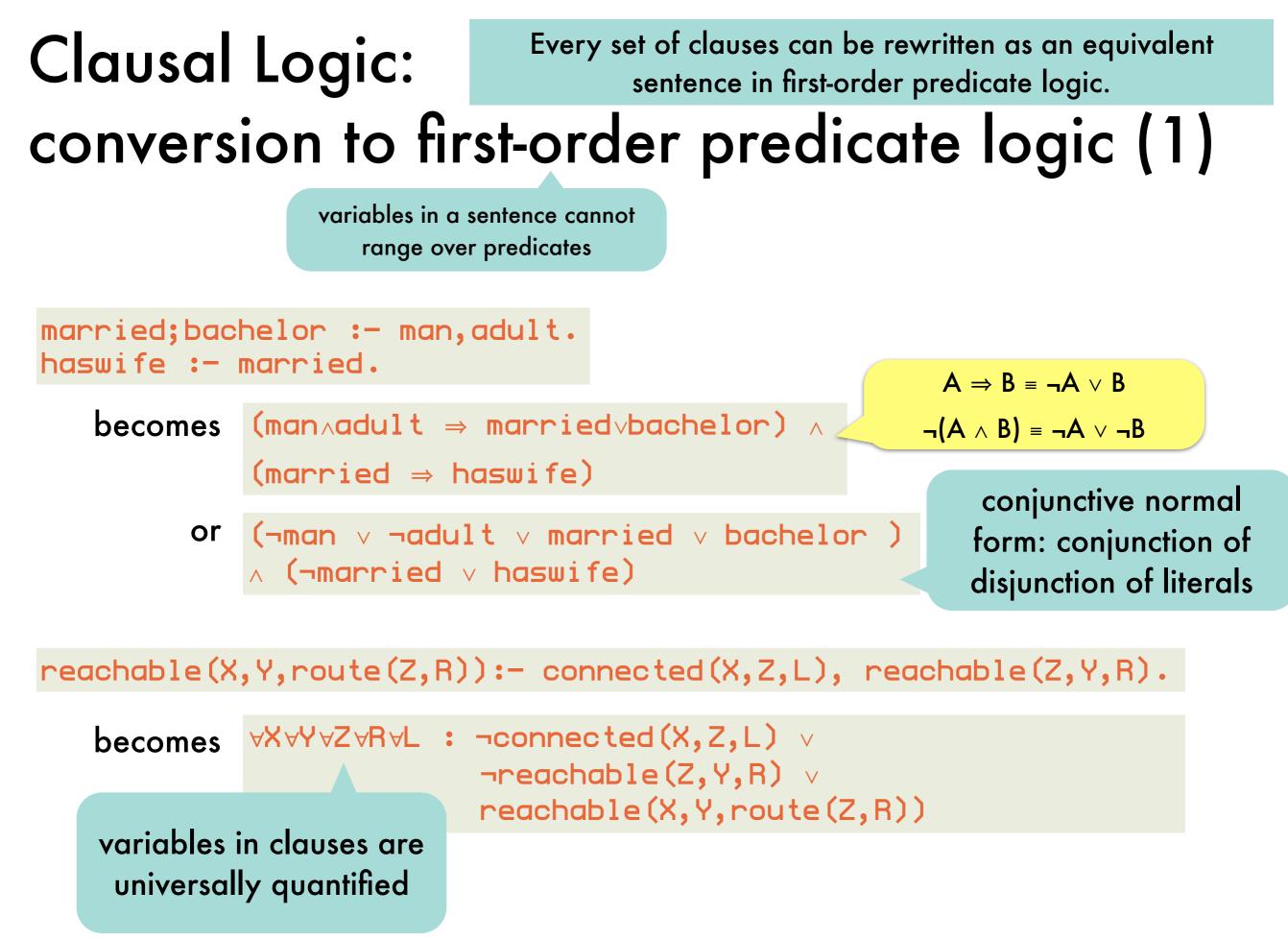
The question "P⊧C?" is only semi-decidable.

there is no algorithm that will always answer the question (with "yes" or "no") in finite time; but there is an algorithm that, if $P \models C$,

will answer "yes" in finite time but this algorithm may loop if $P \not\models C$.

Clausal Logic: overview





Clausal Logic: Every set of clauses can be rewritten as an equivalent sentence in first-order predicate logic. Conversion to first-order predicate logic (2)

nonempty(X) :- contains(X,Y).

| <pre>becomes VXVY: nonempty(X)v-contains(X,Y)</pre> |
|---|
|---|

| or | ∀X: (| (nonempty(X)vyY¬contains(X,Y)) | |
|----|-------|--------------------------------|--|
|----|-------|--------------------------------|--|

| or | <pre>∀X: nonempty(X) v¬(∃Y:contains(X,Y))</pre> |
|----|---|
|----|---|

or ∀X: (∃Y:contains(X,Y))⇒ nonempty(X))

variables that occur only in the body of a clause are existentially qualified

Clausal Logic: an "almost equivalent" set of clauses. Conversion from first-order predicate logic (1)

∀X [brick(X) ⇒ (∃Y [on(X,Y) ^¬pyramid(Y)] ^ ¬∃Y [on(X,Y) ^ on(Y,X)] ^ ∀Y [¬brick(Y) ⇒¬equal(X,Y)])]

eliminate \Rightarrow using $A \Rightarrow B = \neg A \lor B$.

2

put into negation normal form: negation only occurs immediately before propositions

∀X [¬brick(X)∨(∃Y [on(X,Y)∧¬pyramid(Y)]∧ ∀Y [¬on(X,Y)∨¬on(Y,X)]∧ ∀Y [brick(Y)∨¬equal(X,Y)])]

 $\neg(A \land B) \equiv \neg A \lor \neg B$ $\neg(A \lor B) \equiv \neg A \land \neg B$ $\neg(\neg A) \equiv A$ $\neg \forall X [p(X)] \equiv \exists X [\neg p(X)]$ $\neg(\exists X [p(X)] \equiv \forall X [\neg p(X)]$

For each first order sentence, there exists an "almost equivalent" set of clauses.

∃X∀Y: loves(X,Y)

Skolem constants substitute for an

of a universal quantifier

conversion from first-order predicate logic (2)

 $\forall X [\neg brick(X) \lor (\exists Y [on(X,Y) \land \neg pyramid(Y)] \land$ $\forall Y [\neg on(X,Y) \lor \neg on(Y,X)]_{\wedge}$ \fystyle \fystyl

Clausal Logic:

model {loves[paul,anna]]

3

existentially quantified variable which does not occur in the scope $\forall X \exists Y : loves(X,Y)$ ∀X:loves(X,person_loved_by(X))

can be converted to equivalent {loves[paul,person_loved_by[paul]]] replace existentially quantified variable by a compound term of which the arguments are the universally quantified variables in whose scope the existentially quantified variable occurs

replace \exists using Skolem functors (abstract names for objects, functor has to be new)

 $\forall X [\neg brick(X) \lor ([on(X, sup(X)) \land \neg pyramid(sup(X))] \land$ $\forall Y [\neg on(X,Y) \lor \neg on(Y,X)] \land$ \fystyle \fystyl

Clausal Logic: an "almost equivalent" set of clauses. Conversion from first-order predicate logic (3)

∀X [¬brick(X)∨([on(X, sup(X))∧¬pyramid(sup(X))]∧ ∀Y [¬on(X,Y)∨¬on(Y,X)]∧ ∀Y [brick(Y)∨¬equal(X,Y)])]

standardize all variables apart such that each quantifier has its own unique variable

∀X [¬brick(X)∨([on(X,sup(X))∧¬pyramid(sup(X))]∧ ∀Y [¬on(X,Y)∨¬on(Y,X)]∧ ∀Z [brick(Z)∨¬equal(X,Z)])]

move ∀ to the front

5

\delta \del

Clausal Logic: an "almost equivalent" set of clauses. Conversion from first-order predicate logic (4)

convert to conjunctive normal form using $A \lor (B \land C) = (A \lor B) \land (A \lor C)$

\X\Y\Z[(\-brick(X)\[on(X,sup(X))\-pyramid(sup(X))])\ (\-brick(X)\[-on(X,Y)\-on(Y,X)])\ (\-brick(X)\[brick(Z)\-equal(X,Z)])]

\delta \del

Clausal Logic: an "almost equivalent" set of clauses. Conversion from first-order predicate logic (5)

\delta \del

split the conjuncts in clauses (a disjunction of literals)

```
\forall \Lambda \
```

8

convert to clausal syntax (negative literals to body, positive ones to head)

```
on(X,sup(X)) :- brick(X).
:- brick(X), pyramid(sup(X)).
:- brick(X), on(X,Y), on(Y,X).
brick(X) :- brick(Z), equal(X,Z).
```

Clausal Logic: an "almost equivalent" set of clauses. Conversion from first-order predicate logic (6)

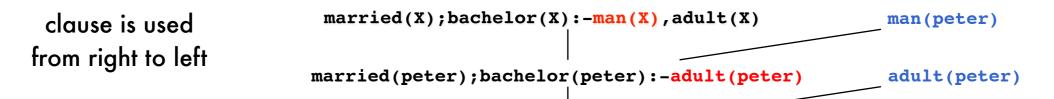
 $\forall X: (\exists Y:contains(X,Y)) \Rightarrow nonempty(X))$

| 1 | eliminate \Rightarrow | ∀X: ¬(∃Y:contains(X,Y))∨nonempty(X)) |
|---|---|--|
| 2 | put into negation normal form | ∀X: (∀Y:¬contains(X,Y))∨nonempty(X)) |
| 3 | replace \exists using Skolem functors | |
| 4 | standardize variables | |
| 5 | move ∀ to the front | <pre>∀X∀Y: ¬contains(X,Y) vnonempty(X)</pre> |
| 6 | convert to conjunctive normal form | |
| 7 | split the conjuncts in clauses | |
| 8 | convert to clausal syntax | <pre>nonempty(X) :- contains(X,Y)</pre> |

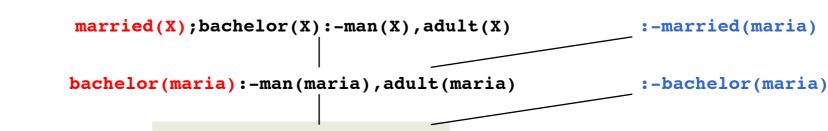
Definite Clause Logic: motivation



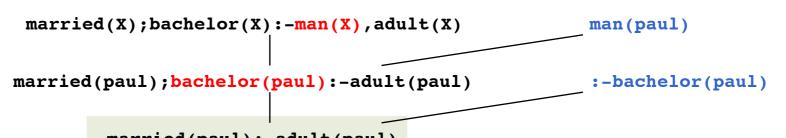
how to use the clause depends on what you Now to use the clause depends on what to prove, but this indeterminacy is a contract of the clause o wom to prove, but mis inverentiation of inefficiency in refutation proofs married(X);bachelor(X) :- man(X), adult(X). man(peter). adult(peter). man(paul). :-married(maria). :-bachelor(maria). :-bachelor(paul).



married(peter);bachelor(peter)



:-man(maria),adult(maria)



both literals from head and body are resolved away

clause is used

from left to right

married(paul):-adult(paul)

can be derived in two resolution steps logical consequences that

indefinite

conclusion

Definite Clause Logic: syntax and proof procedure

for efficiency's sake

rules out indefinite conclusions

full clausal logic clauses are restricted: at most one atom in the head

 $A := B_1, ..., B_n$

fixes direction to use clauses

from right to left: procedural interpretation

"prove A by proving each of B_i "

Definite Clause Logic: recovering lost expressivity

semantics and proof theory for the not in a general clause will be discussed later; Prolog actually provides a special predicate not/1 which can only be understood procedurally

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

```
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

problem