Full Clausal Logic - Syntax:

clauses

cr

cn
compund terms
aggregate objects

Add function symbols (functors), with an arity; constants are 0-ary functors.

functor : single word starting with lower case
variable : single word starting with upper case
term : variable | functor[((term [,term])*]]
predicate : single word starting with lower case
atom : predicate[((term [,term])*]]
clause : head [:- body]
head : [atom [,atom]*]
body : proposition [,proposition]*

“adding two Peano-encoded naturals”

plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
Full Clausal Logic - Semantics:
Herbrand universe, base, interpretation

**Herbrand universe** of a program $P$

\[
\{ 0, s(0), s(s(0)), s(s(s(0))), \ldots \}
\]

terms that can be constructed from the constants and functors

**Herbrand base** $B_P$ of a program $P$

\[
\{ \text{plus}(0,0,0), \text{plus}(s(0),0,0), \\
\text{plus}(0,s(0),0), \text{plus}(s(0),s(0),0), \ldots \}
\]

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 $P$

\[
\{ \text{plus}(0,0,0), \text{plus}(s(0),0,s(0)), \text{plus}(0,s(0),s(0)) \}
\]

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

analogous to relational clausal logic

is this a model?

infinite!
Full Clausal Logic - Semantics: infinite models are possible

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

\[
\begin{align*}
\text{plus}(0,0,0) \\
\text{plus}(s(0),0,s(0)) &: \neg \text{plus}(0,0,0) \\
\text{plus}(s(s(0)),0,s(s(0))) &: \neg \text{plus}(s(0),0,s(0)) \\
\ldots \\
\text{plus}(0,s(0),s(0)) \\
\text{plus}(s(0),s(0),s(s(0))) &: \neg \text{plus}(0,s(0),s(s(0))) \\
\text{plus}(s(s(0)),s(s(0)),s(s(s(0)))) &: \neg \text{plus}(s(0),s(s(0)),s(s(s(0)))) \\
\ldots
\end{align*}
\]

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

all models of this program are necessarily infinite
Full Clausal Logic - *Proof Theory*: computing the most general unifier

atoms

\[
\text{\texttt{\textit{plus}}(s(0), X, s(X)) \text{ and } \text{\texttt{\textit{plus}}}(s(Y), s(0), s(s(Y)))}
\]

have most general unifier

\[
\{ Y/0, X/s(0) \}
\]

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

```plaintext
repeat
  select $s = t \in \mathcal{E}$
  case $s = t$ of
    $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ ($n \geq 0$):
      replace $s = t$ by \{ $s_1 = t_1, \ldots, s_n = t_n$ \}
    $f(s_1, \ldots, s_m) = g(t_1, \ldots, t_n)$ ($f/m \neq g/n$):
      fail
      $X = X$:
      remove $X = X$ from $\mathcal{E}$
    $t = X$ ($t \not\in \text{Var}$):
      replace $t = X$ by $X = t$
    $X = t$ ($X \in \text{Var}$ \& $X \neq t$ \& $X$ occurs more than once in $\mathcal{E}$):
      if $X$ occurs in $t$
        then fail
      else replace all occurrences of $X$ in $\mathcal{E}$ (except in $X = t$) by $t$
  esac
until no change
```

Resulting set = mgu

Examples:

\[
\frac{f(X, g(Y)) = f(g(Z), Z)}{X = g(Z), g(Y) = Z}
\]

\[
\frac{X = g(Z), Z = g(Y)}{X = g(g(Y)), Z = g(Y)}
\]

\[
\frac{f(X, g(X), b) = f(a, g(Z), Z)}{X = a, g(X) = g(Z), b = Z}
\]

\[
\frac{X = a, X = Z, b = Z}{X = a, a = Z, b = Z}
\]

\[
\frac{X = a, Z = a, b = Z}{X = a, Z = a, b = a}
\]

\[
\Rightarrow \text{fail}
\]
Full Clausal Logic - Proof Theory: importance of occur check

Before substituting a term for a variable, verify that the variable does not occur in the term; if so: fail

Program:

\[ \text{loves}(X, \text{person}_\text{loved}_\text{by}(X)). \]

Query:

\[ \text{:- loves}(Y, Y). \]

Without occur check, atoms to be resolved upon unify under substitution

\[ \{Y/X, \, X/\text{person}_\text{loved}_\text{by}(X)\} \]

And therefore resolving to the empty clause

Try to print answer:

\[ X=\text{person}_\text{loved}_\text{by}(\text{person}_\text{loved}_\text{by}(\text{person}_\text{loved}_\text{by}(\ldots)))) \]

Moreover, not a logical consequence of the program

But

Omitting occur check renders resolution unsound

No semantics for infinite terms as there are no such terms in the Herbrand base.
**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**

\[
\{ l(Y, Y) = l(X, f(X)) \} \\
\Rightarrow \{ Y = X, Y = f(X) \} \\
\Rightarrow \{ Y = X, X = f(X) \} \\
\Rightarrow \text{fail}
\]

---

**SWI-Prolog**

?- \(l(Y,Y) = l(X,f(X)).\)
\(Y = f(**),\)
\(X = f(**).\)

?- unify_with_occurs_check(l(Y,Y),l(X,f(X))).
false.

in rare cases where the occurs check is needed
Full Clausal Logic - Meta-theory: soundness, completeness, decidability

sound

full clausal logic is sound

\[ P \vdash C \implies P \not\models C \]

complete

full clausal logic is refutation-complete

\[ P \cup \{C\} \text{ inconsistent} \implies P \cup \{C\} \vdash \square \]

Decidability

The question “\( P \not\models C? \)” 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 \( P \not\models C \), will answer “yes” in finite time but this algorithm may loop if \( P \models C \).
## Clausal Logic: overview

<table>
<thead>
<tr>
<th></th>
<th>propositional</th>
<th>relational</th>
<th>full</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Herbrand universe</strong></td>
<td>-</td>
<td>{a, b}</td>
<td>{a, f(a), f(f(a)), ...}</td>
</tr>
<tr>
<td><strong>Herbrand base</strong></td>
<td>{p, q}</td>
<td>{p(a,a), p(b,a), ...}</td>
<td>infinite</td>
</tr>
<tr>
<td>clause</td>
<td>p :- q</td>
<td>p(X,Z) :-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>q(X,Y), p(Y,Z)</td>
<td></td>
</tr>
<tr>
<td><strong>Herbrand models</strong></td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td></td>
<td>{p}</td>
<td>{p(a,a)}</td>
<td>{p(a,f(a)), q(a)}</td>
</tr>
<tr>
<td></td>
<td>{p, q}</td>
<td>{p(a,a), p(b,a), q(b,a)}</td>
<td>infinite number of finite or infinite models</td>
</tr>
<tr>
<td><strong>meta-theory</strong></td>
<td>sound</td>
<td>sound</td>
<td>sound (occurs check)</td>
</tr>
<tr>
<td></td>
<td>refutation-complete</td>
<td>refutation-complete</td>
<td>refutation-complete</td>
</tr>
<tr>
<td></td>
<td>decidable</td>
<td>decidable</td>
<td>semi-decidable</td>
</tr>
</tbody>
</table>
Clausal Logic: conversion to first-order predicate logic (1)

Every set of clauses can be rewritten as an equivalent sentence in first-order predicate logic.

Variables in a sentence cannot range over predicates.

married, bachelor :- man, adult.
haswife :- married.

becomes
(man ∧ adult ⇒ married ∨ bachelor) ∧
(married ⇒ haswife)

or
(¬man ∨ ¬adult ∨ married ∨ bachelor ) ∧
(¬married ∨ haswife)

reachable(X, Y, route(Z, R)) :- connected(X, Z, L), reachable(Z, Y, R).

becomes
∀X∀Y∀Z∀R∀L : ¬connected(X, Z, L) ∨
¬reachable(Z, Y, R) ∨
reachable(X, Y, route(Z, R))

Variables in clauses are universally quantified.

A ⇒ B ≡ ¬A ∨ B
¬(A ∧ B) ≡ ¬A ∨ ¬B

conjunctive normal form: conjunction of disjunction of literals
Clausal Logic: conversion to first-order predicate logic (2)

Every set of clauses can be rewritten as an equivalent sentence in first-order predicate logic.

\[
\text{nonempty}(X) :\neg \text{contains}(X,Y).
\]

becomes

\[
\forall X \forall Y : \text{nonempty}(X) \lor \neg \text{contains}(X,Y)
\]

or

\[
\forall X : (\text{nonempty}(X) \lor \forall Y \neg \text{contains}(X,Y))
\]

or

\[
\forall X : \neg (\exists Y : \text{contains}(X,Y))
\]

or

\[
\forall X : (\exists Y : \text{contains}(X,Y)) \Rightarrow \text{nonempty}(X)
\]

variables that occur only in the body of a clause are existentially qualified
Clausal Logic: conversion from first-order predicate logic (1)

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

∀\[\text{brick}(X)\] ⇒ (∃\[\text{Y} [\text{on}(X, Y) \land \neg \text{pyramid}(Y)]] \land

\neg \exists\[\text{Y} [\text{on}(X, Y) \land \text{on}(Y, X)]] \land

∀\[\text{Y} [\neg \text{brick}(Y) \Rightarrow \neg \text{equal}(X, Y)]]

1. eliminate ⇒ using A ⇒ B = ¬A ∨ B.

∀\[\neg \text{brick}(X) \lor (\exists\[\text{Y} [\text{on}(X, Y) \land \neg \text{pyramid}(Y)]] \land

\neg \exists\[\text{Y} [\text{on}(X, Y) \land \text{on}(Y, X)]] \land

∀\[\text{Y} [\neg (\neg \text{brick}(Y) \lor \neg \text{equal}(X, Y)]]

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

∀\[\neg \text{brick}(X) \lor (\exists\[\text{Y} [\text{on}(X, Y) \land \neg \text{pyramid}(Y)]] \land

\forall\[\text{Y} [\neg \text{on}(X, Y) \lor \neg \text{on}(Y, X)]] \land

\forall\[\text{Y} [\text{brick}(Y) \lor \neg \text{equal}(X, Y)]]

¬(A \land B) ≡ ¬A ∨ ¬B
¬(A \lor B) ≡ ¬A \land ¬B
¬(¬A) ≡ A
¬(\forall X [p(X)]) ≡ \exists X [\neg p(X)]
¬(\exists X [p(X)]) ≡ \forall X [\neg p(X)]
Clausal Logic: conversion from first-order predicate logic (2)

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

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

∀X∃Y : loves(X,Y)
∀X:loves(X,person_loved_by(X))

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

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

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

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

∃X∀Y : loves(X,Y)

Skolem constants substitute for an existentially quantified variable which does not occur in the scope of a universal quantifier

model {loves(paul,anna)} 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

3 replace ∃ using Skolem functors (abstract names for objects, functor has to be new)
Clausal Logic: conversion from first-order predicate logic (3)

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

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

4 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)] ∧ ∀Y [brick(Y) ∨ ¬equal(X,Y)]]

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

5 move ∀ to the front
Clausal Logic: conversion from first-order predicate logic (4)

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

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

∀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)])

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

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

A ∨ (B ∨ C) ≡ A ∨ B ∨ C
Clausal Logic:
conversion from first-order predicate logic (5)

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

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

7 split the conjuncts in clauses (a disjunction of literals)

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

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: conversion from first-order predicate logic (6)

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

1. **eliminate $\Rightarrow$**

2. **put into negation normal form**

3. **replace $\exists$ using Skolem functors**

4. **standardize variables**

5. **move $\forall$ to the front**

6. **convert to conjunctive normal form**

7. **split the conjuncts in clauses**

8. **convert to clausal syntax**

<table>
<thead>
<tr>
<th>Step</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\forall X: (\exists Y: \text{contains}(X,Y)) \Rightarrow \text{nonempty}(X))$</td>
</tr>
<tr>
<td>2</td>
<td>$\forall X: \neg (\exists Y: \text{contains}(X,Y)) \lor \text{nonempty}(X))$</td>
</tr>
<tr>
<td>3</td>
<td>$\forall X: (\forall Y: \neg \text{contains}(X,Y)) \lor \text{nonempty}(X))$</td>
</tr>
<tr>
<td>4</td>
<td>$\forall X: \forall Y: \neg \text{contains}(X,Y) \lor \text{nonempty}(X))$</td>
</tr>
<tr>
<td>5</td>
<td>$\text{nonempty}(X) :- \text{contains}(X,Y)$</td>
</tr>
</tbody>
</table>
Definite Clause Logic: motivation

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

how to use the clause depends on what you want to prove, but this indeterminacy is a source of inefficiency in refutation proofs.

logical consequences that can be derived in two resolution steps:

- clause is used from right to left
  - married(X); bachelor(X); :- man(X), adult(X)
  - married(peter); bachelor(peter); :- adult(peter)
  - married(peter); bachelor(peter)

- clause is used from left to right
  - married(X); bachelor(X); :- man(X), adult(X)
  - bachelor(maria); :- man(maria), adult(maria)
  - :- man(maria), adult(maria)

- both literals from head and body are resolved away
  - married(X); bachelor(X); :- man(X), adult(X)
  - married(paul); bachelor(paul); :- adult(paul)
  - married(paul); :- adult(paul)

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

\[ A \leftarrow B_1, \ldots, B_n \]

fixes direction to use clauses

from right to left:

\[ \Rightarrow \] procedural interpretation

“prove A by proving each of \( B_i \)”
Definite Clause Logic: recovering lost expressivity

can no longer express

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

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 \textbf{general} clause

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

second model is minimal model of \textbf{general} clause

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

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.