Horn Clauses
and SLD Resolution

Marco Piastra
Back to Propositional Logic
Horn Clauses (in $L_p$)

- **Definition**
  
  A *Horn Clause* is a wff in CF
  that contains at most one literal in positive form

- **Three types of Horn Clauses:**

  **Rule:** two or more literals, one positive
  
  Examples: $\{B, \neg D, \neg A, \neg C\}$, $\{A, \neg B\}$  
  (equivalent to: $(D \land A \land C) \rightarrow B$, $B \rightarrow A$)

  **Facts:** just one positive literal
  
  Examples: $\{B\}$, $\{A\}$

  **Goal:** one or more literals, all negative
  
  Examples: $\{\neg B\}$, $\{\neg A, \neg B\}$

  More terminology:
  
  Rules and facts are also called *definite clauses*
  
  Goals are also called *negative clauses*
Lost in Translation...

Many wffs can be translated into Horn clauses:

\[(A \land B) \rightarrow C\]
\[\neg (A \land B) \lor C\]
\[\neg A \lor \neg B \lor C\]

\[(A \rightarrow (B \land C))\]
\[\neg A \lor (B \land C)\]
\[(\neg A \lor B) \land (\neg A \lor C)\]
\[(\neg A \lor B), (\neg A \lor C)\]

\[(A \lor B) \rightarrow C\]
\[\neg (A \lor B) \lor C\]
\[(\neg A \land \neg B) \lor C\]
\[(\neg A \lor C) \land (\neg B \lor C)\]
\[(\neg A \lor C), (\neg B \lor C)\]

But not all of them:

\[(A \land \neg B) \rightarrow C\]
\[\neg (A \land \neg B) \lor C\]
\[\neg A \lor B \lor C\]

\[(A \rightarrow (B \lor C))\]
\[\neg A \lor B \lor C\]

(rewriting →)
(De Morgan - CF – it is a rule)
(rewriting →)
(Distributing V)
(CF – two rules)
(rewriting →)
(De Morgan)
(Distributing V)
(CF – two rules)
(rewriting →)
**SLD Resolution**

*Linear resolution with Selection function for Definite clauses*

- **Algorithm**
  
  Starts from a set of *definite clauses* (also the *program*) + a *goal*

  1) At each step, the *selection function* identifies a *literal* in the *goal* (i.e. *subgoal*)

  2) All *definite clause* applicable to the *subgoal* are selected, in the given order

  3) The resolution rule is applied generating the *resolvent*

  Termination: either the empty clause \{ \} is obtained or step 2) fails.

---

Example:

*Selection function: leftmost subgoal first*

*Definite clauses:* \{ C \}, \{ D \}, \{ B, \neg D \}, \{ A, \neg B, \neg C \}

*Goal:* \{ \neg A \}
SLD trees

SLD derivations

Example: \{C\}, \{D\}, \{B, \neg D\}, \{A, \neg B, \neg C\} goal \{\neg A\}

*In this example each subgoal can be resolved in one mode only*

*This is not true in general*

---

**SLD trees** (= trace of all SLD derivations from a goal)

Example: \{C\}, \{D\}, \{B, \neg F\}, \{B, \neg E\}, \{B, \neg D\}, \{A, \neg B, \neg C\} goal \{\neg A\}

*A few new rules have been added: there are now different possibilities*

Each branch correspond to a possible resolution for a subgoal
SLD Resolution

- A resolution method for Horn clauses in $L_P$
  - It always terminates
  - It is correct: $\Gamma \vdash \varphi \Rightarrow \Gamma \models \varphi$
  - It is complete: $\Gamma \models \varphi \Rightarrow \Gamma \vdash \varphi$

- Computationally efficient
  - It has polynomial time complexity (w.r.t the # of propositional symbols occurring in $\Gamma$ and $\varphi$)
SLD resolution in First-Order Logic
Horn Clauses in $L_{FO}$

The definition is very similar to the propositional case

- **Horn Clauses** (of the skolemization of a set *sentences*)
  
  Each clause contains at most one literal in positive form

**Facts, rules and goals**

**Fact:** a clause with just an individual *atom*

$$\{ \text{Greek(socrates)}, \{ \text{Pyramid(x)}, \{ \text{Sister(sally, motherOf(paul))} \} \}$$

**Rule:** a clause with at least two literals, exactly one in positive form

$$\{ \text{Human(x)}, \neg \text{Greek(x)} \}, \forall x \ (\text{Greek}(x) \rightarrow \text{Human}(x))$$

$$\{ \neg \text{Female}(x), \neg \text{Parent}(k(x),x), \neg \text{Parent}(k(y),y), \text{Sister}(x,y) \}$$

$$\forall x \forall y (\text{ Female}(x) \land \exists z (\text{Parent}(z,x) \land \text{Parent}(z,y))) \rightarrow \text{Sister}(x,y))$$

$$\forall x \forall y (\neg \text{Above}(x,y), \text{ On}(x,k(x))) \}, \{ \neg \text{Above}(x,y), \text{ On}(j(y),y) \}$$

$$\forall x \forall y (\text{Above}(x,y) \rightarrow (\exists z \text{ On}(x,z) \land \exists v \text{ On}(v,y)))$$

**Goal:** a clause containing negative literals only

$$\{ \neg \text{Mortal(socrates)} \}$$

$$\{ \neg \text{Sister(sally,x)}, \neg \text{Sister(x,paul)} \}$$

Negation of $\exists x (\text{Sister(sally,x) \land Sister(x,paul)})$
SLD Resolution in $L_{FO}$

- **Input:** a program $\Pi$ and a goal $\phi$

  Program $\Pi$ (i.e. a set of definite clauses: rules + facts) in some predefined linear order:

  $$\gamma_1, \gamma_2, \ldots, \gamma_n$$  
  (each $\gamma_i$ is a definite clause)

  Goal $\phi$ (i.e. a conjunction of facts in negated form), which becomes the **current goal** $\psi$

**Procedure:**

1) Select a negative literal $\neg \alpha$ (i.e. the **subgoal**) in the **current goal** $\psi$

2) Scan the program (in the predefined order) to identify a clause candidate literal $\gamma_i$

3) Try unifying $\neg \alpha$ and $std(\gamma_i)$ (i.e. apply the standardization of variables to $\alpha'$)

4) If there is a *unifier* $\sigma$ of $\neg \alpha$ and $std(\gamma_i)$, replace the current goal with the **resolvent** of $std(\gamma_i)[\sigma]$ and $\psi[\sigma]$ (i.e. first apply $\sigma$ to both $std(\gamma_i)$ and $\psi$ and then generate the resolvent)

5) Then, if the **resolvent** is the empty clause, terminate with **success**, otherwise select a new **current goal** and resume from step 1)

6) Else, if the unification fails, scan the program and select a new candidate literal $\gamma_i$ and resume from step 3)

7) Else, if there are no further clauses in the program, select a new **current goal** and resume from step 1)

8) If all the goals in the tree have been fully explored, terminate with **failure**
**SLD Resolution in** $L_{FO}$

- **Two alternative selection functions:**

  **Depth-first** (which is the most common...)
  - Always select the *most recent goal*, i.e. the *resolvent* which has been generated last, as the *current goal* $\phi$
  - Then, in the current goal $\phi$, select the leftmost *subgoal* $\neg \alpha$ not selected yet
  - When the current goal $\phi$ is fully explored and no new *resolvent* has been generated, select the next *most recent* goal in the tree (*backtracking*)

  **Breadth-first**
  - Always select the *least recent* goal as the *current goal* $\phi$
  - Then, in the current goal $\phi$, select the leftmost *subgoal* $\neg \alpha$ not selected yet
  - When the current goal $\phi$ is fully explored select the next *least recent* goal in the tree

**Comparison**

Breadth-first is a *fair* selection function, in the sense that every possible resolution will be eventually attempted (i.e. ‘it leaves nothing behind’).

Depth-first tends to save memory and be more efficient, but it is NOT *fair* (more to follow)
SLD Trees

Example (depth-first selection function):

\[ \Pi \equiv \{ \{ \text{Human}(x), \neg \text{Greek}(x) \}, \{ \text{Mortal}(y), \neg \text{Human}(y) \}, \{ \text{Greek}(\text{socrates}) \}, \{ \text{Greek}(\text{plato}) \}, \{ \text{Greek}(\text{aristotle}) \} \} \]

\[ \text{goal} \equiv \{ \neg \text{Mortal}(x) \} \]

"Is there anyone who is both human and mortal?"

1: \{ \neg \text{Mortal}(x) \} []

\{ \neg \text{Mortal}(x) \}, \{ \text{Mortal}(y_1), \neg \text{Human}(y_1) \} []

2: \{ \neg \text{Human}(y_1) \} [x/y_1]

\{ \neg \text{Human}(y_1) \}, \{ \text{Human}(x_1), \neg \text{Greek}(x_1) \} [x/y_1]

3: \{ \neg \text{Greek}(x_1) \} [x/y_1][y_1/x_1]

\{ \neg \text{Greek}(x_1) \} \{ \text{Greek}(\text{socrates}) \} [x/y_1][y_1/x_1][x_1/socrates]

4: \{ \} [x/y_1][y_1/x_1][x_1/socrates]
**Example** (depth-first selection function, forcing full exploration of SLD tree):

\[ \Pi \equiv \{ \{ Human(x), \neg Greek(x) \}, \{ Mortal(y), \neg Human(y) \}, \{ Greek(socrates) \}, \{ Greek(plato) \}, \{ Greek(aristotle) \} \} \]

\[ goal \equiv \{ \neg Mortal(x) \} \]

"Is there anyone who is both human and mortal?"

1: \{ \neg Mortal(x) \} []

\[ \neg Mortal(x), \{ Mortal(y_1), \neg Human(y_1) \} [] \]

2: \{ \neg Human(y_1) \} \[ x/y_1 \]

\[ \neg Human(y_1), \{ Human(x_1), \neg Greek(x_1) \} \[ x/y_1 \] \]

3: \{ \neg Greek(x_1) \} \[ x/y_1 \][y_1/x_1]

\[ \neg Greek(x_1), \{ Greek(socrates) \} \[ x/y_1 \][y_1/x_1] \]

\[ \neg Greek(x_1), \{ Greek(plato) \} \[ x/y_1 \][y_1/x_1] \]

\[ \neg Greek(x_1), \{ Greek(aristotle) \} \[ x/y_1 \][y_1/x_1] \]

4: \{ \} \[ x/y_1 \][y_1/x_1][x_1/socrates] \n
5: \{ \} \[ x/y_1 \][y_1/x_1][x_1/plato] \n
6: \{ \} \[ x/y_1 \][y_1/x_1][x_1/aristotle]
Another example (depth-first selection function):

\[ \Pi \equiv \{ \{\text{Mortal}(\text{felix}), \neg \text{Cat}(\text{felix})\}, \{\text{Human}(x), \neg \text{Greek}(x)\}, \{\text{Mortal}(y), \neg \text{Human}(y)\}, \{\text{Greek}(\text{socrates})\}, \{\text{Greek}(\text{plato})\}, \{\text{Greek}(\text{aristotle})\}\} \]

\[ \text{goal} \equiv \{ \neg \text{Mortal}(x)\} \]

"Is there anyone who is both human and mortal?"

1: \{ \neg \text{Mortal}(x)\} []

2: \neg \text{Cat}(\text{felix}) [x/\text{felix}]

2: cannot be resolved

3: \{ \neg \text{Human}(y_1)\} [x/y_1]

4: \{ \neg \text{Greek}(x_1)\} [x_1][y_1/x_1]

\{ \neg \text{Greek}(x_1)\} \{\text{Greek}(\text{socrates})\} [x_1][y_1/x_1][x_1/\text{socrates}]

\{\} [x_1][y_1/x_1][x_1/\text{socrates}]
*The discreet charme of functions*

- Representing data structures: *lists of items* $[a, b, c, \ldots]$

  Symbols in $\Sigma$
  
  - cons/2
  - *it's a function* that associates items (e.g. $a$) to a list (e.g. $[b, c]$)
  - $\text{cons}(a, \text{cons}(b, \text{cons}(c, \text{nil})))$ represents the list $[a, b, c]$

  - Append/3
  - *it's a predicate:* each pair of lists $x$ and $y$ is associated to their concatenation $z$
  - nil
  - *it's a constant,* represents the empty list.

  **Axioms (AL)**
  
  $\forall x \ Append(nil, x, x)$
  $\forall x \ \forall y \ \forall z \ (\text{Append}(x, y, z) \rightarrow \forall s \ Append(\text{cons}(s, x), y, \text{cons}(s, z)))$

  **Examples of entailment**
  
  $\{\text{AL} + \exists z \ Append(\text{cons}(a, \text{nil}), \text{cons}(b, \text{cons}(c, \text{nil})), z) \} \quad \vdash \quad Append(\text{cons}(a, \text{nil}), \text{cons}(b, \text{cons}(c, \text{nil})), \text{cons}(a, \text{cons}(b, \text{cons}(c, \text{nil}))))$

  $\{\text{AL} + \exists x \ \exists y \ Append(x, y, \text{cons}(a, \text{cons}(b, \text{nil})))\} \quad \vdash \quad Append(\text{cons}(a, \text{nil}), \text{cons}(b, \text{nil}), \text{cons}(a, \text{cons}(b, \text{nil})))$
  
  $\vdash \quad Append(\text{nil}, \text{cons}(a, \text{cons}(b, \text{nil})), \text{cons}(a, \text{cons}(b, \text{nil})))$
  
  $\vdash \quad Append(\text{cons}(a, \text{cons}(b, \text{nil})), \text{nil}, \text{cons}(a, \text{cons}(b, \text{nil})))$
The world of lists

- Lists of items \([a, b, c, \ldots]\)

  \textit{cons}/2
  
  it’s a function that associates items (e.g. \(a\)) to a list (e.g. \([b, c]\))
  
  \(\text{cons}(a, \text{cons}(b, \text{cons}(c, \text{nil})))\) is the list \([a, b, c]\)

  \textit{Append}/3
  
  it’s a predicate: each pair of lists \(x\) and \(y\) is associated to their \textit{concatenation} \(z\)

  \(\text{nil}\)
  
  it’s a constant, the empty list.

Shorthand notation (Prolog):

\[
[\ ] \leftrightarrow \text{nil} \\
[a] \leftrightarrow \text{cons}(a, \text{nil}) \\
[a,b] \leftrightarrow \text{cons}(a, \text{cons}(b, \text{nil})) \\
[a|[b,c]] \leftrightarrow \text{cons}(a, [b,c])
\]

Axioms (\textbf{AL})

\[
\forall x \ \text{Append}(\text{nil}, x, x) \\
\forall x \forall y \forall z \ (\text{Append}(x, y, z) \rightarrow \forall s \ \text{Append}([s, x], y, [s, z]))
\]
The world of lists

Problem: $\forall x \ Append(nil, x, x) \models \exists y \ \forall x \ Append(nil, cons(y, x), cons(a, x))$

1: $\forall x \ Append(nil, x, x), \neg \exists y \ \forall x \ Append(nil, cons(y, x), cons(a, x))$  (refutation)
2: $\forall x \ Append(nil, x, x), \forall y \ \exists x \ \neg Append(nil, cons(y, x), cons(a, x))$  (prenex normal form)
3: $\{\ Append(nil, x, x)\}, \{\neg Append(nil, cons(y, k(y)), cons(a, k(y))))\}$

$(k/1 \text{ is a Skolem function, clausal form})$

(N.B. there is no skolemization in Prolog : the programmer does it)

The pair of literals

$\ Append(nil, x, x), \neg Append(nil, cons(y, k(y)), cons(a, k(y))))$

... contains the same predicate $\ Append/3$ but the arguments are different

There is however an MGU $\sigma = [x/cons(a, k(a)), y/a]$ that yields

$\{\ Append(nil, cons(a,k(a)), cons(a,k(a))))\}, \{\neg Append(nil, cons(a, k(a)), cons(a, k(a))))\}$

From this, the resolvent is the empty clause.
The world of lists in Prolog

% Identical to built-in predicate append/3, although it uses "cons"
% as a defined predicate, thus allowing trace-ability.

append(cons(S,X),Y,cons(S,Z)) :- append(X,Y,Z).
append(nil,X,X).

% WARNING: express your queries with cons. Examples:
% ?- append(cons(a,nil), cons(b,cons(c, nil)),cons(a,cons(b,cons(c, nil)))).
% ?- append(X,Y,cons(a,cons(b,cons(c, nil)))).
Infinite SLD Trees (fairness of SLD)

- An example:

  \[ \Pi \equiv \{ \{ S(a,b) \}, \{ S(b,c) \}, \{ S(x,z), \neg S(x,y), \neg S(y,z) \} \} \]

  \[ \neg \phi \equiv \{ \neg S(a,x) \} \]

  goal: \[ \neg S(a,x) \]

  \[ \{ \neg S(a,x), \{ S(a,b) \} \] [\[ \]

  \[ \{ \} [x/b] \]

  Easy...
Infinite SLD Trees (fairness of SLD)

An example:

$$\Pi \equiv \{\{S(a,b)\}, \{S(b,c)\}, \{S(x,z), \neg S(x,y), \neg S(y,z)\}\}$$

$$\neg \phi \equiv \{\neg S(a,x)\}$$

Goal: $$\neg S(a,x)$$

\[\{\neg S(a,x)\}, \{S(a,b)\} \quad \{\neg S(a,x)\}, \{S(x_3,z_3), \neg S(x_3,y_3), \neg S(y_3,z_3)\} \quad \{\neg S(a,y_3), \neg S(y_3,z_3)\}, \{S(a,b)\} \quad \{\neg S(b,z_3)\} \quad \{\neg S(b,z_3)\}, \{S(b,c)\} \quad \{\} \quad \{\} \quad (\Rightarrow [x/c])\]

Forcing to backtrack…
(easy again)
Infinite SLD Trees (fairness of SLD)

- An example:

\[ \Pi \equiv \{\{S(a,b)\}, \{S(b,c)\}, \{S(x,z), \neg S(x,y), \neg S(y,z)\}\} \]
\[ \neg \phi \equiv \{\neg S(a,x)\} \]

goal: \( \neg S(a,x) \) [ ]

\[ [... \} \neg S(a,x), \{S(x,3, z_3), \neg S(x,3, y_3), \neg S(y,3, z_3)\} [x_3/a, x_3/z_3] \]
\[ \{\neg S(a,y_3), \neg S(y,3, z_3)\} [x_3/a, x_3/z_3] \]
\[ \{\neg S(a,y_3), \neg S(y,3, z_3)\}, \{S(a,b)\} [x_3/z_3, x_3/a] \]
\[ \{\neg S(b,z_3)\} [x_3/z_3, x_3/a] \]
\[ \{\neg S(b,z_3)\}, \{S(b,c)\} [x_3/z_3, x_3/a] \]
\[ \{\neg S(b,z_3)\}, \{S(x_4,z_4), \neg S(x_4,y_4), \neg S(y_4,z_4)\} [x_3/z_3, x_3/a] \]
\[ \{\neg S(b,y_4), \neg S(y_4,z_4)\} [x_3/z_3, x_3/a, z_3/z_4, x_4/b] \]
\[ \{\neg S(b,y_4), \neg S(y_4,z_4)\}, \{S(x_5,z_5), \neg S(x_5,y_5), \neg S(y_5,z_5)\} [x_3/z_3, x_3/a, z_3/z_4, x_4/b] \]
\[ \{\neg S(b,y_5), \neg S(y_5,z_5), \neg S(z_5,z_4)\} [x_3/z_3, x_3/a, z_3/z_4, x_4/b, y_4/z_5, x_5/b] \]

Forcing to backtrack…
(infinite loop)
Infinite SLD Trees (fairness of SLD)

- A second example:

\[ \Pi \equiv \{ \{ S(x,z), \neg S(x,y), \neg S(y,z) \}, \{ S(a,b) \}, \{ S(b,c) \} \} \]
\[ \neg \phi \equiv \{ \neg S(a,x) \} \]

goal: \[ \neg S(a,x) \]

\[ \{ \neg S(a,x), \{ S(x_1,z_1), \neg S(x_1,y_1), \neg S(y_1,z_1) \} \] \[
\{ \neg S(a,y_1), \neg S(y_1,z_1) \} [x_1/a, x/z_1] \]

\[ \{ \neg S(a,y_1), \neg S(y_1,z_1) \}, \{ S(x_2,z_2), \neg S(x_2,y_2), \neg S(y_2,z_2) \} [x_1/a, x/z_1] \]
\[ \{ \neg S(z_2,z_1), \neg S(x_2,y_2), \neg S(y_2,z_2) \} [x_1/a, x/z_1, x_2/a, y_1/z_2] \]

[...]

The infinite loop occurs immediately …
Infinite SLD Trees (fairness of SLD)

A second example:

\[ \Pi \equiv \{ \{ S(x,z), \neg S(x,y), \neg S(y,z) \}, \{ S(a,b) \}, \{ S(b,c) \} \} \]

\[ \neg \phi \equiv \{ \neg S(a,x) \} \]

\[
\begin{array}{c}
\text{goal: } \neg S(a,x) [] \\
\{ \neg S(a,x) \}, \{ S(x_1,z_1), \neg S(x_1,y_1), \neg S(y_1,z_1) \} [] \\
\{ \neg S(a,y_1), \neg S(y_1,z_1) \} [x_1/a, x/z_1] \\
\{ \neg S(a,y_1), \neg S(y_1,z_1) \}, \{ S(x_2,z_2), \neg S(x_2,y_2), \neg S(y_2,z_2) \} [x_1/a, x/z_1, x_2/a, y_1/z_2] \\
[...] \\
\{ \neg S(a,y_3), \neg S(y_3,z_3) \}, \{ S(a,b) \} [x/z_3, x_3/a] \\
\{ \neg S(b,z_3) \} [x/z_3, x_3/a] \\
\{ \neg S(b,z_3), \{ S(b,c) \} [x/z_3, x_3/a] \\
\{ \} [x/z_3, x_3/a, z_3/c] (\Rightarrow [x/c])
\end{array}
\]

Notice the change in clause ordering.....

The infinite loop occurs immediately ...

Backtracking never occurs in this case (due to the infinite loop), yet, if it occurred it would have produced the two correct results
Infinite SLD Trees (fairness of SLD)

- Moral

  - In both previous examples the infinite loop (i.e. *divergence*) is unavoidable
  - Yet in the first one, the method first produces the right results and then diverges
  - So in the first case the result is *complete* (i.e. all entailed formulae are derived)
    while in the second case the method is not

A **fair** selection function is such that no possible resolution will be postponed
indefinitely: that is, any possible resolution will be performed, eventually.

In the two previous examples, we used a *depth-first* exploration method of the SLD tree:
which is not complete (in the above sense)

A *breadth-first* exploration method is **fair** hence it is complete (in the above sense)

*In actual programming systems (e.g. Prolog) the depth-first is preferred for memory efficiency
since the breadth-first method forces to keep (most of) the whole SLD tree in memory*