Protokollsicherheit

Automated Verification – Reachability Secrecy
Logic Programming (Prolog)
Logic Programming (Prolog)

The main motto is to described declaratively the solution of a problem and not how it should be solved.
Logic Programming (Prolog)

The main motto is to described declaratively the solution of a problem and not how it should be solved.

Example

Imperative solution:

```java
int sum(int[] list) =
    result = 0;
    for(int i = 0; i < list.length; ++i)
        result += list[i]
    return result
```
Logic Programming (Prolog)

The main motto is **to described declaratively the solution of a problem and not how it should be solved.**

Example

Imperative solution:

```java
int sum(int[] list) =
    result = 0;
    for(int i = 0; i < list.length; ++i)
        result += list[i]
    return result
```

We define how to compute!
Logic Programming (Prolog)

The main motto is to described declaratively the solution of a problem and not how it should be solved.

Example

Functional solution:

```plaintext
fun sum ([]):= 0
  | sum(h :: lst) = h + sum(lst)
```

We define how to compute!
Logic Programming (Prolog)

The main motto is to described declaratively the solution of a problem and not how it should be solved.

Example

Prolog solution:

\[
\begin{align*}
\text{sum([], 0)} & \land \\
\forall H, T, N, M. [\text{sum}(T, M) & \land \text{eq}(N, M + H) \supset \text{sum}([H|T], N)]
\end{align*}
\]
Logic Programming (Prolog)

The main motto is **to described declaratively the solution of a problem and not how it should be solved.**

Example

Prolog solution:

\[
\begin{align*}
\text{sum}([], 0) \land \\
\forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], N)]
\end{align*}
\]

Programs are logical theories!

**We do not specify how to compute it, but just state what is true about it.**
Logic Programming (Prolog)

Queries → Prolog Program → Answers
Logic Programming (Prolog)

Prolog Program

Prolog Program

\[
\text{sum}([], 0) \land \\
\forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], N)]
\]
Logic Programming (Prolog)

Prolog Program

sum([], 0) ∧
∀H, T, N, M. [sum(T, M) ∧ eq(N, M + H) ⊃ sum([H|T], N)]

Query

sum([4, 2, 3, 4], X).
Logic Programming (Prolog)

Prolog Program

\[ \text{sum([], 0)} \land \forall H, T, N, M. \left[ \text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], N) \right] \]

Query

\[ \text{sum([4, 2, 3, 4], X)}. \]

Answer

\[ X = 13 \]
Logic Programming (Prolog)

Prolog Program

\[
\text{sum}([], 0) \land \\
\forall H, T, N, M. [ \text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], N) ]
\]

Query

\[
\text{sum}([4, 2, 3, 4], X).
\]

Answer

\[
X = 13
\]

What the prolog engine is doing is searching for a proof demonstrating that the query logically follows from the logic program.
Proof theory is the field of mathematics that studies the properties and structure of formal proofs.
Proof theory is the field of mathematics that studies the properties and structure of formal proofs.

Sequents

\[ F_1, \ldots, F_n \rightarrow G_1, \ldots, G_m \]
\[ F_1 \land \cdots \land F_n \supset G_1 \lor \cdots \lor G_m \]
Proof Theory Basics – Inference Rules
Proof Theory Basics – Inference Rules

Identity Rules

\[ \frac{\Gamma_1 \rightarrow A, \Delta_1 \Gamma_2, A \rightarrow \Delta_2}{\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2} \quad \text{Cut} \]

Init

\[ \frac{A \rightarrow A}{A \rightarrow A} \]
**Proof Theory Basics – Inference Rules**

**Identity Rules**

\[ \frac{}{A \rightarrow A} \text{ Init} \]

\[ \frac{\Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2}{\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2} \text{ Cut} \]

**Logical Rules**

\[ \frac{\Gamma_1, A \rightarrow \Delta_1 \quad \Gamma_2, B \rightarrow \Delta_2}{\Gamma_1, \Gamma_2, A \lor B \rightarrow \Delta_1, \Delta_2} \text{ } \lor_l \]

\[ \frac{}{\Gamma \rightarrow A_1, A_2, \Delta} \text{ } \lor_r \]
Proof Theory Basics – Inference Rules

**Identity Rules**

- $A \rightarrow A$ (Init)
- $\Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2$ (Cut)

**Logical Rules**

- $\Gamma_1, A \rightarrow \Delta_1 \quad \Gamma_2, B \rightarrow \Delta_2 \quad \Gamma \rightarrow A_1, A_2, \Delta$
  \[ \Rightarrow \Gamma_1, \Gamma_2, A \lor B \rightarrow \Delta_1, \Delta_2 \quad \lor_l \]

- $\Gamma, A, B \rightarrow \Delta$ (Negation)
  \[ \Rightarrow \Gamma, A \land B \rightarrow \Delta \quad \land_l \]

- $\Gamma_1 \rightarrow \Delta_1, G_1 \quad \Gamma_2 \rightarrow \Delta_2, G_2 \quad \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2, G_1 \land G_2$
  \[ \Rightarrow \land_r \]
Proof Theory Basics – Inference Rules

**Init**

\[ A \rightarrow A \]

**Identity Rules**

\[ \Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2 \]

\[ \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2 \]

**Cut**

**Logical Rules**

\[ \Gamma_1, A \rightarrow \Delta_1 \quad \Gamma_2, B \rightarrow \Delta_2 \]

\[ \Gamma_1, \Gamma_2, A \lor B \rightarrow \Delta_1, \Delta_2 \quad \lor_l \]

\[ \Gamma \rightarrow A_1, A_2, \Delta \]

\[ \Gamma_1 \rightarrow \Delta_1, G_1 \quad \Gamma_2 \rightarrow \Delta_2, G_2 \]

\[ \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2, G_1 \land G_2 \quad \land_r \]

\[ \Gamma, A, B \rightarrow \Delta \quad \land_l \]

\[ \Gamma, A \land B \rightarrow \Delta \]

\[ \Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, B \rightarrow \Delta_2 \]

\[ \Gamma_1, \Gamma_2, A \supset B \rightarrow \Delta_1, \Delta_2 \quad \supset_l \]

\[ \Gamma, A \rightarrow B, \Delta \]

\[ \Gamma \rightarrow A \supset B, \Delta \quad \supset_r \]
Proof Theory Basics – Inference Rules

**Identity Rules**

\[
\begin{align*}
\Gamma_1 \rightarrow A, \Delta_1 &\quad \Gamma_2, A \rightarrow \Delta_2 \\
\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2
\end{align*}
\]

**Logical Rules**

\[
\begin{align*}
\Gamma, A, B \rightarrow \Delta &\quad \land_l \\
\Gamma, A \land B \rightarrow \Delta
\end{align*}
\]

\[
\begin{align*}
\Gamma_1, A \rightarrow \Delta_1 &\quad \Gamma_2, B \rightarrow \Delta_2 \\
\Gamma_1, \Gamma_2, A \lor B \rightarrow \Delta_1, \Delta_2
\end{align*}
\]

\[
\begin{align*}
\Gamma \rightarrow A_1, A_2, \Delta &\quad \lor_r \\
\Gamma \rightarrow A_1 \lor A_2, \Delta
\end{align*}
\]

\[
\begin{align*}
\Gamma, A, B \rightarrow \Delta &\quad \lor_l \\
\Gamma, A \lor B \rightarrow \Delta
\end{align*}
\]

\[
\begin{align*}
\Gamma_1, A \rightarrow \Delta_1 &\quad \land_r \\
\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2, G_1 \land G_2
\end{align*}
\]

\[
\begin{align*}
\Gamma_1 \rightarrow \Delta_1, G_1 &\quad \lor_r \\
\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2, G_1 \lor G_2
\end{align*}
\]

\[
\begin{align*}
\Gamma, A \rightarrow B, \Delta &\quad \lor_l \\
\Gamma \rightarrow A \lor B, \Delta
\end{align*}
\]

\[
\begin{align*}
\Gamma, {A \{t/x\}} \rightarrow \Delta &\quad \forall_l \\
\Gamma, \forall A \rightarrow \Delta
\end{align*}
\]

\[
\begin{align*}
\Gamma, A \rightarrow B, \Delta &\quad \lor_r \\
\Gamma \rightarrow A \lor B, \Delta
\end{align*}
\]

\[
\begin{align*}
\Gamma, A \rightarrow B, \Delta &\quad \lor_l \\
\Gamma \rightarrow A \lor B, \Delta
\end{align*}
\]

\[
\begin{align*}
\Gamma, A \rightarrow \Delta &\quad \forall_r \\
\Gamma \rightarrow \forall x A, \Delta
\end{align*}
\]

\[
\begin{align*}
\Gamma, A \rightarrow \Delta &\quad \forall_l \\
\Gamma \rightarrow \forall x A, \Delta
\end{align*}
\]
Proof Theory Basics – Inference Rules

Identity Rules

Init: \[ A \rightarrow A \]

Cut: \[ \Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2 \]
\[ \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2 \]

Logical Rules

\[ \Gamma_1, A \rightarrow \Delta_1 \quad \Gamma_2, B \rightarrow \Delta_2 \]
\[ \Gamma_1, \Gamma_2, A \lor B \rightarrow \Delta_1, \Delta_2 \]
\[ \Gamma \rightarrow A_1, A_2, \Delta \]
\[ \Gamma_1 \rightarrow A \lor A_2, \Delta \]
\[ \Gamma_1 \rightarrow \Delta_1, G_1 \quad \Gamma_2 \rightarrow \Delta_2, G_2 \]
\[ \Gamma, A \land B \rightarrow \Delta \]
\[ \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2, G_1 \land G_2 \]
\[ \Gamma \rightarrow A \land B, \Delta \]
\[ \Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, B \rightarrow \Delta_2 \]
\[ \Gamma_1, \Gamma_2, A \supset B \rightarrow \Delta_1, \Delta_2 \]
\[ \Gamma, A \rightarrow B, \Delta \]
\[ \Gamma_1 \rightarrow \Delta_1, G_1 \quad \Gamma_2 \rightarrow \Delta_2, G_2 \]
\[ \Gamma \rightarrow A \supset B, \Delta \]
\[ \Gamma, A \{t/x\} \rightarrow \Delta \]
\[ \Gamma \rightarrow \forall x A, \Delta \]
\[ \Gamma, \forall A \rightarrow \Delta \]
\[ \Gamma, \bot \rightarrow \Delta \quad \bot_L \]
\[ \Gamma, A \{c/x\}, \Delta \rightarrow \Delta \quad \forall_r \]

Init: \[ A \rightarrow A \]

Cut: \[ \Gamma \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2 \]
\[ \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2 \]

Logical Rules

\[ \Gamma_1, A \rightarrow \Delta_1 \quad \Gamma_2, B \rightarrow \Delta_2 \]
\[ \Gamma, A_1, A_2, \Delta \]
\[ \Gamma \rightarrow A_1 \lor A_2, \Delta \]
\[ \Gamma_1 \rightarrow \Delta_1, G_1 \quad \Gamma_2 \rightarrow \Delta_2, G_2 \]
\[ \Gamma \rightarrow A_1 \land A_2, \Delta \]
\[ \Gamma, A \land B \rightarrow \Delta \]
\[ \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2, G_1 \land G_2 \]
\[ \Gamma \rightarrow A, \Delta_1 \quad \Gamma_2, B \rightarrow \Delta_2 \]
\[ \Gamma_1, \Gamma_2, A \supset B \rightarrow \Delta_1, \Delta_2 \]
\[ \Gamma, A \rightarrow B, \Delta \]
\[ \Gamma_1 \rightarrow \Delta_1, G_1 \quad \Gamma_2 \rightarrow \Delta_2, G_2 \]
\[ \Gamma \rightarrow A \supset B, \Delta \]
\[ \Gamma, A \{t/x\} \rightarrow \Delta \]
\[ \Gamma \rightarrow \forall x A, \Delta \]
\[ \Gamma, \forall A \rightarrow \Delta \]
\[ \Gamma, \bot \rightarrow \Delta \quad \bot_L \]
Proof Theory Basics – Inference Rules

Structural Rules
Proof Theory Basics – Inference Rules

Structural Rules

\[ \frac{\Gamma \rightarrow \Delta}{\Gamma, A \rightarrow \Delta} \quad W_l \quad \frac{\Gamma \rightarrow \Delta}{\Gamma \rightarrow A, \Delta} \quad W_r \]
Proof Theory Basics – Inference Rules

Structural Rules

\[ \frac{\Gamma \rightarrow \Delta}{\Gamma, A \rightarrow \Delta} \quad W_l \]
\[ \frac{\Gamma, A, A \rightarrow \Delta}{\Gamma, A \rightarrow \Delta} \quad C_l \]
\[ \frac{\Gamma \rightarrow \Delta}{\Gamma \rightarrow A, \Delta} \quad W_r \]
\[ \frac{\Gamma \rightarrow A, A, \Delta}{\Gamma \rightarrow A, \Delta} \quad C_r \]
Proof Theory Basics – Inference Rules

Structural Rules

\[
\begin{align*}
\frac{\Gamma \rightarrow \Delta}{\Gamma, A \rightarrow \Delta} & \quad \text{\(W_l\)} \\
\frac{\Gamma, A, A \rightarrow \Delta}{\Gamma, A \rightarrow \Delta} & \quad \text{\(C_l\)} \\
\frac{\Gamma, B, A, \Gamma' \rightarrow \Delta}{\Gamma, A, B, \Gamma' \rightarrow \Delta} & \quad \text{\(X_l\)}
\end{align*}
\]

\[
\begin{align*}
\frac{\Gamma \rightarrow \Delta}{\Gamma \rightarrow A, \Delta} & \quad \text{\(W_r\)} \\
\frac{\Gamma \rightarrow A, A, \Delta}{\Gamma \rightarrow A, \Delta} & \quad \text{\(C_r\)} \\
\frac{\Gamma \rightarrow \Delta, B, A, \Delta'}{\Gamma \rightarrow \Delta, A, B, \Delta'} & \quad \text{\(X_r\)}
\end{align*}
\]
**Proof Theory Basics – Inference Rules**

**Structural Rules**

\[
\begin{align*}
& \frac{\Gamma \rightarrow \Delta}{\Gamma, A \rightarrow \Delta} \quad W_l \\
& \frac{\Gamma, A, A \rightarrow \Delta}{\Gamma, A \rightarrow \Delta} \quad W_r \\
& \frac{\Gamma, B, A, \Gamma' \rightarrow \Delta}{\Gamma, A, B, \Gamma' \rightarrow \Delta} \quad X_l \\
& \frac{\Gamma \rightarrow \Delta, B, A, \Delta'}{\Gamma \rightarrow \Delta, A, B, \Delta'} \quad X_r
\end{align*}
\]

These contexts can be treated as sets of formulas.

We assume that the structural rules are applied implicitly.
Proof Theory Basics – Proofs
Proof Theory Basics – Proofs

Proofs are tree-like structures constructed using inference rules and whose leaves are instances of the initial rule and/or of the false rule.
Proofs are tree-like structures constructed using inference rules and whose leaves are instances of the initial rule and/or of the false rule.

\[
\begin{aligned}
P \rightarrow P & \quad \text{Init} \\
Q \rightarrow Q & \\
\frac{P, Q}{P \wedge Q} & \quad \wedge_r \\
\frac{P \wedge Q}{P \wedge Q} & \quad \wedge_l \\
\frac{}{(P \wedge Q) \supset (P \wedge Q)} & \quad \supset_r
\end{aligned}
\]
Proof Theory Basics – Proofs

Proofs are tree-like structures constructed using inference rules and whose leaves are instances of the **initial rule** and/or of the **false** rule.

\[
\begin{array}{c}
P \rightarrow P \quad \text{Init} \\
Q \rightarrow Q \quad \text{Init} \\
P, Q \rightarrow P \land Q \quad \land_r \\
P \land Q \rightarrow P \land Q \quad \land_l \\
\therefore (P \land Q) \supset (P \land Q) \quad \supset_r 
\end{array}
\]

**Theorem:** (Soudness and Completeness) A formula \( F \) is a **tautology** if and only if there is proof of \( \rightarrow F \).
Proof Theory Basics – Cut Elimination
Proof Theory Basics – Cut Elimination

\[
\Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2 \\
\frac{\Gamma_1, \Gamma_2}{\Delta_1, \Delta_2} \quad \text{Cut}
\]
Proof Theory Basics – Cut Elimination

This a new formula in the proof. Use of a Lemma.

$$\frac{\Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2}{\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2} \text{ Cut}$$
Proof Theory Basics – Cut Elimination

This a new formula in the proof. Use of a Lemma.

\[ \frac{\Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2}{\Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2} \]

Cut

Proof with Cuts

Proof without Cuts
Proof Theory Basics – Cut Elimination

This a new formula in the proof. Use of a Lemma.

\[ \Gamma_1 \rightarrow A, \Delta_1 \quad \Gamma_2, A \rightarrow \Delta_2 \]

\[ \Gamma_1, \Gamma_2 \rightarrow \Delta_1, \Delta_2 \]  

Cut

Proof with Cuts

Proof without Cuts

Consistency

\[ \rightarrow A \quad A \rightarrow \]

Sub-formula

No needs for Lemmas.
Logic Programming (Prolog)

Horn Clauses
Logic Programming (Prolog)

Horn Clauses

Goals (or queries)

\( G ::= true \mid A \mid G \land G \mid \exists x. G \)
Logic Programming (Prolog)

Horn Clauses

Goals (or queries)

\[ G ::= true \mid A \mid G \land G \mid \exists x. G \]

Programs

\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x. P \]
Logic Programming (Prolog)

Horn Clauses

Goals (or queries)

\[ G ::= \text{true} \mid A \mid G \land G \mid \exists x. G \]

Programs

\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x. P \]

Prove the sequent: \[ P \rightarrow G \]

\[
\begin{align*}
\text{sum}([], 0) \land \\
\forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], N)] \\
\rightarrow \\
\exists X. \text{sum}([4, 2, 3, 4], X)
\end{align*}
\]
Logic Programming (Prolog)

Computation is Cut-free Proof Search
Logic Programming (Prolog)

Computation is Cut-free Proof Search

\[ \mathcal{P} \rightarrow G \]
Logic Programming (Prolog)

Computation is Cut-free Proof Search

\( P \rightarrow G \)
Logic Programming (Prolog)

Computation is Cut-free Proof Search

\[ P \rightarrow G \]
Logic Programming (Prolog)

Computation is Cut-free Proof Search

Backtracking

$\mathcal{P} \rightarrow G$
Logic Programming (Prolog)

Computation is Cut-free Proof Search

Backtracking

Proof

\( P \rightarrow G \)
Logic Programming (Prolog)

\[ G ::= true \mid A \mid G \land G \mid \exists x. G \]
\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x. P \]

No disjunction on the right of sequents.
Logic Programming (Prolog)

\[ G ::= \text{true} \mid A \mid G \land G \mid \exists x.G \]

\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x.P \]

No disjunction on the right of sequents.

Single formula to the right-hand-side: \( F_1, \ldots, F_n \rightarrow G \)
Logic Programming (Prolog)

\[ G ::= \text{true} \mid A \mid G \land G \mid \exists x.G \]

\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x.P \]

No disjunction on the right of sequents.

Single formula to the right-hand-side: \( F_1, \ldots, F_n \rightarrow G \)

Proof Theory – Sequent Calculus LJ
Logic Programming (Prolog)

\[ G ::= \text{true} \mid A \mid G \land G \mid \exists x.G \]

\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x.P \]

No disjunction on the right of sequents.

Single formula to the right-hand-side: \[ F_1, \ldots, F_n \rightarrow G \]

Proof Theory – Sequent Calculus LJ

\[
\frac{\Gamma, P, Q \rightarrow G}{\Gamma, P \land Q \rightarrow G} \quad \land_l \\
\frac{\Gamma \rightarrow G_1 \quad \Gamma \rightarrow G_2}{\Gamma \rightarrow G_1 \land G_2} \quad \land_r
\]

\[
\frac{\Gamma \rightarrow P}{\Gamma, P \supset Q \rightarrow G} \quad \supset_l \\
\frac{\Gamma, Q \rightarrow G}{\Gamma, P \supset Q \rightarrow G} \quad \supset_r
\]

\[
\frac{\Gamma, P \{t/x\} \rightarrow G}{\Gamma, \forall P \rightarrow G} \quad \forall_l \\
\frac{\Gamma \rightarrow G\{t/x\}}{\Gamma \rightarrow \exists x.G} \quad \exists_r
\]

\[
\frac{\Gamma \rightarrow G}{\Gamma, \forall P \rightarrow G} \quad \forall_l
\]

\[
\frac{\text{Init}}{A \rightarrow A}
\]
Logic Programming (Prolog)

\[ G ::= true \mid A \mid G \land G \mid \exists x. G \]

\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x. P \]

No disjunction on the right of sequents.

Single formula to the right-hand-side: \( F_1, \ldots, F_n \rightarrow G \)

Proof Theory – Sequent Calculus LJ

\[
\frac{\Gamma, P, Q \rightarrow G}{\Gamma, P \land Q \rightarrow G} \quad \land_l \quad \frac{\Gamma \rightarrow G_1}{\Gamma \rightarrow G_1 \land G_2} \quad \land_r \\
\frac{\Gamma \rightarrow P}{\Gamma, Q \rightarrow G} \quad \supset_l \quad \frac{\Gamma \rightarrow G \{t/x\}}{\Gamma \rightarrow \exists x. G} \quad \exists_r \\
\frac{\Gamma, P \{t/x\} \rightarrow G}{\Gamma, \forall P \rightarrow G} \quad \forall_l \quad \frac{A \rightarrow A}{\text{Init}}
\]

Searching a proof with the rules above is too inefficient! The search space is huge!
Logic Programming (Prolog)

\[ G ::= \text{true} \mid A \mid G \land G \mid \exists x. G \]

\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x. P \]

Backchaining Rule
Logic Programming (Prolog)

\[ G ::= \text{true} \mid A \mid G \land G \mid \exists x.G \]

\[ P ::= A \mid G 
\supset A \mid P_1 \land P_2 \mid \forall x.P \]

Backchaining Rule

\[
\Gamma, \forall \vec{x}.[G_1 \land \cdots \land G_n \supset A] \rightarrow G_1\theta \quad \cdots \quad \Gamma, \forall \vec{x}.[G_1 \land \cdots \land G_n \supset A] \rightarrow G_n\theta
\]

\[
bc
\Gamma, \forall \vec{x}.[G_1 \land \cdots \land G_n \supset A] \rightarrow A_g
\]

where \( \theta \) is the most general unifier of \( A \) and \( A_g \).
Logic Programming (Prolog)

\[ G ::= true \mid A \mid G \land G \mid \exists x . G \]
\[ P ::= A \mid G \supset A \mid P_1 \land P_2 \mid \forall x . P \]

Backchaining Rule

\[
\frac{\Gamma, \forall \vec{x}.[G_1 \land \cdots \land G_n \supset A] \longrightarrow G_1\theta \quad \cdots \quad \Gamma, \forall \vec{x}.[G_1 \land \cdots \land G_n \supset A] \longrightarrow G_n\theta}{bc}
\]

where \( \theta \) is the **most general unifier** of \( A \) and \( A_g \).

The backchaining rule yields a goal directed search.

\[
\forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], N)]
\]

\[
\frac{\Gamma \longrightarrow \text{sum}(l, m) \quad \Gamma \longrightarrow \text{eq}(m + 2, m + 2)}{bc}
\]

\[
\frac{\Gamma \longrightarrow \text{sum}([2 \mid l], m + 2)}{bc}
\]
Logic Programming (Prolog)

\[\text{sum}([], 0) \land \]
\[\forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], M)]\]

\[
\frac{\Gamma \rightarrow \text{sum}(l, m)}{\Gamma} \quad \frac{\Gamma \rightarrow \text{eq}(m + 2, m + 2)}{\Gamma} \quad \frac{\Gamma \rightarrow \text{sum}([2 | l], m + 2)}{bc}
\]
Logic Programming (Prolog)

\[ \text{sum}([], 0) \land \]
\[ \forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], M)] \]

\[ \frac{\Gamma \rightarrow \text{sum}(l, m)}{\Gamma \rightarrow \text{sum}([2 | l], m + 2)} \quad \frac{\Gamma \rightarrow \text{eq}(m + 2, m + 2)}{\Gamma \rightarrow \text{sum}(l, m)} \quad \frac{\Gamma \rightarrow \text{sum}(l, m)}{\Gamma \rightarrow \text{eq}(m + 2, m + 2)} \quad \text{bc} \]

The backchaining rule can be interpreted as a collection of atomic rules.
Logic Programming (Prolog)

**Question:** Clearly proof search using the backchaining rule is sound, but is it complete?
Logic Programming (Prolog)

**Question:** Clearly proof search using the backchaining rule is sound, but is it complete?

**Theorem:** Any LJ proof of a sequent of the form $P \rightarrow G$ can be transformed into a proof that contains only occurrences of $\land_l$, $bc$, and $\exists_r$. 
Logic Programming (Prolog)

Question: Clearly proof search using the backchaining rule is sound, but is it complete?

Theorem: Any LJ proof of a sequent of the form $P \rightarrow G$ can be transformed into a proof that contains only occurrences of $\land_l$, $bc$, and $\exists_r$.

In fact, the proof of the theorem above tells a bit more. The set of proofs of the following shape is complete.
Logic Programming (Prolog)

**Question:** Clearly proof search using the backchaining rule is sound, but is it complete?

**Theorem:** Any LJ proof of a sequent of the form \( P \rightarrow G \) can be transformed into a proof that contains only occurrences of \( \land_l \), \( bc \), and \( \exists_r \).

In fact, the proof of the theorem above tells a bit more. The set of proofs of the following shape is complete.

In literature, these proofs are called **uniform proofs**.
Logic Programming (Prolog)

**Theorem:** Any LJ proof of a sequent of the form $P \rightarrow G$ can be transformed into a proof that contains only occurrences of $\land_l$, $bc$, and $\exists_r$.

**Proof Sketch:**
Logic Programming (Prolog)

**Theorem:** Any LJ proof of a sequent of the form $P \rightarrow G$ can be transformed into a proof that contains only occurrences of $\land_l$, $bc$, and $\exists_r$.

**Proof Sketch:**

The proof relies in permutation lemmas:

**Lemma 1:** All rules permute over $\land_l$. 
Logic Programming (Prolog)

**Theorem:** Any LJ proof of a sequent of the form $P \rightarrow G$ can be transformed into a proof that contains only occurrences of $\land_l$, $bc$, and $\exists_r$.

**Proof Sketch:**

The proof relies in permutation lemmas:

**Lemma 1:** All rules permute over $\land_l$.

\[
\begin{align*}
\frac{\Gamma, A, B \rightarrow F}{\Gamma, A \land B \rightarrow F} & \quad \land_l & \quad \frac{\Gamma, G, A, B \rightarrow R}{\Gamma, G, A \land B \rightarrow R} & \quad \land_l \\
\frac{\Gamma, F \supset G, A \land B \rightarrow R}{\Gamma, F \supset G, A, B \rightarrow R} & \quad \supset_l & \quad \frac{\Gamma, F \supset G, A, B \rightarrow R}{\Gamma, F \supset G, A \land B \rightarrow R} & \quad \supset_l \\
\end{align*}
\]
Logic Programming (Prolog)

**Theorem:** Any LJ proof of a sequent of the form $P \rightarrow G$ can be transformed into a proof that contains only occurrences of $\land_l, bc,$ and $\exists_r$.

**Proof Sketch:**

The proof relies in permutation lemmas:

**Lemma 1:** All rules permute over $\land_l$.  

**Lemma 2:** All other rules permute over each other.
Logic Programming (Prolog)

**Theorem:** Any LJ proof of a sequent of the form $P \rightarrow G$ can be transformed into a proof that contains only occurrences of $\land_l$, $bc$, and $\exists_r$.

**Proof Sketch:**

The proof relies in permutation lemmas:

**Lemma 1:** All rules permute over $\land_l$.

**Lemma 2:** All other rules permute over each other.

\[
\begin{align*}
\Gamma \rightarrow F & \quad \Gamma, G \rightarrow A \quad \Gamma, G \rightarrow B \\
\hline & \Gamma, G \rightarrow A \land B \\
\hline & \Gamma, F \supset G \rightarrow A \land B \\
\hline & \Gamma, F \supset G \rightarrow A \land B \\
\hline & \Gamma \rightarrow F \quad \Gamma, G \rightarrow B \\
\hline & \Gamma, F \supset G \rightarrow B \\
\hline & \Gamma, F \supset G \rightarrow A \land B \\
\hline & \Gamma, F \supset G \rightarrow A \land B \\
\hline & \Gamma \rightarrow F \quad \Gamma, G \rightarrow A \\
\hline & \Gamma, F \supset G \rightarrow A \\
\hline & \Gamma, F \supset G \rightarrow A \land B \\
\hline & \Gamma, F \supset G \rightarrow A \land B
\end{align*}
\]
Logic Programming (Prolog)

Proof Sketch:

**Lemma 1:** All rules permute over $\land_l$.

**Lemma 2:** All other rules permute over each other.
Logic Programming (Prolog)

Proof Sketch:

**Lemma 1:** All rules permute over $\land_l$.

**Lemma 2:** All other rules permute over each other.

Using Lemma 1, we can move all instances of $\land_l$ **downwards** in the proof.
Logic Programming (Prolog)

Proof Sketch:

**Lemma 1:** All rules permute over $\land_l$.

**Lemma 2:** All other rules permute over each other.

Using Lemma 2, we can organize the remaining rules to form a backchaining rule instance.
How about the witnesses for existentials?
Logic Programming (Prolog)

How about the witnesses for existentials?

\[ \text{sum([], 0)} \land \forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], M)] \rightarrow \exists X. \text{sum}([4, 2, 3, 4], X) \]
Logic Programming (Prolog)

How about the witnesses for existentials?

\[
\text{sum}([], 0) \land \\
\forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], M)] \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \rightarrow \\
\exists X. \text{sum}([4, 2, 3, 4], X)
\]

Unification delays the search for witnesses for the \( \exists \) rule.

\[
\text{sum}([], 0) \land \\
\forall H, T, N, M. [\text{sum}(T, M) \land \text{eq}(N, M + H) \supset \text{sum}([H|T], M)] \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \rightarrow \\
\text{sum}([4, 2, 3, 4], _X)
\]

where \( _X \) is a logical variable created by the logic interpreter (Prolog engine). That is, this is a strictly implementation technique. There is no correspondence to logic.
There is still one decision to make: which clause to backchain?
There is still one decision to make: which clause to backchain?

\[ \forall X. p(f(X)) \supset p(X) \]

\[ \Gamma, \forall X. [p(f(X)) \supset p(X)] \rightarrow p(f(a)) \]

\[ \Gamma, \forall X. [p(f(X) \supset p(X)] \rightarrow p(a) \]

\[ bc \]
Logic Programming (Prolog)

There is still one decision to make: which clause to backchain?

\[ \forall X. p(f(X)) \supset p(X) \]

\[ \Gamma, \forall X. [p(f(X)) \supset p(X)] \rightarrow p(f(a)) \]

\[ \Gamma, \forall X. [p(f(X)) \supset p(X)] \rightarrow p(a) \]

If the interpreter always backchains on the clause above, then it will not terminate.
Verifying Protocols with Prolog

We model the intruder as well as the protocol by using Horn clauses.

As before, we assume that there are constructors and destructors.

For each constructor \( f \), we add a rule of the form:

\[
\text{att}(M_1) \land \cdots \land \text{att}(M_n) \supset \text{att}(f(M_1, \ldots, M_n))
\]

Examples

\[
\begin{align*}
\text{att}(m) \land \text{att}(pk) & \supset \text{att}(\text{aenc}(m, pk)) \\
\text{att}(sk) & \supset \text{att}(\text{pk}(sk)) \\
\text{att}(m) \land \text{att}(sk) & \supset \text{att}(\text{sign}(m, sk))
\end{align*}
\]
For each destructor defined by $g(M_1, \ldots, M_n) = M$, we add a rule of the form:

$$\text{att}(M_1) \land \cdots \land \text{att}(M_n) \supset \text{att}(M)$$

Destructors never appear in the rules. They are modelled using pattern-matching.
Verifying Protocols with Prolog

Specification of a Protocol (by example)

\[ A \rightarrow B : \{\text{sign}(k, sk_A)\}_{pk_B} \]

The intruder starts protocols with anyone he wants, or alternatively it can intercept \( A \)'s message faking to be \( B \).

\[ \text{att}(pk(k)) \supset \text{att}(\text{aenc}(\text{sign}(k, sk_A[])), pk(k)) \]

However, keys are fresh!

\[ \text{att}(pk(k)) \supset \text{att}(\text{aenc}(\text{sign}(k[pk(x)], sk_A[])), pk(k)) \]

This is the first abstraction that Proverif uses. Fresh values are considered as **functions** of the messages previously received by the protocol.
Once $B$ receives the message of the form $\{\text{sign}(k', sk)\}_{pk_B}$ she checks whether the signature is correct, that is, $sk = sk_A[]$:

$$B \rightarrow A : \{s[]\}_{k'}$$

In our model, the attacker does all this:

$$\text{att(enc(}\text{sign}(k', sk_A[]), pk(sk_B)) \supset \text{att(aenc(s[], k'))}$$

A symmetric rule is added for when $B$ starts the protocol.

In general, a protocol that contains $n$ messages encoded by $n$ sets of rules, each set containing the same corresponding rule for each principal.
Verifying Protocols with Prolog

Reachability Secrecy Problem

Given a logic program \( P \) encoding a protocol containing a secret \( s \), is the fact \( \text{att}(s[]) \) derivable from \( P \).

Notice that since we are in classical logic, the number of times a message is used is not evident.

\[
\text{att(enc}(s[], k))
\]

In a derivation, this message can be derived as many times needed. This is because formulas can be contracted. Hence, messages meant to appear only once in a future protocol may mix up with messages in the past. This may lead to false attacks.
Verifying Protocols with Prolog

Problem with Prolog Proof Search

Using the backchaining rule with the rule below would lead to non-termination.

\[
\text{att(enc}(m, \text{pk}(sk))) \land \text{att}(sk) \supset \text{att}(m)
\]

We need another solving algorithm!
Verifying Protocols with Prolog

Main Idea

The solving algorithm works in two phases:

In the first phase, we pre-process the logic program by constructing rewriting rules so that the body of all rules of the form \texttt{att}(x).

The second phase is to use the same Prolog search strategy but using the pre-processed program.
Verifying Protocols with Prolog

Phase One

We want that the body of all clauses are of the form $\text{att}(x)$.

**Selection function:** $\text{sel}(F_1 \land \cdots \land F_n \supset F) \subseteq \{F_1, \ldots, F_n\}$

$\text{sel}(F_1 \land \cdots \land F_n \supset F) = \emptyset$ if $F_i = \text{att}(x)$ for all $1 \leq i \leq n$.

$\text{sel}(F_1 \land \cdots \land F_n \supset F) = \{F_i\}$ otherwise, where $F_i$ has the greatest size.

**Example**

$\text{sel}(\text{att(enc}(m, \text{pk}(sk))) \land \text{att}(sk) \supset \text{att}(m)) = \text{att(enc}(m, \text{pk}(sk)))$
Verifying Protocols with Prolog

Phase One – Resolution

\[ R = F_1 \land \cdots \land F_n \supset F \quad R' = F'_1 \land \cdots \land F'_{n'} \supset F' \]

\( \text{sel}(R) = \emptyset \)

\[ F'_i \in \text{sel}(R') \]

Add to the program the following clause, denoted as \( R \circ_{F'_i} R' \)

\[ F_1 \theta \land \cdots \land F_n \theta \land F'_1 \theta \land \cdots F'_{i-1} \theta \land F'_{i+1} \theta \land \cdots \land F'_{n'} \theta \supset F' \theta \]

This operation can be seen as a forward-chaining step.

Example

\[ \text{att}(aenc(\text{sign}(z, sk_A[]), pk_B[])) \supset \text{att}(\text{enc}(s, z)) \]

\[ \text{att}(m) \land \text{att}(pk) \supset \text{att}(\text{enc}(m, pk)) \]

\[ \text{att} (\text{sign}(z, sk_A[])) \land \text{att}(pk_B[]) \supset \text{att}(\text{enc}(s, z)) \]