Automated Reasoning and Learning
-
CP0 SLD Resolution Trees

Ramon Béjar Torres[†]

March 21, 2024

[†]ramon.bejar@udl.cat

Table of Contents

Solving CP0 Problems with with SLD Resolution

So far, we have considered no fixed strategy when we want to answer a logical consequence question from a set of clauses P:

$$P \models Q$$

This query is always transformed to a single logical formula:

$$P \cup \neg Q$$

and then we use any solving algorithm to check if it is SAT/UNSAT.

Right ?

Solving CP0 Problems with with SLD Resolution

But there is an **almost deterministic** strategy to solve the problem, that works even with first-order logic (CP1), that is well suited for the use of logical formulas as a programming language !

Solving CP0 Problems with with SLD Resolution

But there is an **almost deterministic** strategy to solve the problem, that works even with first-order logic (CP1), that is well suited for the use of logical formulas as a programming language !

It is called SLD-Resolution, but works only if all the clauses of the formula are Horn clauses.

Solving CP0 Problems with with SLD Resolution

But there is an **almost deterministic** strategy to solve the problem, that works even with first-order logic (CP1), that is well suited for the use of logical formulas as a programming language !

It is called SLD-Resolution, but works only if all the clauses of the formula are Horn clauses.

A Horn clause is a clause where there is **at most** one positive literal.

Solving CP0 Problems with with SLD Resolution

SLD-Resolution stands for *Selection-rule driven Linear resolution for Definite clauses*. These are the ingredients of the almost deterministic SLD-Resolution strategy to derive the empty clause:

- The first resolvent is always obtained from the top-goal clause and another input clause, giving resolvent R_1
- The next resolvent R_{i+1} is obtained from the previous one R_i and one input clause of the formula.
- If there are many literals that can be selected to build a resolvent from two clauses, the literal to use is selected with a fixed selection rule.

Solving CP0 Problems with with SLD Resolution

The adjetive *linear* comes from the fact that each resolvent is always obtained from the previous one (and one input clause of the formula). So, they form a linked chain of resolvents:

$$R_1, R_2, \ldots, R_n$$

So, resolution proofs can be described as lineal chains of resolvents (no need to obtain resolvents from TWO previous resolvents), where the last resolvent is the empty clause.

However this is still not a fully deterministic strategy: there can be many possible such chains of resolvents, and not all of them will lead to the empty clause (if the empty clause can be derived).

Solving CP0 Problems with with SLD Resolution

Good news:

> *if there is a proof of the empty clause from the set of clauses, then it can be obtained with one sequence of resolvents obtained with this strategy*

The set of all SLD resolution-chains can be organized in a search tree, but we still have to decide how to search trough all possible SLD resolution-chains.

CP0 Logic Programming

The logic programming system Prolog, searchs for a refutation of Q with a logic program P using SLD-resolution
To consider all possible SLD resolution-chains, it selects the clauses in the original order given by the user, and each chain is totally explored before exploring the next one (depth-first search).
A logic program is simply a sequence of Horn clauses:

- clauses with only a positive literal (p_1), are called **facts**.
- clauses with a positive literal p_0 and some negative ones $\neg p_1, \neg p_2, \ldots, \neg p_n$ are called rules, because:

$$(p_0 \vee \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n) \equiv p_0 \leftarrow p_1, p_2, \ldots, p_n$$

- clauses with only negative literals $\neg p_1, \neg p_2, \ldots, \neg p_n$ are called goal clauses, as they are generated when we negate a query clause Q:

$$P \cup \neg(p_1 \wedge p_2 \wedge \ldots \wedge p_n) \equiv P \cup (\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n)$$

CP0 Resolution Trees - Example 1

Given the following propositional logic program P:

1. $no\_barro\_zapatos.$
2. $no\_es\_asesino \leftarrow no\_ha\_saltado.$
3. $no\_ha\_saltado \leftarrow no\_barro\_zapatos.$

Then, the answer to the question:

$$P \models no\_es\_asesino \quad \equiv \quad P \cup \{\neg no\_es\_asesino\} \text{ is } \text{UNSAT}$$

is obtained with SLD resolution as follows

CP0 Resolution Trees - Example 1 (shorter names)

Same logic program P, but with shorter names por propositions:

$$1.\ nbz.$$

$$2.\ noase \leftarrow nosalt.$$
$$3.\ nosalt \leftarrow nbz.$$

Then, the answer to the question:

$$P \models noase \quad \equiv \quad P \cup \{\neg noase\} \text{ is } \text{UNSAT}$$

is obtained with SLD resolution as follows

CP0 Resolution Trees - Example 1

Clauses Notation:

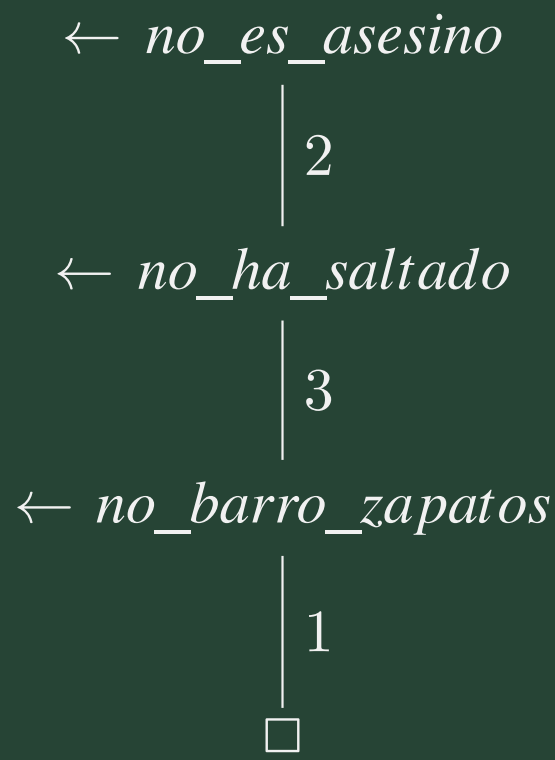
1. clause $\{\neg noase\}$ is resolved with clause 2 $\{noase, \neg nosalt\}$, obtaning the resolvent $\{\neg nosalt\}$
2. resolvent $\{\neg nosalt\}$ is resolved with the clause 3 $\{nosalt, \neg nbz\}$, obtaning the resolvent: $\{\neg nbz\}$
3. Finally, clause $\{\neg nbz\}$ is resolved with the clause 1 $\{nbz\}$, obtaining the empty resolvent : $\square$.

Logic Programming Notation:

1. the top goal $\leftarrow noase$ is resolved with $noase \leftarrow nosalt$, obtaning the new goal: $\leftarrow nosalt$
2. the goal $\leftarrow nosalt$ is resolved with $nosalt \leftarrow nbz$, obtaning the new goal: $\leftarrow nbz$
3. Finally, the goal $\leftarrow nbz$ is resolved with $nbz \leftarrow$, obtaining the empty resolvent : $\square$

CP0 Resolution Trees - Example 1

This SLD-resolution proof can be represented graphically as the following tree with a single branch:



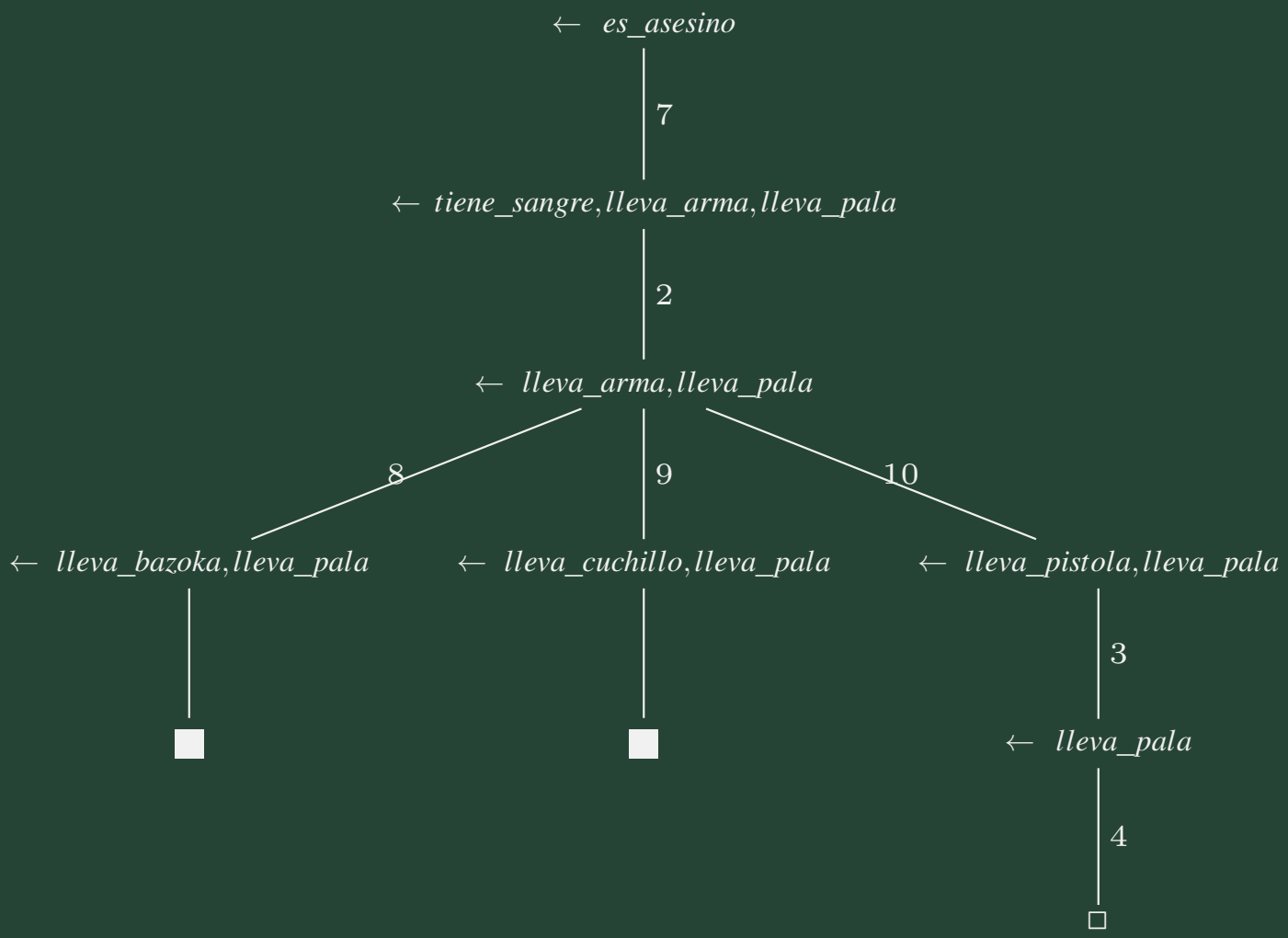
CP0 Resolution Trees - Example 2

But in general, we may have many resolution paths in the tree, and only some of them lead to the empty clause. The other paths, we say they are failed (■).

1. $no\_barro\_zapatos.$
2. $tiene\_sangre.$
3. $lleva\_pistola.$
4. $lleva\_pala.$

5. $no\_es\_asesino \leftarrow no\_ha\_saltado.$
6. $no\_ha\_saltado \leftarrow no\_barro\_zapatos.$
7. $es\_asesino \leftarrow tiene\_sangre, lleva\_arma, lleva\_pala.$
8. $lleva\_arma \leftarrow lleva\_bazoka.$
9. $lleva\_arma \leftarrow lleva\_cuchillo.$
10. $lleva\_arma \leftarrow lleva\_pistola.$

CP0 Resolution Trees - Example 2 ($P \models es\_asesino$)



CP0 Resolution Trees - Example 2

But remember, if there is a resolution proof of the empty clause from the set of clauses, then it exists a linear sequence of resolvents (SLD resolution-chain) that leads to the empty clause !

So, if the search tree considers all such possible resolution chains, the empty clause will appear in at least one of the paths of the tree. We say that SLD-Resolution is complete for Horn clauses.

Why Prolog uses (by default) depth-first search and not other strategies ?