Reasoning with Propositional Logic (CP0)
SLD-Resolution: Examples

Ramón Béjar Torres

March 21, 2024

1 CP0 Logic Programs - Examples

Example 1 *Given the following propositional logic program P:*

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

What consequences can we obtain from it ?

$$\begin{array}{c}
\leftarrow \;\; no_es_asesino \\
\Big|\; 2 \\
\leftarrow \;\; no_ha_saltado \\
\Big|\; 3 \\
\leftarrow \;\; no_barro_zapatos \\
\Big|\; 1 \\
\square
\end{array}$$

Figure 1: SLD-Resolution tree for the logic program of Example **??** with top goal $\leftarrow no\_es\_asesino$

If we want to prove the logical consequence:

$$P \models no\_es\_asesino$$

We can do it by finding an SLD-Refutation for $\neg no\_es\_asesino$ via P, or in logic programming terminology: an SLD-Refutation for the goal $\leftarrow no\_es\_asesino$ via P. Figure **??** shows the refutation tree for such goal, where we can see that there is indeed an unique refutation proof for the goal. Every edge of the tree connects a goal clause (at the top of the edge) with a resolvent clause (at the bottom of the edge) obtained from the goal clause and the clause indicated in the label of the edge. For example, the top-most edge of the tree of Figure **??** indicates that from the goal clause $\leftarrow no\_es\_asesino$ and the clause $no\_es\_asesino \leftarrow no\_ha\_saltado$ (clause 2) we obtain the resolvent clause $\leftarrow no\_ha\_saltado$. We can write this resolution inference step with the traditional CP0 clausal form syntax as follows:

$$\{\neg no\_es\_asesino\}, \{no\_es\_asesino, \neg no\_ha\_saltado\} \vdash \{\neg no\_ha\_saltado\}$$

and analogously for the other SLD-resolution steps in the tree.

With a Prolog interpreter searching for SLD-refutations in depth-first manner, that refutation is found with the following sequence of resolvents:

1. the top goal $\leftarrow no\_es\_asesino$ is resolved with the clause 2, obtaning the resolvent (new goal): $\leftarrow no\_ha\_saltado$

2. the goal $\leftarrow no\_ha\_saltado$ is resolved with the clause 3, obtaning the resolvent (new goal): $\leftarrow no\_barro\_zapatos$

3. Finally, the goal $\leftarrow no\_barro\_zapatos$ is resolved with the clause 1, obtaining the empty resolvent : $\square$

this sequence of SLD-resolvents corresponds to the unique branch of the SLD-Resolution Tree of Figure **??**.

Example 2 *Consider next the following propositional logic program P:*

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. $lleva\_arma \leftarrow lleva\_pistola.$

Observe that this example has rules for two opposite propositions: $no\_es\_asesino$ and $es\_asesino$, but because they are represented as independent propositions, it is possible to have proofs (SLD-refutations) for both propositions.

We clearly have a refutation for $\leftarrow no\_es\_asesino$, because the SLD-Resolution tree we have seen in the previous example is still valid with this new program.

So, we would like to check if we have also enough evidence so that $\leftarrow es\_asesino$ is also a consequence of P. The SLD-Resolution Tree for this second goal is shown in Figure **??**.

This new SLD-Resolution Tree is quite different from the first one. Observe that we have two failed branches, that correspond with trying to find a SLD-Resolution proof for $\leftarrow es\_asesino$ using either clause $lleva\_arma \leftarrow lleva\_bazoka$ (clause 8) or clause $lleva\_arma \leftarrow lleva\_cuchillo$ (clause 9). But for building proofs using theses clauses we

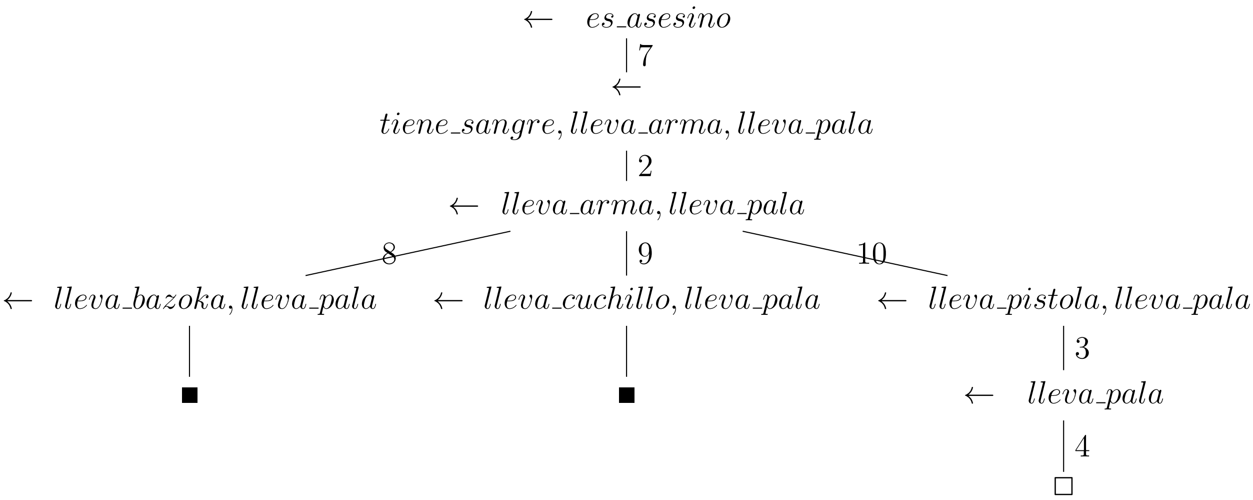

Figure 2: SLD-Resolution Tree for the logic program of Example **??** with top goal $\leftarrow$ *es_asesino*

would need to have either the fact *lleva_bazoka* or *lleva_cuchillo* in the program P, so this is why these SLD-Resolution proofs, related to these two branches, fail. However, with the third branch, the one that uses clause *lleva_arma* $\leftarrow$ *lleva_pistola* (clause 10), we have a succesful SLD-Resolution proof, because we can finally resolve all the subgoals, thanks to clauses 3 and 4.

With a Prolog interpreter searching for SLD-refutations in depth-first manner, and considering that clauses are selected with the same order found in the program, the refutation is found with the following sequence of resolvents:

1. the top goal $\leftarrow$ *es_asesino* is resolved with the clause 7, obtaning the resolvent (new goal): $\leftarrow$ *tiene_sangre*, *lleva_arma*, *lleva_pala*

2. the goal $\leftarrow$ *tiene_sangre*, *lleva_arma*, *lleva_pala* is resolved with the clause 2, obtaning the resolvent (new goal): $\leftarrow$ *lleva_arma*, *lleva_pala*

3. the goal $\leftarrow$ *lleva_arma*, *lleva_pala* is first resolved with the clause 8, obtaning the resolvent (new goal): $\leftarrow$ *lleva_bazoka*, *lleva_pala*. But this new goal cannot be resolved, as there is no clause for resolving with $\leftarrow$ *lleva_bazoka*.

4. So, the interpreter backtracks to the goal obtained in step 2, and obtains a new resolvent but this time with clause 9, obtaning the resolvent (new goal): $\leftarrow$ *lleva_cuchillo*, *lleva_pala*. But this new goal cannot also be resolved, as there is no clause for resolving with $\leftarrow$ *lleva_cuchillo*.

5. So, the interpreter backtracks again to the goal obtained in step 2, and obtains a new resolvent but this time with clause 10, obtaining the resolvent (new goal): $\leftarrow$ *lleva_pistola*, *lleva_pala*.

6. the goal $\leftarrow$ *lleva_pistola*, *lleva_pala* is resolved with clause 3, obtaining the resolvent (new goal): $\leftarrow$ *lleva_pala*.

7. Finally, the goal $\leftarrow lleva\_pala$ is resolved with clause 4, obtaining the empty resolvent : $\Box$

Example 3 *Consider next the following propositional logic program P:*

1. $head\_ache.$
2. $high\_temperature.$
3. $white\_skin.$
4. $mad\_cow\_disease \leftarrow white\_skin, high\_temperature, feel\_bad.$
5. $mad\_cow\_disease \leftarrow white\_skin, feel\_bad, i\_say\_moo.$
6. $flu\_virus \leftarrow white\_skin, head\_ache, feel\_bad.$
7. $feel\_bad \leftarrow i\_see\_barcenas.$
8. $feel\_bad \leftarrow i\_see\_cospedal.$

This is a logic program for trying to decide whether a patient is having one of two possible diseases: $mad\_cow\_disease$ and flu_virus, given the sintoms of the patient, that are represented as propositional facts (clauses 1 to 3) of the program.

To check first whether $mad\_cow\_disease$ can be considered a consequence of the facts and rules of the program P, we traverse the SLD-Resolution Tree for the goal $\leftarrow mad\_cow\_disease$. That tree is shown in Figure **??**. Observe that it contains 4 branches, all of them failed, so $mad\_cow\_disease$ is not a consequence of the program P. They fail because to be succesfully solved we would need to have either the fact $i\_see\_barcenas$ or the fact $i\_see\_cospedal$ in the program P. Observe that we fail two times because of the missing $i\_see\_barcenas$ fact (branches 1 and 3) and the two other times because of the missing $i\_see\_cospedal$ fact (branches 2 and 4). So, even if the branches are all different, the reasons for failure are not different between all the branches, but in a Prolog interpreter there is no general way of *predicting* whether a previously seen failed goal will be encountered again in future branches of the SLD-Resolution Tree.

At this point, you should be able to understand the sequence of steps followed by a depth-first search on such tree. So, consider the next exercise.

Exercise 1 *With the help of the SLD-Resolution Tree of Figure* **??**, *write down the sequence of steps followed by a Prolog interpreter, working with depth-first search, when looking for a SLD-Resolution proof for the goal $\leftarrow mad\_cow\_disease$ via the program of Example* **??**.

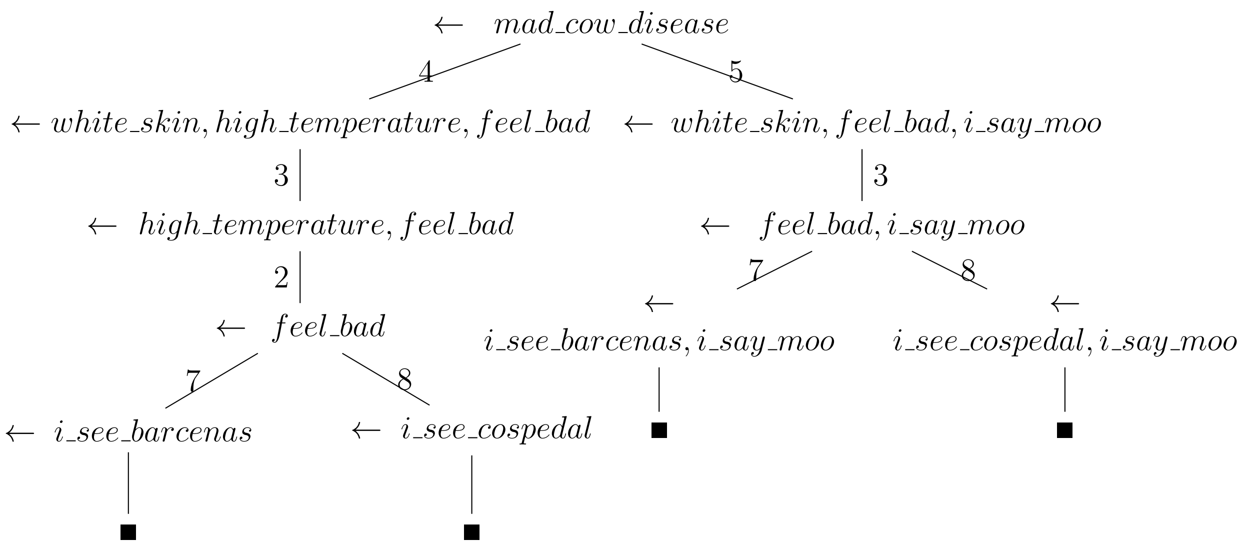

Figure 3: Failed SLD-Resolution Tree for the logic program of Example **??** with top goal $\leftarrow mad\_cow\_disease$

Next, we can check whether flu_virus is a consequence of the program P. This time, the corresponding SLD-Resolution Tree is shown in Figure **??**. Again, it is a failed SLD-Resolution Tree (its two branches are failed). And even if this tree and the one of the previous top goal are different, again the reason of failure of their branches is the same: the goal $\leftarrow feel\_bad$ can never be succesfully solved because we do not have the fact $i\_see\_barcenas$ and the fact $i\_see\_cospedal$ in the program P.

A final exercise:

Exercise 2 *As in the previous exercise, with the help of the SLD-Resolution Tree of Figure* **??***, write down the sequence of steps followed by a Prolog interpreter, working with depth-first search, when looking for a SLD-Resolution proof for the goal* $\leftarrow flu\_virus$ *via the program of Example* **??***.*

2 A Decision Procedure for Logic Programs with Propositional Logic

We say that given a propositional logic program P and a goal clause $\leftarrow G$, searching for a proof in its associated SLD-Resolution Tree, in depth-first manner or even in breadth-first manner, is a decision procedure for the problem of deciding whether:

$$P \models G$$

That is, the procedure will finish in one of these two situations:

1. If $P \models G$ holds, then it will find a refutation proof for $\leftarrow G$ (a succesful branch in the tree).
2. If $P \models G$ does not hold, then it will inform that there is no such a refutation proof for $\leftarrow G$, because **all the branches** in the tree will be failed.

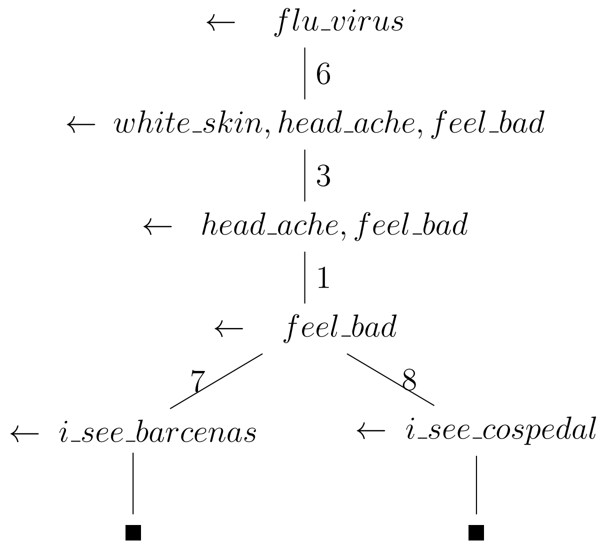

Figure 4: Failed SLD-Resolution Tree for the logic program of Example **??** with top goal $\leftarrow flu\_virus$

So, the procedure always *decides correctly* whether $P \models G$ holds. The basic property of propositional logic programs that make this procedure to give always a correct answer is that their SLD-Resolution Trees are always finite, so the search for a succesful branch always finishes in two possible states: found or not found.

By contrast, when we will consider SLD-Resolution Trees for logic programs with **first order logic** (CP1), we will see that we do not have a decision procedure, but a semi-decision procedure: if $P \models G$ holds, then its SLD-Resolution Tree will have at least one succesful branch, and with breadth-first search we will be able to find it (and some times also using depth-first search). But when $P \models G$ does not hold, then its SLD-Resolution Tree will have no succesful branches and some of them may be **infinite branches**, that would cause the algorithm not to stop the search in any finite time. But this is not a particular weakness of the algorithm based on SLD-Resolution: there is no decision procedure for first order logic programs, as it happens with the general case of first order formulas.