Formal Methods Notes

Clemens Koza, 01126573

Contents

Revision history

- no “published” version yet.

About this document

This document is based on the Formal Methods in Informatics lecture at Vienna University of Technology; particularly from taking it in the 2023/24 winter term. Corrections and additions are welcome as pull requests at

https://github.com/SillyFreak/tu-wien-software-engineering-notes

This document leaves out several details and was written primarily for me, but I hope it is useful for other people as well.

If you have questions, feel free to reach out on Github. I may at least occasionally be motivated enough to answer questions, extend the document with explanations based on your question, or help you with adding it yourself.

1. Tseitin translation

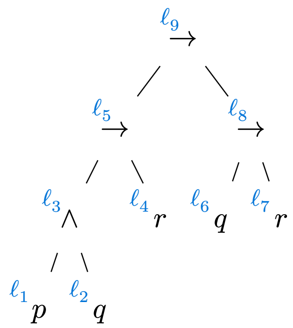
The Tseitin translation's purpose is to transform an arbitrary propositional formula into a conjunction of disjunctions (clauses), which can then be fed to a typical SAT solver. In contrast to the CNF, the resulting formula is linear in the input formula's length.

Following, we will translate the example formula from the lecture slides:

$$\varphi : (p \land q \to r) \to (q \to r)$$

1.1. Labelling of subformula occurrences (SFOs)

In the tree representation of the formula, every subformula receives a new atom. That new atom is set equivalent to the subformula; if the subformula is not itself atomic, the branches of the subformula are themselves replaced by new atoms.



$$\begin{aligned}
&\ell_1 \leftrightarrow p\\
&\ell_2 \leftrightarrow q\\
&\ell_3 \leftrightarrow \ell_1 \land \ell_2\\
&\ell_4 \leftrightarrow r\\
&\ell_5 \leftrightarrow \ell_3 \to \ell_4\\
&\ell_6 \leftrightarrow q\\
&\ell_7 \leftrightarrow r\\
&\ell_8 \leftrightarrow \ell_6 \to \ell_7\\
&\ell_9 \leftrightarrow \ell_7 \to \ell_8
\end{aligned}$$

We can easily see that subformula p is satisfiable iff $(\ell_1 \leftrightarrow p) \land \ell_1$ is satisfiable: substituting the left clause into the right one, we are left with exactly p. This conjunction of the new atom and the corresponding equivalence clause thus captures the satisfiability of the original subformula. Likewise, for a more complex subformula like $p \land q$, we can take $(\ell_1 \leftrightarrow p) \land (\ell_2 \leftrightarrow q) \land (\ell_3 \leftrightarrow \ell_1 \land \ell_2) \land \ell_3$ and get the same result. Applying this to the whole formula φ, we get

$$(\ell_1 \leftrightarrow p) \land \ldots \land (\ell_9 \leftrightarrow \ell_7 \to \ell_8) \land \ell_9$$

This formula has three crucial properties:

- as stated above, it is satisfiable iff φ is satisfiable (although not logically equivalent because it contains new atoms),
- its length is linear in terms of the length of φ, and
- it is essentially *flattened*, which means that the linearity in length will be preserved when transforming it to CNF.

1.2. Generating clauses for the CNF

SAT solvers by convention accept formulas as a set of clauses over a set of atoms. To make the above result usable by such a tool, we need to convert it into this form. Since we have constructed a flat set of equivalence clauses, we can enumerate all possible forms of clauses and how they are translated into CNF. For example:

$$\begin{aligned}
\ell_i \leftrightarrow x &\equiv (\ell_i \to x) &&\land (\ell_i \leftarrow x) &&\equiv (\neg\ell_i \lor x) \land (\ell_i \lor \neg x)\\
\ell_i \leftrightarrow (x \land y) &\equiv (\ell_i \to (x \land y)) &&\land (\ell_i \leftarrow (x \land y)) &&\equiv (\neg\ell_i \lor x) \land (\neg\ell_i \lor y) \land (\ell_i \lor \neg x \lor \neg y)\\
\ell_i \leftrightarrow (x \to y) &\equiv (\ell_i \to (x \to y)) &&\land (\ell_i \leftarrow (x \to y)) &&\equiv (\neg\ell_i \lor \neg x \lor y) \land (\ell_i \lor x) \land (\ell_i \lor \neg y)
\end{aligned}$$

The equivalence of these formulas can be easily verified using a truth table; I find splitting equivalences into two implications useful as an intermediate step to more easily see the CNF clauses. We can now finally translate our formula into a *definitional form* that is in CNF:

$$(\neg \ell_1 \vee p) \wedge (\ell_1 \vee \neg p) \wedge \dots \wedge (\neg \ell_9 \vee \neg \ell_7 \vee \ell_8) \wedge (\ell_9 \vee \ell_7) \wedge (\ell_9 \vee \neg \ell_8) \wedge \ell_9$$

Or, as a set of clauses instead of a formula:

$$\delta(\varphi) = \hat{\delta}(\varphi) \cup \{\ell_9\} = \{\neg \ell_1 \vee p, \ell_1 \vee \neg p, \dots, \neg \ell_9 \vee \neg \ell_7 \vee \ell_8, \ell_9 \vee \ell_7, \ell_9 \vee \neg \ell_8, \ell_9\}$$