Project 22703: SYRF

**Deliverable 3.2.1**

# Use of the Stålmarck Method

---

## 1 Introduction

This deliverable presents the results achieved within task 3.2.1 during SYRF's first year of operation. According to the project program[21], the deliverable consists of four different types of results:

- Developing an interface to the workbench in NP-Tools, NP Technology's verification toolbox;
- Support for expressing properties and requirements in synchronous systems;
- Improving the Stålmarck method for analysis of synchronous systems;
- System verification of supplied cases.

In Section 2 we describe the Stålmarck method and the improvements made to its implementation. In Section 3 we describe the developed support for expressing properties and requirements. In Section 4 we describe the developed workbench interface. In Section 5 we describe the status of the system verification of the supplied case studies.

## 2 Improvements of the Stålmarck Method

The Stålmarck method is a patented method for solving the propositional satisfiability problem in practical applications. Systems are verified by modelling and verifying at the level of propositional logic [4, 7]. Two parallel tracks have been pursued in order to improve the implementation of the Stålmarck method:

- improving the Stålmarck method implementation by adding stronger rules, e.g., *pair-wise comparison rules* (see Section 2.6). These rules have fruitfully been applied, in conjunction with an approach of adding definitions and axioms for the arithmetical part of the proof system, for verification of recursively defined synchronous arithmetic circuits [19] (see Section 2.7). For instance, large (hundreds of bits) multiplier circuits have been verified in this way. The set of so-called simple rules (See Section 2.4) for arithmetical operators has also been made complete;
- by merging the Stålmarck method with a finite domain constraint solver using indexicals [9], the dilemma rule of the Stålmark method has been generalized to finite integer domains.

These two tracks are described to some extent below, though full reports describe them in more detail [10, 19]. Other tracks, beyond the scope of the SYRF project, generalizes the Stålmarck method to propositional linear temporal logic [15] and S4 [16].

## 2.1 About Stålmarck method

Prover[1] is an automated theorem-prover based on a proof system for propositional integer arithmetic. The propositional part of the proof procedure was invented and patented by Stålmarck [20], and has further been described by e.g. Harrison [13]. The method is based on refutation trees similar to Mondadori's proof trees [17, 12], extended with a more efficient set of propagation rules, a restricted use of lemmas and a weak extension of the subformula principle. However, the efficiency of Stålmarck's method is essentially due to the so called saturation procedure, an exhaustive search procedure for shallow proof trees. Stålmarck defines a notion of proof depth, corresponding to the maximum number of simultaneously free assumptions in a proof. The saturation procedure then exhaustively searches for proofs up to a given depth $n$. The saturation procedure for a given $n$ is polynomially time bounded in the length of the input formulas.

The Prover implementation extends Stålmarck's method with propagation rules for integer constraints.

The saturation procedure has an obvious advantage over well-known proof techniques such as BDD based methods [8] in applications with very large formulas and where shallow proofs can be expected. Shallow proofs can often be expected in verification and validation applications as opposed to, for instance, in combinatorial optimization.

## 2.2 Basic Concepts

The basic concepts of Stålmarck's method are the following.

1. The subformula principle, stating that if there exists a proof of a tautology $A$, then there exists a proof using only subformulas or negated subformulas of $A$.
2. A set of *simple rules* on the form $\dfrac{A_1, \ldots, A_n}{B}$ to be interpreted as: given $A_i$, $1 \leq i \leq n$, $B$ follows.
3. An assumption rule, also called the *dilemma rule*, involving a case split of variables.

A proof of a formula $A$ is a refutation of $\neg A$. The proof begins with the set $\{\neg A\}$ and ends with a set of formulas including $B$ and $\neg B$, for some formula $B$.

## 2.3 Subformulas

Compound subformulas are represented by triplets, having the form $x : y \bowtie z$. The *triplet variable* $x$ represents the compund formula obtained by applying the binary operator $\bowtie$ to the *triplet operands* $y$ and $z$ of type boolean or integer. $x$ represents a subformula, and $y$ and $z$ are literals, i.e., either variables (real or subformula variables) or negated variables.

[1]Prover is a trademark of NP Technology. NP-Tools is based on Prover.

Prover input, a propositional formula extended with finite domain integer constraints, is reduced into triplets. The reduction into triplets involves identification of identical subexpressions and normalization of expressions [14]. In order to illustrate the reduction, consider the following tautological formula.

$$(a \wedge (b \vee c)) \Rightarrow ((a \wedge b) \vee (a \wedge c)) \tag{1}$$

This formula is reduced into the subformulas $a_0, \ldots, a_{10}$, given below. Each subformula $a_i$ is given by its index $i$. Subformulas $a_0$ and $a_1$ represent the two truth constants, and subformulas $a_2 \ldots a_4$ represent the real variables.

```
0:  FALSE
1:  TRUE
2:  a
3:  b
4:  c
5:  3 OR 4
6:  2 AND 5
7:  2 AND 3
8:  2 AND 4
9:  7 OR 8
10: 6 -> 9
```

The verification procedure described below is performed on the triplets and the subformula variables.

## 2.4 Simple rules

Simple rules are used for deriving new facts by evaluating triplets. In order to illustrate the simple rules, we consider the rules for the boolean equivalence triplet $x : (y \Leftrightarrow z)$. If y and z have assigned truth values, the truth table may be inspected to deduce the value of x. Otherwise, the first (if any) of the following rules is applied.

```
if x = y    then z := TRUE
if x = ~y   then z := FALSE
if x = z    then y := TRUE
if x = ~z   then y := FALSE
if y = z    then x := TRUE
if y = ~z   then x := FALSE

if x=TRUE   then y := z
if x=FALSE  then y := ~z
if y=TRUE   then x := z
if y=FALSE  then x := ~z
if z=TRUE   then x := y
if z=FALSE  then x := ~y
```

As can be seen above, the proof system is not limited to deduce judgements of the form $A$ *is true* and $A$ *is false*, but includes also judgements of the form $A$ *has the same* *truth value as B* or *A and B have different truth values.* Similarly, for arithmetical triplets, judgements of the form *A is equal to an integer constant*, *A has the same value as B* and *A has the negated value of B* (that is, $A = -B$) are concluded. The simple rules can be seen as natural deduction rules, and are defined for the usual propositional connectives and also the standard arithmetic operators.

Given a set of triplets, applying the simple rules until we reach a fixed point is called a *0-saturation.* If we should encounter a contradictory assignment during 0-saturation we have found a proof.

**Example 1.1** *The valid formula in equation (1) is proven by 0-saturation. Its proof starts with the initial assumption that the the whole formula (represented by triplet number 10) is false. A number of triplet evaluations results in that we derive facts on the form* $i \sim j$*, meaning that variable* $i$ *has the same value as variable* $j$*, and facts on the form* $i \not\sim j$ *meaning that variables* $i$ *and* $j$ *have different values. Finally, we derive a contradiction* ($\perp$)*.*

$$
\begin{array}{c}
\boxed{10 \not\sim 1} \\
6 \sim 1 \\
9 \not\sim 1 \\
7 \not\sim 1 \\
8 \not\sim 1 \\
2 \sim 1 \\
5 \sim 1 \\
3 \not\sim 1 \\
4 \not\sim 1 \\
\perp
\end{array}
$$

In general, using only 0-saturation will not be enough in order to prove a formula. In that case the dilemma rule is needed.

## 2.5 The Dilemma Rule

The Dilemma rule performs case splitting of variables. The consequences of assuming a real variable or a triplet variable to be true and false are derived into two sets of facts $\Delta_T$ and $\Delta_F$ respectively, using 0-saturation. If both $\Delta_T$ and $\Delta_F$ contain a contradiction we have found a proof. Otherwise, we combine the results by adding the intersection $\Delta_T \cap \Delta_F$. The intersection obviously holds no matter the value of the variable that was case split. The dilemma rule is repeatedly applied for all variables, real or triplet variables, until either a contradiction has been found in both branches or a fixed point has been reached, i.e., until no new facts can be generated by an application of the dilemma rule. Applying the following schema for all real and subformula variables $a_1, \ldots, a_n$ until a fixed point is reached or a proof has been found is a complete 1-saturation.

$$
\begin{array}{c}
\Delta \\
\hline
\begin{array}{lr}
\Delta \cup \neg a_1 & \Delta \cup a_1 \\
\vdots & \vdots \\
\Delta_{1_F} & \Delta_{1_T}
\end{array} \\
\hline
\Delta_{1_F} \cap \Delta_{1_T} \\
\hline
\begin{array}{lr}
\Delta \cup \neg a_2 & \Delta \cup a_2 \\
\vdots & \vdots \\
\Delta_{2_F} & \Delta_{2_T}
\end{array} \\
\hline
\Delta_{2_F} \cap \Delta_{2_T} \\
\vdots \\
\hline
\begin{array}{lr}
\Delta \cup \neg a_n & \Delta \cup a_n \\
\vdots & \vdots \\
\Delta_{n_F} & \Delta_{n_T}
\end{array} \\
\hline
\Delta_{n_F} \cap \Delta_{n_T}
\end{array}
$$

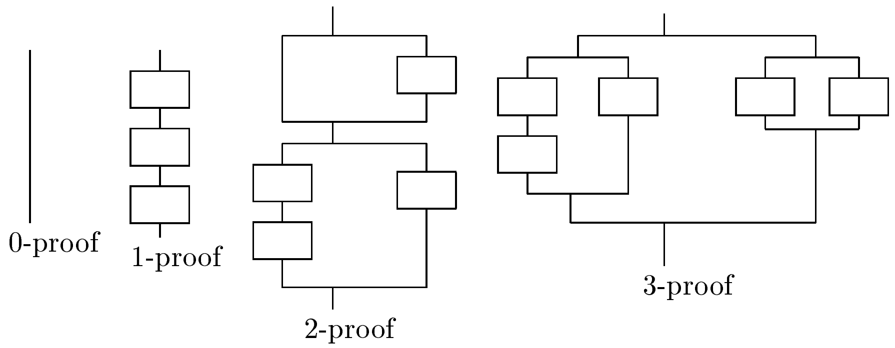
If a 1-saturation does not reach a contradiction, then 2-saturation may be attempted, involving case splits over all *pairs* of variables. If that does not succeed, 3-saturation may be attempted and so on. Similarly, one can $n$-saturate for any natural number $n$, case splitting over $n$-tuples.

**Example 1.2** *The formula in equation (1), where $\Rightarrow$ is replaced by $\Leftarrow$, cannot be proven by 0-saturation. Below a 1-saturation proof of the formula is given.*

$$
\begin{array}{c}
\boxed{10 \not\sim 1} \\
9 \sim 1 \\
6 \not\sim 1 \\
\hline
\begin{array}{lcr}
\boxed{2 \not\sim 1} & \qquad\qquad & \boxed{2 \sim 1} \\
7 \not\sim 1 & & 5 \not\sim 1 \\
8 \sim 1 & & 3 \not\sim 1 \\
\perp & & 4 \not\sim 1 \\
| & & 7 \not\sim 1 \\
| & & 8 \sim 1 \\
 & & \perp
\end{array} \\
\hline
\perp
\end{array}
$$

### 2.5.1 Degree of hardness

The degree of a proof $\prod$ is the maximum number of simultaneously free assumptions used in $\prod$, or equivalently, the number of nested Dilemma rule applications. Below, the structure for a few $n$-proofs are illustrated, where horizontal lines indicate opening and closing of a dilemma rule, and vertical lines denote application of the simple rules.




Obeying the subformula principle, the valid formulas (tautologies) can be divided into hardness degrees according to the following definition: *$A$ has degree $n$ of hardness if and only if there is an $n$-proof of $A$ and there is no $m$-proof of $A$ for any $m < n$.*

It can be shown that if a tautology $A$ with size (number of connectives) $|A|$ can be proven with $n$-saturation, then there is a proof of size $O(|A|^{n+1})$, where size refers to the number of rule applications. Furthermore, it can be shown that the execution time of the saturation procedure to find this proof is bounded by $O(|A|^{2n+1})$.

## 2.6 Pairwise Comparison Rules

The simple rules deduce judgements based on the values of and relations between the operands of a single binary operator. During 0-saturation, *pairwise comparison rules* can optionally be applied as well. The pairwise comparison rules deduce judgements based on the relations between a pair of binary operators and the values of and relations between their operands. More precisely, if a triple $t$ is evaluated and none of the ordinary simple rules apply, pairwise comparison rules are applied to $t$ and all $t_i$, where $t_i$ is a triplet with the same operator and the same operands (possibly rearranged or negated) as $t$. The pairwise comparison rules are applied together with the ordinary simple rules until a fixed point is reached.

Applying the pairwise comparison rules involves search for the triplet $t_i$, which can lead to a degrading of efficiency when 0-saturation is performed. However, this may be worthwhile since the rules can reduce the degree of saturation needed to perform a proof, and even, for the arithmetic part of the proof system, add proof power. The added proof power by pairwise comparison rules has been essential in the verification of the arithmetical circuits in [19]. Another observation of using pairwise comparison rules has been a noticeable more efficient model search.

Currently, pairwise comparison rules are implemented for implication and the less than operator. One could imagine pairwise comparison rules also for other operators, for instance, the plus operator.

### 2.6.1 Less Than operator

The pairwise comparison rules for the less than operator are, assuming the two triplets $x \Leftrightarrow y < z$ and $x1 \Leftrightarrow z < y$ (or $x1 \Leftrightarrow -y < -z$), the following.

```
x=TRUE  => x1 := FALSE
x1=TRUE => x  := FALSE
x=x1    => y  := z
```

### 2.6.2 Implication operator

The pairwise comparison rules for the implication operator are, assuming the two triplets $x \Leftrightarrow y \Rightarrow z$ and `ti` (defined in (1)—(3) below), the following.

```
(1) ti: x1 =  z  ->  y       or ti: x1 = ~y -> ~z
        x  =  x1 =>  y :=  z
        x  = ~x1 =>  y := ~z
(2) ti: x1 =  y  -> ~z       or ti: x1 = z -> ~y
        x  =  x1 =>  y := FALSE
        x  = ~x1 =>  y := TRUE
(3) ti: x1 = ~y  ->  z       or ti: x1 = ~z -> y
        x  =  x1 =>  z := TRUE
        x  = ~x1 =>  z := FALSE
```

## 2.7 Strengthening the Proof System

Two basic ideas for strengthening the Stålmarck's proof system by adding information to the set of subformulas and axioms generated by default are:

1. By adding extra definitions, triplets, we extend the subformula principle, and thereby are able to prove more theorems. This approach is similar to, for instance, extended Frege systems or extensions of resolution.

2. By adding axioms, that is, subformulas with given values, we are able to prove more theorems.

During SYRF's first year of operation, we have developed an hierarchical approach for verification of synchronous arithmetical circuits [19]. The approach has been made feasible by adding definition and axioms for the arithmetical part of the proof system in conjunction with the newly developed pairwise comparison rules. The axioms express, e.g., distributivity of multiplication and addition. The axioms are deduced based on the shape of the the triplet universe. The added axioms cannot be deduced by $n$-saturation for any $n$ without adding information of the domains of integer variables. We do not want to add domains since we want to reason about integer values of arbitrary domain (or bit-width), and also since adding domain information currently has a degrading effect on efficiency (for large domains).

This approach and the results are described in [19] (currently draft version).

## 2.8 NP(fd)

The NP(fd) algorithm is a hybrid between the Stålmarck method and a finite domain constraint solver using indexicals [9]. NP(fd) is a sound and complete proof system for a propositional language of finite domain constraints. The system includes simple rules for propagating logical consequences and a dilemma rule for assumption-based reasoning. More specifically, the dilemma rule (see Section 2.5) has been generalised into finite domains:

$$
\begin{array}{cccc}
\multicolumn{4}{c}{\Delta} \\
\hline
\Delta \cup A = A_1 & \Delta \cup A = A_2 & \ldots & \Delta \cup A = A_n \\
\vdots & \vdots & \vdots & \vdots \\
\Delta_1 & \Delta_2 & \ldots & \Delta_n \\
\hline
\multicolumn{4}{c}{\Delta_1 \cap \Delta_2 \cap \ldots \cap \Delta_n}
\end{array}
$$

The NP(fd) implementation [11] is described in [10], as are some verification benchmark results from the railway signalling domain and from combinatorial optimization. The first type of benchmark results were somewhat biased towards the Prover implementation, since the benchmarks were isolated formulas containing, in some sense, only uncomplicated finite domain constraints. Therefore NP(fd) need to be further evaluated for verification of system containing more complicated finite domain constraints. Currently, NP(fd) is evaluated for verification of SAAB's user case [6] and constraint satisfaction problems.

# 3 Support for Expressing Requirements

The modelling language of Prover has been extended to enumerated types and a construct for finite integer domains. This makes modelling and requirement formulation more natural, and facilitates analysis of synchronous systems. The added features are described briefly below.

## 3.1 Enumerated types

The input language of Prover and NP-Tools has been extended to support enumerated types. This makes system modelling easier and analysis results are easier to interpret, since they are given using the enumerated type.

In this context, NP Technology performed the modelling and verification of NASA's SAFER system, making heavy use of enumerated types. The system was originally modelled and verified by NASA using PVS [18] and IFAD made, based on the PVS model, a model of the system in VDM-SL using their VDM-SL toolbox. The VDM-SL specification was modelled in NP-Tools using the enumerated type construct, and all safety requirements could be verified automatically [1]. Without support for enumerated types, the modelling would have been a very effort-demanding task, and would have resulted in a less natural model.

## 3.2 Finite domains

Previously, a domain 1 to 9 for an integer variable $a$ was input to Prover as $a = 1 \vee a = 2 \vee \ldots \vee a = 9$. A syntactic and more user-friendly feature now allows for giving domains as intervals and/or points in an interval instead: $a\ in\ 1 \ldots 9$. If an analysis does not result in a constant value for an arithmetical variable, its *possible domain* is displayed. For instance, 1-saturating the expression $a\ MOD\ 2 = 1$ together with the domain for $a$ would result in $a\ in\ 1, 3, 5, 7, 9$. Internally in the Prover system, domains are expanded into the original form (with equality and disjunction).

When analysing some systems, e.g., SAAB's case study, it is not desirable to expand very large domains, since this has a degrading effect on the performance of automatic proof. Furthermore, it is in many applications the case that safety requirements do not depend on variables with very large domains. By giving a limit on the maximum domain size to expand, only domains containing fewer elements than specified by the limit are expanded. In the modelling made by the automatic LSA to NP-Tools translator, SAAB's case study (not the "toy version") contains some 10k propositional sub-formulas. Furthermore, it contains some tens of variables with domains $-32000 \ldots 32000$. Generating the complete domain for these variables would generate almost a million triplet subformulas. By turning off the generation of the complete domain for these variables, analyses become feasible, and safety requirement that are independent of the values of variables with large domains can be proven.

Models generated when using a domain size limit may however result in so-called *false negatives*; models that are not real models can be presented, e.g., a model where both $A > B$ and $A < B$ are true.

# 4 Workbench Interface

## 4.1 ASA tool Interface

SAAB's user case, a synchronous model of the air supply system in the military attack plane JAS, is available as a model in the CASE tool ASA (by Verilog). ASA is based on the LSA language. NP Technology's has developed a translator from (synchronous) LSA to NP-Tools [5], which is used for analyses of the SAAB case study. The translator tool from LSA to NP-Tools has been extended to include the enumerated types and the new domain syntax.

## 4.2 Lustre Interface

Pascal Raymond of Verimag has developed a translator from Lustre to NP-Tools. Together with Verimag a few extensions have been identified, and will be implemented.

# 5 System Verification

The system verification studies done during SYRF's first year of operation include:

- Analysis of a switch software case study in Esterel [2]. During 1997, a direct contact with Lucent has been established, and the case study has been further analysed.

- SAAB's case study has been analysed with respect to a set of safety requirements that have been formulated together with SAAB during 1997. Both Prover and NP(fd) are used for verification of this study. The results so far are reported in [6];

- A number of recursively defined hardware circuits, e.g., multipliers and adders, have been verified. Circuit formulas have been generated from the functional language Haskell [19], and been verified using Prover's new stronger rules.

# References


[1] S. Agerholm and P. G. Larsen. Modeling and validating safer in vdm-sl. In M. Holloway, editor, *Fourth NASA Langley Formal Methods Workshop*. NASA, September 1997.

[2] H. Ågren. Case Study: Verifying Switch Software. Technical Report U-96040, Logikkonsult NP AB, 1996.

[3] O. Åkerlund. Resultat av tillämpning av formella metoder vid analyser av Demonstrationssystem och delar av Luftsystemet i fpl 39. Technical Report FN-97.008, Saab Military Aircraft, Jan. 1997.

[4] R. Alur and T. A. Henzinger. Real-time Logics: Complexity and Expressiveness. *Information and Computing*, 104(1):35–77, 1993.

[5] H. Bolinder. lsa2npc 1.2 LSA Translator Description. Technical Report NPT-01-07-05-01, Issue 2 Rev 1, Logikkonsult NP AB, 1997.

[6] A. Borälv and C. J. Block. Analysis of a model of the Air Supply system in JAS. Technical Report U-9702-02, Logikkonsult NP AB, 1997. Draft.

[7] A. Borälv and G. Stålmarck. *Industrial-Strength Formal Methods*, chapter Prover Technology. Academic Press, International Series in Formal Methods. 1998. In preparation.

[8] R. Bryant. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. Technical report, Carnegie Mellon University, Nov. 1995.

[9] B. Carlsson. *Compiling and executing finite domain constraints*. PhD thesis, Computing Science Department, Uppsala University, 1995. Also available in SICS Dissertation Series.

[10] B. Carlsson, M. Carlsson, and G. Stålmarck. NP(FD): A proof System for Finite Domain Constraints. Technical report, Xerox Parc, Swedish Institute of Computer Science and Logikkonsult, 1997. Short version of this paper has been submitted.

[11] M. Carlsson. *Enhanced Finite Domain Constraints in SICStus Prolog*. Swedish Institute of Computer Science, 1997.

[12] M. D'Agostino. *Investigations into the Complexity of some Propositional Calculi*. PhD thesis, Oxford University, 1990. Technical Monograph PRG-88.

[13] J. Harrison. The Stålmarck Method as a HOL Derived Rule. In *Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, pages 221–234. Springer Verlag, 1996.

[14] A. Jonasson. Normalization of Expressions in prover3.1. Technical Report U-96029, Logikkonsult NP AB, Apr. 1996. Internal report.

[15] A. Jonasson. pl; Evaluation Report on a Temporal Prover. Technical Report U-96054, Logikkonsult NP AB, Sept. 1996.

[16] P. Larsson. Ett effektivt bevissystem för S4. Master's thesis, The Department of Philosophy, Stockholm University, 1996.

[17] M. Mondadori. Classical analytical deduction (part I and II). Annali dell'Universita de Ferrara, Sezione III, 1988.

[18] National Aeronautics and Space Administration. Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems. Volume II: A Practitioners's Companion. Technical Report NASA-GB-001-97, Washington, DC 20546, USA, May 1997.

[19] M. Sheeran and A. Borälv. Verification of Arithmetic Circuits using a Functional Language and Stålmarck's method. Technical report, Chalmers University of Technology and Prover Technology, Oct. 1997. To be submitted.

[20] G. Stålmarck. A System for Determining Propositional Logic Theorems by Applying Values and Rules to Triplets that are Generated from a Formula, 1989. Swedish Patent No. 467 076 (approved 1992), U.S. Patent No. 5 276 897 (1994), European Patent No. 0403 454 (1995).

[21] Esprit project 22703 SYnchronous Reactive Formalisms SYRF Annex I Project Programme, 1996. http://www.imag.fr/VERIMAG/SYNCHRONE/SYRF/.