Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

How many times should a program be unfolded for proving invariant properties?

Jan Mikáč and Paul Caspi

Report n^o TR-2004-9

March 2004

How many times should a program be unfolded for proving invariant properties?

Jan Mikáč and Paul Caspi

March 2004

Abstract

This report proposes an improvement of automation for inductive proofs of invariant properties: a minimum number of unfoldings to perform *a priori* is proposed. The minimality comes from a characterisation of the pairs system–property where the property holds and can be proved by induction. Such a characterisation relies on several proof-conserving changes of variables. A real-world example of proof is given as an illustration.

Keywords: induction, invariant, unfolding, abstraction

Reviewers: Y. Lakhnech

Notes: This work has been partially supported by the European Commission through IST project Next-TTA.

How to cite this report:

```
@techreport { ,
title = { How many times should a program be unfolded for proving invariant properties?},
authors = { Jan Mikáč and Paul Caspi},
institution = { Verimag Technical Report },
number = {TR-2004-9},
year = { 2004},
note = { }
}
```

1 Introduction

Reactive programs implement dynamical systems which continuously interact with some environment, by getting inputs and providing outputs. When critical systems are considered, it is important to prove invariant properties of such systems. In some cases, when systems are finite-state ones, model-checking is the technique of choice which allows performing these proofs in a fully automatic way. Examples of popular model-checkers are [7, 10]. When systems are not finite state, however, some user interaction is needed. Two approaches can be followed then, model-checking by abstraction [2] and, more generally abstract interpretation [4], or induction. Induction is the main technique used in interactive provers [8, 15] and in correct by construction programming [1] and is the technique considered in this paper.

Induction is proved to be a complete technique, provided an adequate strong enough auxiliary invariant is found. Several techniques are used for finding such an invariant. For instance, some automatic generation of invariants can be performed [3] but it is never sure that these guesses are the right ones. Another popular technique consists of simply examining the reasons why the proof by induction fails: just looking at the unsuccessful proof obligations can give some hint on how to strengthen the property we want to prove.

For instance, let us consider the following program (expressed in a functional style) which computes iteratively the Fibonnacci sequence:

$$\begin{array}{rcl} f(0) & = & 1 \\ f(1) & = & 1 \\ f(n+2) & = & f(n+1)+f(n) \end{array}$$

and assume we want to prove that every element of this sequence is positive. The usual proof by induction rule on sequences :

$$\frac{H \vdash P(0), \quad H \vdash \forall n, P(n) \Rightarrow P(n+1)}{H \vdash \forall n, P(n)}$$

yields two proof obligations:

$$\frac{}{1>0} \qquad \frac{f(n+2)>0}{f(n+2)+f(n+1)>0}$$

The first one is obvious but the second one cannot be proved. Examining this failure leads to "strengthen" the property by using the rule:

$$\frac{H \vdash P(0), \quad H \vdash P(1), \quad H \vdash \forall n, P(n) \wedge P(n+1) \Rightarrow P(n+1) \wedge P(n+2)}{H \vdash \forall n, P(n) \wedge P(n+1)}$$

which simplifies to:

$$\frac{H \vdash P(0), \quad H \vdash P(1), \quad H \vdash \forall n, P(n) \wedge P(n+1) \Rightarrow P(n+2)}{H \vdash \forall n, P(n) \wedge P(n+1)}$$

and yields the three obvious obligation rules

$$\frac{}{1>0} \qquad \frac{}{1>0} \qquad \frac{f(n+2)>0 \quad f(n+3)>0}{f(n+3)>0 \wedge f(n+3)+f(n+2)>0}$$

The question raised here is thus: wasn't this result obvious and couldn't we have skipped the first step so as to jump directly to the second one, thus getting a "more automatic" proof? While the iterative approach seems quite popular (see for instance [13]), this idea of a direct jump to the supposed right number of iterations seems to be less often considered. Yet it is found in [14] where it is suggested that:

> "The length of the longest serial connection of latches (or other delay elements) is usually a lower bound on the number of iterations needed".

However, this suggestion is given without justification.

Clearly, this question of the number of required iterations doesn't have a definite answer. The strongest auxiliary invariant may require unbounded unfolding because it results from a least fix-point computation. On the contrary, even complex programs can only need a single unfolding if the property is simple enough. However, we will provide here some evidence that the number of memories in the program is a sensible value for this iteration number.

The presentation is organised as follows: in a second section, we provide a formalisation of the problem and derive the solution in section 3. An example of application, based on the Gloups tool [5], a PVS proof obligation generator for Lustre programs [6], is presented in section 4.

2 Formalisation

In this section, we shall introduce notations for the systems and properties that we consider and we shall formally state the problem we are interested in. We shall also give some very general properties of space-changing which apply to our systems.

2.1 Programs, Invariant Properties and Induction

We consider only closed programs (most interesting reactive system properties only hold when the program is considered in closed loop with its environment and, thus, have no inputs nor outputs).

A program is then described by:

- $S = \prod_{i=1}^{n} S_i$, a state set, composed of a Cartesian product of the state spaces associated with the state variables of the program.
- I_0, a set of possible initial states: $\emptyset \subset I_0 \subseteq S$
- $T \subseteq (S \times S)$, a transition relation. We assume here that the the relation is total, *i.e.*, the domain of the relation $dom(T)$ is equal to S. This means that we do not aim at studying dead-lock properties.

Then the property we want to prove is given by the predicate $P \subseteq S$.

The strongest invariant of the system I_∞ is the least solution of the fix-point equation:

$$I_\infty = I_0 \cup T(I_\infty)$$

where $T(X) = \{y \mid \exists x \in X.\ T(x, y)\}$. Equivalently,

$$I_\infty = \cap\{X \mid I_0 \cup T(X) \subseteq X\} \tag{1}$$

The property P is an invariant (holds on (S, I_0, T)) if

$$I_\infty \subseteq P \tag{2}$$

and the induction proof principle says that:

$$I_0 \cup T(P) \subseteq P \tag{3}$$

proves (2). In the following, we shall use a short-hand notation for a system: the tuple (S, I_0, T, P). But before exactly stating the problem we are interested in, we shall introduce some more notions.

2.2 Co-images and induction

Given the total relation $T \subseteq S \times S$ we can consider its co-image:

$$T^-(Y) = \cup\{X \mid T(X) \subseteq Y\}$$

Then it is well-known that images and co-images verify the so-called "Galois connection property":

$$\begin{array}{rcl} Id & \subseteq & T^- \circ T \\ T \circ T^- & = & Id \end{array} \tag{4}$$

where Id is the identity mapping.

In the case of a system (S, I_0, T, P), we have

- if $I_\infty \subseteq P$
- · since $I_\infty = I_0 \cup T(I_\infty)$ we have also $T(I_\infty) \subseteq P$
- · which gives $I_\infty \subseteq T^-(P)$
- ▷ hence $I_\infty \subseteq P \cap T^-(P)$
- obviously $I_\infty \subseteq P \cap T^-(P)$ gives $I_\infty \subseteq P$

Thus, the problem (S, I_0, T, P) is *logically* equivalent to the problem $(S, I_0, T, P \cap T^-(P))$ or any other $(S, I_0, T, \bigcap_{i=0}^{m-1} T^{-i}(P))$. However, these problems are not equivalent with respect to *proofs*: our aim is to give a minimum value of m for each class of problems, such that an inductive proof has "better chances" to work than for any smaller m.

In order to achieve this goal, we shall now give a characterisation of the systems (S, I_0, T, P) where (2) holds and can be proved by an m-times unfolded instance of (3).

3 Characterisation of inductively provable invariants

Our main result relies on abstractions given by some change of variable. This is why we shall introduce change of variables and some of their properties, then we shall consider some particular cases and we shall conclude on unfolding of inductive proofs.

3.1 Abstraction and induction

Given a state-space S, a change of variable is any mapping $G : S \to S'$. Such a mapping can be naturally lifted to sets and, then admits a co-image G^- and satisfies the Galois connection properties.

A change of variables G generates a new system:

$$\begin{array}{rcll} G(S, I_0, T, P) & = & (S', I'_0, T', P') & (5) \\ \text{where } I'_0 & = & G(I_0) & (6) \\ P' & = & G(P) & (7) \\ T' & = & G \circ T \circ G^- & (8) \end{array}$$

In general, a change of variable doesn't preserve properties. Yet, in [4, 9] was proved the fundamental theorem of abstraction:

Proposition 1 *If*

$$P = G^- \circ G(P) \tag{9}$$

then (S, I_0, T, P) *holds if* $G(S, I_0, T, P)$ *holds*

Moreover one can easily prove (using (4) and (9)) that for any set $X \subseteq S$

$$\begin{array}{rcll} G(X) \subseteq G(P) & \Leftrightarrow & X \subseteq P & (10) \\ \text{and } T'(G(P)) \subseteq G(P) & \Leftrightarrow & T(P) \subseteq P & (11) \end{array}$$

As a consequence, we have the following theorem:

Proposition 2 (Inductive proof conservation) *Given $P = G^- \circ G(P)$, there is an inductive proof of (S, I_0, T, P) if and only if there is an inductive proof of (S', I'_0, T', P') .*

Proof

	there is an inductive proof of (S, I_0, T, P)	
$\equiv$	$I_0 \subseteq P$ and $T(P) \subseteq P$	
$\equiv$	$G(I_0) \subseteq G(P)$ and $T(P) \subseteq P$	by (10)
$\equiv$	$G(I_0) \subseteq G(P)$ and $T'(G(P)) \subseteq G(P)$	by (11)
$\equiv$	$I'_0 \subseteq P'$ and $T'(P') \subseteq P'$	
$\equiv$	there is an inductive proof of (S', I'_0, T', P')	

□

3.2 Changes of variables corresponding to unfolded inductions

We show here that (unfolded) induction corresponds to changing variables to the truth value of the property we want to prove and their unfoldings.

3.2.1 Case of P

Given a system (S, I_0, T, P), we define G_1 such that

$$\begin{array}{rcl} G_1 & : & S \to Bool \\ G_1(x) & = & (x \in P) \end{array}$$

It is obvious that $G_1^- \circ G_1(P) = P$, since $\cup\{X \subseteq S \mid G_1(X) \subseteq G_1(P)\} = P$. Therefore, P and G_1 verify the hypothesis of proposition 2.

Furthermore in the new system, properties can be proved by induction:

Proposition 3 *$G_1(S, I_0, T, P)$ holds if and only if it is provable by induction.*

Proof On the one hand, induction is sound, so that

$$I'_0 \subseteq P' \text{ and } T'(P') \subseteq P' \quad \text{imply} \quad I'_\infty = \cap\{X \mid I'_0 \cup T'(X) \subseteq X\} \subseteq P'$$

On the other hand, if $I'_\infty \subseteq P'$ then

- since $I'_\infty = I'_0 \cup T'(I'_\infty)$ we have $\underline{I'_0 \subseteq P'}$
- as $\emptyset \subset I_0$ then necessarily $\emptyset \subset I'_0$
- ▷ thus $\emptyset \subset I'_0 \subseteq P' \subseteq \{\mathtt{t}\}$ which gives $I'_0 = \{\mathtt{t}\} = P'$
- since $I'_\infty = I'_0 \cup T'(I'_\infty) = I'_0 \cup T'(I'_0 \cup T'(I'_\infty))$
  and $T'(I'_0 \cup T'(I'_\infty)) = T'(I'_0) \cup T' \circ T'(I'_\infty)$
  we have also $T'(I'_0) \subseteq P'$
- ▷ in other words, we have $\underline{T'(P') \subseteq P'}$

□

Hence the following proposition gives a characterisation of systems where invariant properties can be proved by 1-time unfolded induction:

Proposition 4 *(S, I_0, T, P) is provable by induction if and only if $G_1(S, I_0, T, P)$ holds.*

Proof

	P is provable by induction	
$\equiv$	$I_0 \subseteq P$ and $T(P) \subseteq P$	
$\equiv$	$I'_0 \subseteq P'$ and $T'(P') \subseteq P'$	(prop. 2)
$\equiv$	$I'_\infty \subseteq P'$	(prop. 3)

□

3.2.2 Case of $P \cap T^-(P)$

Given a system (S, I_0, T, P), we define G_2 by associating to each state, the truth value of the property at this state and its first backward unfolding through the transition relation:

$$\begin{array}{rcl} G_2 & : & S \to Bool^2 \\ G_2(x) & = & (x \in P, x \in T^-(P)) \end{array}$$

It is easy to check that G_2 verifies $G_2^- \circ G_2(P \cap T^-(P)) = P \cap T^-(P)$, so that the hypothesis of proposition 2 holds. Furthermore in the new system, properties can be proved by induction:

Proposition 5 *$G_2(S, I_0, T, P)$ holds if and only if it is provable by induction.*

Proof On the one hand, induction is sound:

$$\begin{array}{l} I'_0 \subseteq P' \cap T'^-(P') \text{ and } T'(P' \cap T'^-(P')) \subseteq P' \cap T'^-(P') \\ \text{imply } I'_\infty \subseteq P' \cap T'^-(P') \end{array}$$

On the other hand, if $I'_\infty \subseteq P' \cap T'^-(P')$ then

- since $I'_\infty = I'_0 \cup T'(I'_\infty)$ we have $\underline{I'_0 \subseteq P' \cap T'^-(P')}$
- as $\emptyset \subset I_0$ then also $\emptyset \subset I'_0$
- ▷ finally $\emptyset \subset I'_0 \subseteq P' \cap T'^-(P') \subseteq \{(\mathtt{t}, \mathtt{t})\}$
  hence $I'_0 = P' \cap T'^-(P') = \{(\mathtt{t}, \mathtt{t})\}$
- since $I'_\infty = I'_0 \cup T'(I'_\infty) = I'_0 \cup T'(I'_0 \cup T'(I'_\infty))$
  and $T'(I'_0 \cup T'(I'_\infty)) = T'(I'_0) \cup T' \circ T'(I'_\infty)$
  we have also $T'(I'_0) \subseteq P' \cap T'^-(P')$
- ▷ in other words, we have $\underline{T'(P' \cap T'^-(P')) \subseteq P' \cap T'^-(P')}$

□

Hence the characterisation follows:

Proposition 6 *$(S, I_0, T, P \cap T^-(P))$ is provable by induction if and only if $G_2(S, I_0, T, P)$ holds.*

Proof

$$\begin{array}{rll} & P \cap T^-(P) \text{ is provable by induction} & \\ \equiv & I_0 \subseteq P \cap T^-(P) \text{ and } T(P \cap T^-(P)) \subseteq P \cap T^-(P) & \\ \equiv & I'_0 \subseteq P' \cap T'^-(P') \text{ and } T'(P' \cap T'^-(P')) \subseteq P' \cap T'^-(P') & \text{(Theorem 2)} \\ \equiv & I'_\infty \subseteq P' \cap T'^-(P') & \text{(Proposition 5)} \end{array}$$

□

3.3 Putting things together

First, we can generalise (by induction!) the previous two results:

Proposition 7 *Given the change of variable*

$$G_m(x) = (x \in P, \ldots x \in T^{-(m-1)}(P))$$

$(S, I_0, T, \bigcap_{i=0}^{m-1} T^{-i}(P))$ is provable by induction if and only if $G_m(S, I_0, T, P)$ holds.

Then, knowing that the state space $S = \prod_{i=1}^{n} S_i$ is a Cartesian product of the state spaces associated with state variables of some program, we can propose the following claim:

Claim 1 *The minimum requirement for an automatic induction-based proof method is to take $m = n$.*

Justification Given that S is of dimension n, taking $m < n$ implies that there is no injective function from S into $B_m = Bool^m$. In particular, G_m is not injective and thus several systems have the same image by G_m. Hence, proving a property by (m-times unfolded) induction on one of these systems would prove the same property on all the systems that have the same image by G_m. Such a proof method would not be specific of one system, but rather of a whole class of systems.

However, we need a proof method as specific of the underlying system as possible: the less systems correspond to a given image by G_m, the more the proof takes care of the dynamics of those systems and the more chances we have to succeed in the proof. Therefore, we need $m \geq n$, the minimum of which is $m = n$.
□

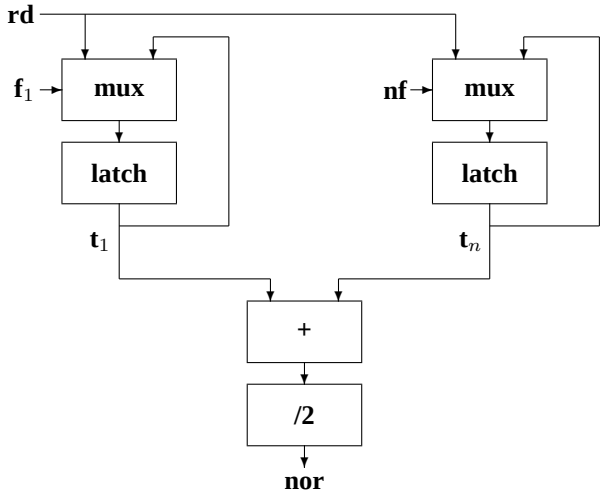
In other words, our claim states that an inductive proof should be unfolded at least the number-of-state-variables times in order to have good chances to work. Of course, some systems and properties have inductive proofs with strictly less unfoldings: this means that due to some algebraic, symmetry or other reasons, the given property holds on a whole class of systems, a class which is characterised by its unique image by some G_k. Yet, the number of state variables in a program is not an intrinsic notion. It may be the case that redundant state variables are used and also that some state variables don't participate (even transitively) in the computation of the property. In using our rule, one should take care of not using "too many" state variables, though we know that minimising the number of state variables is a difficult problem.

As an illustration of the use of unfolded induction, we present in the next section a real-world example of a proof of equivalence of two circuits.

4 Example of the synchroniser

Miner and Johnson [11] proposed in 1996 a co-induction based proof for the problem of equivalence of the outputs of two circuits which realize a fault-tolerant clock synchroniser. Their proof was far from being automatic: they had to construct a bisimulation of the two circuits "by hand". Our aim is to propose a more automatic proof based on induction: the idea is that the first steps (unfolding and inducting) are performed by a machine, the user being charged only with the concluding step. In the following, we shall present the problem in more details, then we shall give a modelling of it and finally we shall discuss the proof itself.

4.1 Presentation of the two circuits



Standard circuit

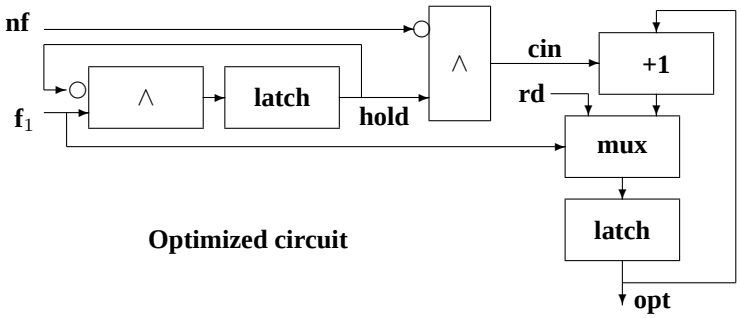
The standard circuit receives three boolean signals ($\mathbf{f}_1$ and **nf** and **R**eset which is not figured on the schema) and one integer signal **rd**. The output is the integer signal **nor**.

All signals are down (or 0 for the integers) if **R**eset is true. Furthemore, **nf** becomes true after $\mathbf{f}_1$ does.

The standard circuit works at follows: **rd** is increased by 1 at every time unit. When x hardware clocks have sent their synchronisation signals, $\mathbf{f}_1$ becomes true. When $x + a$ clocks have sent their signals, **nf** becomes true too, and the output **nor** contains a new system-wide time, to be broadcasted back to the

clocks. At the end of the synchronisation cycle, **R**eset becomes true and the cycle reiterates. Such a circuit can be used for synchronising n hardware clocks (with $n \geq 2x + a - 1$) and tolerates $x - 1$ faults.

The optimised circuit receives the same set of inputs as the standard one and produces a unique output **opt**. The new circuit calculates the final output progressively between the raising edge of $\mathbf{f}_1$ and that of **nf**, rather than doing all calculations when **nf** raises. Thanks to this, an adder and a divider can be saved; only an adder+1 and a few logical gates are needed. Thus, the optimisation is in fact trading space for time.



4.2 Our model

Our aim is to prove that the two circuits give the same output (**nor**=**opt**) using an induction-based technique. For this, we "translate" the problem into a set of fix-point equations over the (infinite) sequences R, f_1, nf, out, opt, t_1, t_n, nor, cin and rd where the scalar operators $+$, $\wedge$, $if\ then\ else\ldots$ are lifted to sequences in a point-wise manner:

$$\begin{array}{rcl} nor & = & (t_1 + t_n)\ div\ 2 \\ t_1 & = & 0.(if\ R\ then\ 0\ else\ (if\ f_1\ then\ t_1\ else\ tl(rd))) \\ t_n & = & 0.(if\ R\ then\ 0\ else\ (if\ nf\ then\ t_n\ else\ tl(rd))) \\ opt & = & 0.(if\ R\ then\ 0\ else \\ & & \quad (if\ f_1\ then\ opt + (if\ cin\ then\ 1\ else\ 0)\ else\ tl(rd)) \\ cin & = & \mathtt{f}.(hold \wedge \neg tl(nf)) \\ hold & = & \mathtt{f}.(\neg tl(R) \wedge tl(f_1) \wedge \neg hold) \end{array}$$

To these equations, we add the following properties, which express the constraints imposed on environment:

$$\begin{array}{l} rd = 0.(if\ R\ then\ 0\ else\ rd + 1) \\ nf \Rightarrow f_1 \\ (R \wedge \neg f_1).\mathtt{t} \\ \mathtt{t}.(R \vee (nf \Rightarrow tl(nf))) \end{array}$$

In fact, such a model can be exactly represented by a Lustre [6] program:

```
node Synchro(R,f1,nf: bool) returns (nor, opt: int);
  var hold, cin: bool;   t1, tn, rd: int ;
  let
    nor = (t1 + tn) div 2 ;
    t1 = 0 → if R then 0 else (if f1 then pre t1 else rd) ;
    tn = 0 → if R then 0 else (if nf then pre tn else rd) ;
    hold = false → (not R) and f1 and (not pre hold) ;
    cin = false → (pre hold) and not nf ;
    opt = 0 → if R then 0 else
          (if f1 then (if cin then 1 else 0)+pre opt else rd);
    assert rd = 0 → if R then 0 else (pre rd + 1) ;
    assert R and not f1 → true;
    assert nf ⇒ f1;
    assert true → R or (pre nf ⇒ nf) ;
  tel;
```

Having a Lustre model allows us to use existing verification tools for this language: here we use the Gloups tool[5] to prove that $\mathbf{nor} = \mathbf{opt}$.

4.3 Proof of the property

Gloups performs several operations on the `Synchro` program and on the property to prove:

1. The program is split into hypothesis on the environment H (given by the `assert` clauses) and equations E. If P is the property we are interested in, we would like to prove some sequent stating that "under H and E, P holds". In fact, Gloups proposes to prove "under H and E, P and E hold" which is logically equivalent, but it turns out to enhance automatic decision procedures of the PVS theorem prover. Thus, the initial property P is transformed into $P \wedge E$[1].

2. Then, the new $P \wedge E$ property is unfolded n times (here $n = 4$): the property is simply rewritten n times by left-to-right oriented equations E. A sequent is formed: the consequent part is the n-times unfolded property, while the antecedent is given by H and all the k-times unfolded properties for $k < n$. Notice that while the initial property $P \wedge E$ was defined on infinite sequences (for instance rd is infinite, if R is infinite), the sequent concerns only finite sequences (and this is exactly why we use induction here).

3. As we have now a property on finite sequences, another inductive rule is applied:

$$\frac{Q(\varepsilon) \qquad Q(s) \vdash Q(s_0.s)}{\forall seq.\ Q(seq)}$$

   This rule reduces the sequent on finite sequences into a set of sequents on scalars: Gloups automatically finds the inductive case and discharges all the proof obligations into PVS.

After that, if the user interactively proves the obligations in PVS, the property $\mathbf{nor} = \mathbf{opt}$ holds. In the present case, the proofs are straightforward: it is sufficient to suggest case-splitting on the three boolean inputs (R, f_1 and nf) and the decision procedures of PVS conclude. Thus, our inductive proof really is more automatic than the original proof by Miner and Johnson. We give the proof obligations produced by Gloups in appendix.

5 Conclusion

In this article, we have proposed to prove invariant properties by induction with number-of-state-variables unfoldings. We have proved that this number of unfoldings is a minimum requirement if the inductive proof is meant to be adapted to the underlying system: strictly less unfoldings necessitate that the property be an invariant of more than one system.

In practice, this minimum requirement turns out to be sufficient in many cases, since it manages to take into account all the dynamics of the underlying system. This proof strategy is implemented in our Gloups tool, thanks to which we were able to prove the example of the fault tolerant clock synchroniser presented in the previous section.

Still, there is unfortunately a drawback to using unfolded induction: unfolding makes the sequents grow in quadratic[2] manner in the number of equations. This is why some examples (for instance the TTA membership algorithm [12]) resist to our effort: the resulting proof obligations are simply too big to fit in PVS. We are currently studying ways of optimising them.

To conclude, induction-based proofs of invariants on finite or infinite systems are both popular and needed. Our purpose was to help at automating such proofs: for that we proposed a criterion of minimum number of to be tried unfolding a priori. Our experience tends to validate the use of such a lower bound in actual proofs.

[1] Equations E do define a conjunction of boolean properties.
[2] or even exponentially, if one does not take care

References

[1] J.-R. Abrial. *The B-Book*. Cambridge University Press, 1995. 1

[2] K. Baukus, Y. Lakhnech, K. Stahl, and M. Steffen. Divide, Abstract and Model-Check. In *Proceedings of the 5th and 6th SPIN Workshops on Theoretical and Practical Aspects of Model Checking*, volume 1680 of *LNCS*. Springer–Verlag, 1999. 1

[3] S. Bensalem and Y. Lakhnech. Automatic Generation of Invariants. *Formal Methods and System Design*, 15(1):75–92, July 1999. 1

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th Principles of Programming Languages*, January 1977. 1, 3.1

[5] C. Dumas and P. Caspi. A PVS proof obligation generator for Lustre programs. In *7th International Conference on Logic for Programming and Automated Reasoning*, volume 1955 of *Lecture Notes in Artificial Intelligence*, 2000. 1, 4.2

[6] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991. 1, 4.2

[7] G. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997. 1

[8] G. Huet, G. Kahn, and Ch. Paulin-Mohring. The Coq proof assistant - a tutorial, version 6.1. rapport technique 204, INRIA, Aot 1997. Version rvise distribue avec Coq. 1

[9] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–32, 1995. 3.1

[10] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. 1

[11] P.S. Miner and S.D. Johnson. Verification of an Optimized Fault-Tolerant Clock Synchronization Circuit. In M. Sheeran and S. Singh, editors, *Designing Correct Circuits*, Electronic Workshops in Computing, Båstad, Sweden, 1996. Springer–Verlag. 4

[12] H. Pfeifer. Formal Verification of the TTP Group Membership Algorithm. In T. Bolognesi and D. Latella, editors, *Formal Methods for Distributed System Development Proceedings of FORTE XIII / PSTV XX 2000*, pages 3–18, Pisa, Italy, October 2000. Kluwer Academic Publishers. 2

[13] A. Pnueli. Deduction is forever. An invited talk at "Formal Methods", http://www.wisdom.weizmann.ac.il/ amir/fm99.ps, Toulouse, September 1999. 1

[14] M. Sheeran, S. Singh, and G. Stälmarck. Checking safety properties using induction and a sat solver. In Jr. W.A. Hunt and S.D. Johnson, editors, *International Conference on Formal Methods in Computer-Aided Design FMCAD 2000*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer Verlag, 2000. 1

[15] Sam Owre Natarajan Shankar John Rushby D.W.J Stringer-Calvert. PVS language reference. Technical report, SRI International, December 2001. 1

Appendix

We give here the three proof obligations produced by Gloups. The first of them corresponds to the proof of initialisation and is easy to prove:

$$
\begin{array}{l}
(rd_0 = 0 \ \wedge \ nor_{0,0} = div(t1_{0,0} + tn_{0,0}, 2) \ \wedge \ R_0 \ \wedge \ nf_0 \Rightarrow f1_0 \ \wedge \ \neg f1_0 \wedge \\
nor_{0,0} = opt_{0,0} \ \wedge \ nor_{0,0} = div(t1_{0,0} + tn_{0,0}, 2) \ \wedge \ t1_{0,0} = 0 \ \wedge \ tn_{0,0} = 0 \wedge \\
\neg hold_{0,0} \ \wedge \ \neg cin_{0,0} \ \wedge \ opt_{0,0} = 0 \ \wedge \ nor_{1,0} = div(t1_{1,0} + tn_{1,0}, 2) \wedge \\
t1_{1,0} = 0 \ \wedge \ tn_{1,0} = 0 \ \wedge \ \neg hold_{1,0} \ \wedge \ \neg cin_{1,0} \ \wedge \ opt_{1,0} = 0 \wedge \\
nor_{1,0} = opt_{1,0} \ \wedge \ nor_{1,0} = div(t1_{1,0} + tn_{1,0}, 2) \ \wedge \ t1_{1,0} = 0 \ \wedge \ tn_{1,0} = 0 \wedge \\
\neg hold_{1,0} \ \wedge \ \neg cin_{1,0} \ \wedge \ opt_{1,0} = 0 \ \wedge \ nor_{2,0} = div(t1_{2,0} + tn_{2,0}, 2) \wedge \\
t1_{2,0} = 0 \ \wedge \ tn_{2,0} = 0 \ \wedge \ \neg hold_{2,0} \ \wedge \ \neg cin_{2,0} \ \wedge \ opt_{2,0} = 0 \wedge \\
nor_{2,0} = opt_{2,0} \ \wedge \ nor_{2,0} = div(t1_{2,0} + tn_{2,0}, 2) \ \wedge \ t1_{2,0} = 0 \ \wedge \ tn_{2,0} = 0 \wedge \\
\neg hold_{2,0} \ \wedge \ \neg cin_{2,0} \ \wedge \ opt_{2,0} = 0 \ \wedge \ nor_{3,0} = div(t1_{3,0} + tn_{3,0}, 2) \wedge \\
t1_{3,0} = 0 \ \wedge \ tn_{3,0} = 0 \ \wedge \ \neg hold_{3,0} \ \wedge \ \neg cin_{3,0} \ \wedge \ opt_{3,0} = 0 \wedge \\
nor_{3,0} = opt_{3,0} \ \wedge \ nor_{3,0} = div(t1_{3,0} + tn_{3,0}, 2) \ \wedge \ t1_{3,0} = 0 \ \wedge \ tn_{3,0} = 0 \wedge \\
\neg hold_{3,0} \ \wedge \ \neg cin_{3,0} \ \wedge \ opt_{3,0} = 0 \ \wedge \ nor_{4,0} = div(t1_{4,0} + tn_{4,0}, 2) \wedge \\
t1_{4,0} = 0 \ \wedge \ tn_{4,0} = 0 \ \wedge \ \neg hold_{4,0} \ \wedge \ \neg cin_{4,0} \ \wedge \ opt_{4,0} = 0) \\
\Rightarrow \\
(nor_{4,0} = opt_{4,0} \ \wedge \ nor_{4,0} = div(t1_{4,0} + tn_{4,0}, 2) \ \wedge \ t1_{4,0} = 0 \wedge \\
tn_{4,0} = 0 \ \wedge \ \neg hold_{4,0} \ \wedge \ \neg cin_{4,0} \ \wedge \ opt_{4,0} = 0)
\end{array}
$$

The div operator stands for integer (Euclidian) division.

The second proof obligation deals with the state following an initialisation (*i.e.* Reset has previously been true). The colours in the following sequent should help to understand how the properties given by the equations E are proved:

$$
\begin{array}{l}
(nor_{0,0} = opt_{0,0} \ \wedge \ nor_{0,0} = div(t1_{0,0} + tn_{0,0}, 2) \ \wedge \ tn_{0,0} = 0 \ \wedge \ t1_{0,0} = 0 \ \wedge \ \neg cin_{0,0} \wedge \\
\neg hold_{0,0} \ \wedge \ opt_{0,0} = 0 \ \wedge \ nor_{1,0} = opt_{1,0} \ \wedge \ nor_{1,0} = div(t1_{1,0} + tn_{1,0}, 2) \ \wedge \ tn_{1,0} = 0 \wedge \\
t1_{1,0} = 0 \ \wedge \ \neg cin_{1,0} \ \wedge \ \neg hold_{1,0} \ \wedge \ opt_{1,0} = 0 \ \wedge \ nor_{2,0} = opt_{2,0} \wedge \\
nor_{2,0} = div(t1_{2,0} + tn_{2,0}, 2) \ \wedge \ tn_{2,0} = 0 \ \wedge \ t1_{2,0} = 0 \ \wedge \ \neg cin_{2,0} \ \wedge \ \neg hold_{2,0} \ \wedge \ opt_{2,0} = 0 \wedge \\
nor_{3,0} = opt_{3,0} \ \wedge \ nor_{3,0} = div(t1_{3,0} + tn_{3,0}, 2) \ \wedge \ tn_{3,0} = 0 \ \wedge \ t1_{3,0} = 0 \ \wedge \ \neg cin_{3,0} \wedge \\
\neg hold_{3,0} \ \wedge \ opt_{3,0} = 0 \ \wedge \ nor_{4,0} = div(t1_{4,0} + tn_{4,0}, 2) \ \wedge \ \neg cin_{4,0} \ \wedge \ \neg hold_{4,0} \ \wedge \ tn_{4,0} = 0 \wedge \\
t1_{4,0} = 0 \ \wedge \ opt_{4,0} = 0 \ \wedge \ \neg f1_0 \ \wedge \ nf_0 \Rightarrow f1_0 \ \wedge \ R_0 \ \wedge \ rd_0 = 0 \wedge \\
rd_1 = if \ R_1 \ then \ 0 \ else \ rd_0 + 1 \ \wedge \ nor_{0,1} = div(t1_{0,1} + tn_{0,1}, 2) \ \wedge \ nf_1 \Rightarrow f1_1 \wedge \\
(R_1 \vee nf_0 \Rightarrow nf_1) \ \wedge \ nor_{0,1} = opt_{0,1} \ \wedge \ nor_{0,1} = div(t1_{0,1} + tn_{0,1}, 2) \wedge \\
t1_{0,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{0,0} \ else \ rd_1) \ \wedge \ cin_{0,1} = (hold_{0,0} \wedge \neg nf_1) \wedge \\
tn_{0,1} = if \ R_1 then \ 0 \ else \ (if \ nf_1 \ then \ tn_{0,0} \ else \ rd_1) \ \wedge \ hold_{0,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{0,0}) \wedge \\
opt_{0,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{0,1} \ then \ 1 \ else \ 0) + opt_{0,0} \ else \ rd_1) \wedge \\
nor_{1,1} = div(t1_{1,1} + tn_{1,1}, 2) \ \wedge \ t1_{1,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{0,0} \ else \ rd_1) \wedge \\
cin_{1,1} = (hold_{0,0} \wedge \neg nf_1) \ \wedge \ tn_{1,1} = if \ R_1 then \ 0 \ else \ (if \ nf_1 \ then \ tn_{0,0} \ else \ rd_1) \wedge \\
hold_{1,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{0,0}) \wedge \\
opt_{1,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{0,1} \ then \ 1 \ else \ 0) + opt_{0,0} \ else \ rd_1) \wedge \\
nor_{1,1} = opt_{1,1} \ \wedge \ nor_{1,1} = div(t1_{1,1} + tn_{1,1}, 2) \wedge \\
t1_{1,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{1,0} \ else \ rd_1) \ \wedge \ cin_{1,1} = (hold_{1,0} \wedge \neg nf_1) \wedge \\
tn_{1,1} = if \ R_1 then \ 0 \ else \ (if \ nf_1 \ then \ tn_{1,0} \ else \ rd_1) \ \wedge \ hold_{1,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{1,0}) \wedge \\
opt_{1,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{1,1} \ then \ 1 \ else \ 0) + opt_{1,0} \ else \ rd_1) \wedge \\
nor_{2,1} = div(t1_{2,1} + tn_{2,1}, 2) \ \wedge \ t1_{2,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{1,0} \ else \ rd_1) \wedge \\
tn_{2,1} = if \ R_1 then \ 0 \ else \ (if \ nf_1 \ then \ tn_{1,0} \ else \ rd_1) \ \wedge \ hold_{2,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{1,0}) \wedge \\
cin_{2,1} = (hold_{1,0} \wedge \neg nf_1) \wedge \\
opt_{2,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{1,1} \ then \ 1 \ else \ 0) + opt_{1,0} \ else \ rd_1) \wedge \\
nor_{2,1} = opt_{2,1} \ \wedge \ nor_{2,1} = div(t1_{2,1} + tn_{2,1}, 2) \wedge \\
t1_{2,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{2,0} \ else \ rd_1) \ \wedge \ cin_{2,1} = (hold_{2,0} \wedge \neg nf_1) \wedge \\
tn_{2,1} = if \ R_1 then \ 0 \ else \ (if \ nf_1 \ then \ tn_{2,0} \ else \ rd_1) \ \wedge \ hold_{2,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{2,0}) \wedge \\
opt_{2,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{2,1} \ then \ 1 \ else \ 0) + opt_{2,0} \ else \ rd_1) \wedge \\
nor_{3,1} = div(t1_{3,1} + tn_{3,1}, 2) \ \wedge \ t1_{3,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{2,0} \ else \ rd_1) \wedge \\
tn_{3,1} = if \ R_1 then \ 0 \ else \ (if \ nf_1 \ then \ tn_{2,0} \ else \ rd_1) \ \wedge \ hold_{3,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{2,0}) \wedge \\
cin_{3,1} = (hold_{2,0} \wedge \neg nf_1) \wedge \\
opt_{3,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{2,1} \ then \ 1 \ else \ 0) + opt_{2,0} \ else \ rd_1) \wedge \\
nor_{3,1} = opt_{3,1} \ \wedge \ nor_{3,1} = div(t1_{3,1} + tn_{3,1}, 2) \wedge \\
t1_{3,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{3,0} \ else \ rd_1) \ \wedge \ cin_{3,1} = (hold_{3,0} \wedge \neg nf_1) \wedge \\
tn_{3,1} = if \ R_1 then \ 0 \ else \ (if \ nf_1 \ then \ tn_{3,0} \ else \ rd_1) \ \wedge \ hold_{3,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{3,0}) \wedge \\
opt_{3,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{3,1} \ then \ 1 \ else \ 0) + opt_{3,0} \ else \ rd_1) \wedge \\
nor_{4,1} = div(t1_{4,1} + tn_{4,1}, 2) \ \wedge \ t1_{4,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{3,0} \ else \ rd_1) \wedge \\
tn_{4,1} = if \ R_1 then \ 0 \ else \ (if \ nf_1 \ then \ tn_{3,0} \ else \ rd_1) \ \wedge \ hold_{4,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{3,0}) \wedge \\
cin_{4,1} = (hold_{3,0} \wedge \neg nf_1) \wedge \\
opt_{4,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{3,1} \ then \ 1 \ else \ 0) + opt_{3,0} \ else \ rd_1)) \\
\Rightarrow \\
(nor_{4,1} = opt_{4,1} \ \wedge \ nor_{4,1} = div(t1_{4,1} + tn_{4,1}, 2) \wedge \\
t1_{4,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ t1_{4,0} \ else \ rd_1) \wedge \\
tn_{4,1} = if \ R_1 \ then \ 0 \ else \ (if \ nf_1 \ then \ tn_{4,0} \ else \ rd_1) \ \wedge \ hold_{4,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{4,0}) \wedge \\
cin_{4,1} = (hold_{4,0} \wedge \neg nf_1) \wedge \\
opt_{4,1} = if \ R_1 \ then \ 0 \ else \ (if \ f1_1 \ then \ (if \ cin_{4,1} \ then \ 1 \ else \ 0) + opt_{4,0} \ else \ rd_1))
\end{array}
$$

After proving the coloured properties, we can simplify the sequent anf focus on proving the remaining property P.

$$
\begin{array}{l}
\ldots \wedge \\
nor_{3,1} = opt_{3,1} \ \wedge \ nor_{3,1} = div(t1_{3,1} + tn_{3,1}, 2) \wedge \\
t1_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{3,0}\ else\ rd_1) \ \wedge \ cin_{3,1} = (hold_{3,0} \wedge \neg nf_1) \wedge \\
tn_{3,1} = if\ R_1 then\ 0\ else\ (if\ nf_1\ then\ tn_{3,0}\ else\ rd_1) \ \wedge \ hold_{3,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{3,0}) \wedge \\
opt_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{3,1}\ then\ 1\ else\ 0) + opt_{3,0}\ else\ rd_1) \wedge \\
nor_{4,1} = div(t1_{4,1} + tn_{4,1}, 2) \ \wedge \ t1_{4,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{3,0}\ else\ rd_1) \wedge \\
tn_{4,1} = if\ R_1 then\ 0\ else\ (if\ nf_1\ then\ tn_{3,0}\ else\ rd_1) \ \wedge \ hold_{4,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{3,0}) \wedge \\
cin_{4,1} = (hold_{3,0} \wedge \neg nf_1) \wedge \\
opt_{4,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{3,1}\ then\ 1\ else\ 0) + opt_{3,0}\ else\ rd_1)) \\
\Rightarrow \\
(nor_{4,1} = opt_{4,1})
\end{array}
$$

Same colour means equality...

Finally, we give the last (and lengthy) proof obligation (we colour the parts of the sequent which are involved in the proof of the equations E):

$$
\begin{array}{l}
((R_1 \vee (nf_0 \Rightarrow nf_1)) \ \wedge \ (nf_1 \Rightarrow f1_1) \ \wedge \ rd_1 = if\ R_1\ then\ 0\ else\ rd_0 + 1 \wedge \\
nor_{0,1} = div(t1_{0,1} + tn_{0,1}, 2) \ \wedge \ t1_{0,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{0,0}\ else\ rd_1) \wedge \\
nor_{0,1} = opt_{0,1} \ \wedge \ tn_{0,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{0,0}\ else\ rd_1) \wedge \\
opt_{0,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{0,1}\ then\ 1\ else\ 0) + opt_{0,0}\ else\ rd_1) \wedge \\
cin_{0,1} = (hold_{0,0} \wedge \neg nf_1) \ \wedge \ hold_{0,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{0,0}) \wedge \\
nor_{1,1} = div(t1_{1,1} + tn_{1,1}, 2) \ \wedge \ t1_{1,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{0,0}\ else\ rd_1) \wedge \\
nor_{1,1} = opt_{1,1} \ \wedge \ t1_{1,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{1,0}\ else\ rd_1) \wedge \\
tn_{1,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{0,0}\ else\ rd_1) \ \wedge \ cin_{1,1} = (hold_{0,0} \wedge \neg nf_1) \wedge \\
tn_{1,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{1,0}\ else\ rd_1) \ \wedge \ cin_{1,1} = (hold_{1,0} \wedge \neg nf_1) \wedge \\
opt_{1,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{0,1}\ then\ 1\ else\ 0) + opt_{0,0}\ else\ rd_1) \\
opt_{1,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{1,1}\ then\ 1\ else\ 0) + opt_{1,0}\ else\ rd_1) \\
hold_{1,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{0,0}) \ \wedge \ hold_{1,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{1,0}) \wedge \\
nor_{2,1} = div(t1_{2,1} + tn_{2,1}, 2) \ \wedge \ t1_{2,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{1,0}\ else\ rd_1) \wedge \\
nor_{2,1} = opt_{2,1} \ \wedge \ t1_{2,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{2,0}\ else\ rd_1) \wedge \\
tn_{2,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{1,0}\ else\ rd_1) \ \wedge \ cin_{2,1} = (hold_{1,0} \wedge \neg nf_1) \wedge \\
tn_{2,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{2,0}\ else\ rd_1) \ \wedge \ cin_{2,1} = (hold_{2,0} \wedge \neg nf_1) \wedge \\
opt_{2,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{1,1}\ then\ 1\ else\ 0) + opt_{1,0}\ else\ rd_1) \\
opt_{2,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{2,1}\ then\ 1\ else\ 0) + opt_{2,0}\ else\ rd_1) \\
hold_{2,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{1,0}) \ \wedge \ hold_{2,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{2,0}) \wedge \\
nor_{3,1} = div(t1_{3,1} + tn_{3,1}, 2) \ \wedge \ t1_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{2,0}\ else\ rd_1) \wedge \\
nor_{3,1} = opt_{3,1} \ \wedge \ t1_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{3,0}\ else\ rd_1) \wedge \\
tn_{3,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{2,0}\ else\ rd_1) \ \wedge \ cin_{3,1} = (hold_{2,0} \wedge \neg nf_1) \wedge \\
tn_{3,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{3,0}\ else\ rd_1) \ \wedge \ cin_{3,1} = (hold_{3,0} \wedge \neg nf_1) \wedge \\
opt_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{2,1}\ then\ 1\ else\ 0) + opt_{2,0}\ else\ rd_1) \wedge \\
opt_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{3,1}\ then\ 1\ else\ 0) + opt_{3,0}\ else\ rd_1) \wedge \\
hold_{3,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{2,0}) \ \wedge \ hold_{3,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{3,0}) \wedge \\
nor_{4,1} = div(t1_{4,1} + tn_{4,1}, 2) \ \wedge \ t1_{4,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{3,0}\ else\ rd_1) \wedge \\
tn_{4,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{3,0}\ else\ rd_1) \wedge \\
opt_{4,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{3,1}\ then\ 1\ else\ 0) + opt_{3,0}\ else\ rd_1) \wedge \\
cin_{4,1} = (hold_{3,0} \wedge \neg nf_1) \ \wedge \ hold_{4,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{3,0}) \wedge
\end{array}
$$

$$
\begin{array}{l}
(R_2 \vee (nf_1 => nf_2)) \ \wedge \ (nf_2 \Rightarrow f1_2) \ \wedge \ rd_2 = if\ R_2\ then\ 0\ else\ rd_1 + 1 \ \wedge \\
nor_{0,2} = div(t1_{0,2} + tn_{0,2}, 2) \ \wedge \ t1_{0,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{0,1}\ else\ rd_2) \ \wedge \\
nor_{0,2} = opt_{0,2} \ \wedge \ tn_{0,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{0,1}\ else\ rd_2) \ \wedge \\
opt_{0,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{0,2}\ then\ 1\ else\ 0) + opt_{0,1}\ else\ rd_2) \ \wedge \\
cin_{0,2} = (hold_{0,1} \wedge \neg nf_2) \ \wedge \ hold_{0,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{0,1}) \ \wedge \\
nor_{1,2} = div(t1_{1,2} + tn_{1,2}, 2) \ \wedge \ t1_{1,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{0,1}\ else\ rd_2) \ \wedge \\
nor_{1,2} = opt_{1,2} \ \wedge \ t1_{1,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{1,1}\ else\ rd_2) \ \wedge \\
tn_{1,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{0,1}\ else\ rd_2) \ \wedge \ cin_{1,2} = (hold_{0,1} \wedge \neg nf_2) \ \wedge \\
tn_{1,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{1,1}\ else\ rd_2) \ \wedge \ cin_{1,2} = (hold_{1,1} \wedge \neg nf_2) \ \wedge \\
opt_{1,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{0,2}\ then\ 1\ else\ 0) + opt_{0,1}\ else\ rd_2) \\
opt_{1,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{1,2}\ then\ 1\ else\ 0) + opt_{1,1}\ else\ rd_2) \\
hold_{1,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{0,1}) \ \wedge \ hold_{1,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{1,1}) \ \wedge \\
nor_{2,2} = div(t1_{2,2} + tn_{2,2}, 2) \ \wedge \ t1_{2,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{1,1}\ else\ rd_2) \ \wedge \\
nor_{2,2} = opt_{2,2} \ \wedge \ t1_{2,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{2,1}\ else\ rd_2) \ \wedge \\
tn_{2,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{1,1}\ else\ rd_2) \ \wedge \ cin_{2,2} = (hold_{1,1} \wedge \neg nf_2) \ \wedge \\
tn_{2,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{2,1}\ else\ rd_2) \ \wedge \ cin_{2,2} = (hold_{2,1} \wedge \neg nf_2) \ \wedge \\
opt_{2,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{1,2}\ then\ 1\ else\ 0) + opt_{1,1}\ else\ rd_2) \\
opt_{2,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{2,2}\ then\ 1\ else\ 0) + opt_{2,1}\ else\ rd_2) \\
hold_{2,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{1,1}) \ \wedge \ hold_{2,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{2,1}) \ \wedge \\
nor_{3,2} = div(t1_{3,2} + tn_{3,2}, 2) \ \wedge \ t1_{3,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{2,1}\ else\ rd_2) \ \wedge \\
nor_{3,2} = opt_{3,2} \ \wedge \ t1_{3,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{3,1}\ else\ rd_2) \ \wedge \\
tn_{3,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{2,1}\ else\ rd_2) \ \wedge \ cin_{3,2} = (hold_{2,1} \wedge \neg nf_2) \ \wedge \\
tn_{3,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{3,1}\ else\ rd_2) \ \wedge \ cin_{3,2} = (hold_{3,1} \wedge \neg nf_2) \ \wedge \\
opt_{3,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{2,2}\ then\ 1\ else\ 0) + opt_{2,1}\ else\ rd_2) \\
opt_{3,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{3,2}\ then\ 1\ else\ 0) + opt_{3,1}\ else\ rd_2) \\
hold_{3,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{2,1}) \ \wedge \ hold_{3,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{3,1}) \ \wedge \\
nor_{4,2} = div(t1_{4,2} + tn_{4,2}, 2) \ \wedge \ t1_{4,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{3,1}\ else\ rd_2) \ \wedge \\
tn_{4,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{3,1}\ else\ rd_2) \ \wedge \\
opt_{4,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{3,2}\ then\ 1\ else\ 0) + opt_{3,1}\ else\ rd_2) \ \wedge \\
cin_{4,2} = (hold_{3,1} \wedge \neg nf_2) \ \wedge \ hold_{4,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{3,1})) \\
\Rightarrow \\
(nor_{4,2} = opt_{4,2} \ \wedge \ nor_{4,2} = div(t1_{4,2} + tn_{4,2}, 2) \ \wedge \\
t1_{4,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ t1_{4,1}\ else\ rd_2) \ \wedge \\
tn_{4,2} = if\ R_2\ then\ 0\ else\ (if\ nf_2\ then\ tn_{4,1}\ else\ rd_2) \ \wedge \\
hold_{4,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{4,1}) \ \wedge \ cin_{4,2} = (hold_{4,1} \wedge \neg nf_2) \ \wedge \\
opt_{4,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{4,2}\ then\ 1\ else\ 0) + opt_{4,1}\ else\ rd_2))
\end{array}
$$

Now we can focus on proving the remaining property $nor_{4,2} = opt_{4,2}$. The case $R_2 = true$ yields:

$$
\begin{array}{l}
(\ldots \ \wedge \\
opt_{3,2} = if\ R_2\ then\ 0\ else\ (if\ f1_2\ then\ (if\ cin_{3,2}\ then\ 1\ else\ 0) + opt_{3,1}\ else\ rd_2) \\
hold_{3,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{2,1}) \ \wedge \ hold_{3,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{3,1}) \ \wedge \\
nor_{4,2} = div(t1_{4,2} + tn_{4,2}, 2) \ \wedge \ t1_{4,2} = 0 \ \wedge \\
tn_{4,2} = 0 \ \wedge \\
opt_{4,2} = 0 \ \wedge \\
cin_{4,2} = (hold_{3,1} \wedge \neg nf_2) \ \wedge \ hold_{4,2} = (\neg R_2 \wedge f1_2 \wedge \neg hold_{3,1})) \\
\Rightarrow \\
(nor_{4,2} = opt_{4,2})
\end{array}
$$

this simplifies to the proof of

$$div(0 + 0, 2) = 0$$

which is true, since $div(0 + 0, 2) = \lfloor \frac{0+0}{2} \rfloor = 0$.

Finally, the case $R_2 = false$ gives the following sequent:

$$
\begin{array}{l}
((R_1 \vee (nf_0 \Rightarrow nf_1)) \ \wedge \ (nf_1 \Rightarrow f1_1) \ \wedge \ rd_1 = if\ R_1\ then\ 0\ else\ rd_0 + 1 \ \wedge \\
nor_{0,1} = div(t1_{0,1} + tn_{0,1}, 2) \ \wedge \ t1_{0,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{0,0}\ else\ rd_1) \ \wedge \\
nor_{0,1} = opt_{0,1} \ \wedge \ tn_{0,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{0,0}\ else\ rd_1) \ \wedge \\
opt_{0,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{0,1}\ then\ 1\ else\ 0) + opt_{0,0}\ else\ rd_1) \ \wedge \\
cin_{0,1} = (hold_{0,0} \wedge \neg nf_1) \ \wedge \ hold_{0,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{0,0}) \ \wedge \\
nor_{1,1} = div(t1_{1,1} + tn_{1,1}, 2) \ \wedge \ t1_{1,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{0,0}\ else\ rd_1) \ \wedge \\
nor_{1,1} = opt_{1,1} \ \wedge \ t1_{1,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{1,0}\ else\ rd_1) \ \wedge \\
tn_{1,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{0,0}\ else\ rd_1) \ \wedge \ cin_{1,1} = (hold_{0,0} \wedge \neg nf_1) \ \wedge \\
tn_{1,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{1,0}\ else\ rd_1) \ \wedge \ cin_{1,1} = (hold_{1,0} \wedge \neg nf_1) \ \wedge \\
opt_{1,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{0,1}\ then\ 1\ else\ 0) + opt_{0,0}\ else\ rd_1) \\
opt_{1,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{1,1}\ then\ 1\ else\ 0) + opt_{1,0}\ else\ rd_1) \\
hold_{1,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{0,0}) \ \wedge \ hold_{1,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{1,0}) \ \wedge \\
nor_{2,1} = div(t1_{2,1} + tn_{2,1}, 2) \ \wedge \ t1_{2,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{1,0}\ else\ rd_1) \ \wedge \\
nor_{2,1} = opt_{2,1} \ \wedge \ t1_{2,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{2,0}\ else\ rd_1) \ \wedge \\
tn_{2,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{1,0}\ else\ rd_1) \ \wedge \ cin_{2,1} = (hold_{1,0} \wedge \neg nf_1) \ \wedge \\
tn_{2,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{2,0}\ else\ rd_1) \ \wedge \ cin_{2,1} = (hold_{2,0} \wedge \neg nf_1) \ \wedge \\
opt_{2,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{1,1}\ then\ 1\ else\ 0) + opt_{1,0}\ else\ rd_1) \ \wedge \\
opt_{2,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{2,1}\ then\ 1\ else\ 0) + opt_{2,0}\ else\ rd_1) \ \wedge \\
hold_{2,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{1,0}) \ \wedge \ hold_{2,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{2,0}) \ \wedge \\
nor_{3,1} = div(t1_{3,1} + tn_{3,1}, 2) \ \wedge \ t1_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{2,0}\ else\ rd_1) \ \wedge \\
nor_{3,1} = opt_{3,1} \ \wedge \ t1_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{3,0}\ else\ rd_1) \ \wedge \\
tn_{3,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{2,0}\ else\ rd_1) \ \wedge \ cin_{3,1} = (hold_{2,0} \wedge \neg nf_1) \ \wedge \\
tn_{3,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{3,0}\ else\ rd_1) \ \wedge \ cin_{3,1} = (hold_{3,0} \wedge \neg nf_1) \ \wedge \\
opt_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{2,1}\ then\ 1\ else\ 0) + opt_{2,0}\ else\ rd_1) \ \wedge \\
opt_{3,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{3,1}\ then\ 1\ else\ 0) + opt_{3,0}\ else\ rd_1) \ \wedge \\
hold_{3,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{2,0}) \ \wedge \ hold_{3,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{3,0}) \ \wedge \\
nor_{4,1} = div(t1_{4,1} + tn_{4,1}, 2) \ \wedge \ t1_{4,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ t1_{3,0}\ else\ rd_1) \ \wedge \\
tn_{4,1} = if\ R_1\ then\ 0\ else\ (if\ nf_1\ then\ tn_{3,0}\ else\ rd_1) \ \wedge \ hold_{4,1} = (\neg R_1 \wedge f1_1 \wedge \neg hold_{3,0}) \ \wedge \\
opt_{4,1} = if\ R_1\ then\ 0\ else\ (if\ f1_1\ then\ (if\ cin_{3,1}\ then\ 1\ else\ 0) + opt_{3,0}\ else\ rd_1) \ \wedge \\
cin_{4,1} = (hold_{3,0} \wedge \neg nf_1) \ \wedge \ (nf_1 \Rightarrow nf_2) \ \wedge \ (nf_2 \Rightarrow f1_2) \ \wedge \ rd_2 = rd_1 + 1 \ \wedge \\
nor_{0,2} = div(t1_{0,2} + tn_{0,2}, 2) \ \wedge \ t1_{0,2} = if\ f1_2\ then\ t1_{0,1}\ else\ rd_2 \ \wedge \ cin_{0,2} = (hold_{0,1} \wedge \neg nf_2) \wedge \\
nor_{0,2} = opt_{0,2} \ \wedge \ tn_{0,2} = if\ nf_2\ then\ tn_{0,1}\ else\ rd_2 \ \wedge \ hold_{0,2} = (f1_2 \wedge \neg hold_{0,1}) \ \wedge \\
opt_{0,2} = if\ f1_2\ then\ (if\ cin_{0,2}\ then\ 1\ else\ 0) + opt_{0,1}\ else\ rd_2 \ \wedge \\
nor_{1,2} = div(t1_{1,2} + tn_{1,2}, 2) \ \wedge \ t1_{1,2} = if\ f1_2\ then\ t1_{0,1}\ else\ rd_2 \ \wedge \ hold_{1,2} = (f1_2 \wedge \neg hold_{0,1}) \wedge \\
nor_{1,2} = opt_{1,2} \ \wedge \ t1_{1,2} = if\ f1_2\ then\ t1_{1,1}\ else\ rd_2 \ \wedge \ hold_{1,2} = (f1_2 \wedge \neg hold_{1,1}) \ \wedge \\
tn_{1,2} = if\ nf_2\ then\ tn_{0,1}\ else\ rd_2 \ \wedge \ tn_{1,2} = if\ nf_2\ then\ tn_{1,1}\ else\ rd_2 \ \wedge \\
opt_{1,2} = if\ f1_2\ then\ (if\ cin_{0,2}\ then\ 1\ else\ 0) + opt_{0,1}\ else\ rd_2 \ \wedge \ cin_{1,2} = (hold_{0,1} \wedge \neg nf_2) \ \wedge \\
opt_{1,2} = if\ f1_2\ then\ (if\ cin_{1,2}\ then\ 1\ else\ 0) + opt_{1,1}\ else\ rd_2 \ \wedge \ cin_{1,2} = (hold_{1,1} \wedge \neg nf_2) \ \wedge \\
nor_{2,2} = div(t1_{2,2} + tn_{2,2}, 2) \ \wedge \ t1_{2,2} = if\ f1_2\ then\ t1_{1,1}\ else\ rd_2 \ \wedge \ hold_{2,2} = (f1_2 \wedge \neg hold_{1,1}) \wedge \\
nor_{2,2} = opt_{2,2} \ \wedge \ t1_{2,2} = if\ f1_2\ then\ t1_{2,1}\ else\ rd_2 \ \wedge \ hold_{2,2} = (f1_2 \wedge \neg hold_{2,1}) \ \wedge \\
tn_{2,2} = if\ nf_2\ then\ tn_{1,1}\ else\ rd_2 \ \wedge \ tn_{2,2} = if\ nf_2\ then\ tn_{2,1}\ else\ rd_2 \ \wedge \\
opt_{2,2} = if\ f1_2\ then\ (if\ cin_{1,2}\ then\ 1\ else\ 0) + opt_{1,1}\ else\ rd_2 \ \wedge \ cin_{2,2} = (hold_{1,1} \wedge \neg nf_2) \ \wedge \\
opt_{2,2} = if\ f1_2\ then\ (if\ cin_{2,2}\ then\ 1\ else\ 0) + opt_{2,1}\ else\ rd_2 \ \wedge \ cin_{2,2} = (hold_{2,1} \wedge \neg nf_2) \ \wedge \\
nor_{3,2} = div(t1_{3,2} + tn_{3,2}, 2) \ \wedge \ t1_{3,2} = if\ f1_2\ then\ t1_{2,1}\ else\ rd_2 \ \wedge \ hold_{3,2} = (f1_2 \wedge \neg hold_{2,1}) \wedge \\
nor_{3,2} = opt_{3,2} \ \wedge \ t1_{3,2} = if\ f1_2\ then\ t1_{3,1}\ else\ rd_2 \ \wedge \ hold_{3,2} = (f1_2 \wedge \neg hold_{3,1}) \ \wedge \\
tn_{3,2} = if\ nf_2\ then\ tn_{2,1}\ else\ rd_2 \ \wedge \ tn_{3,2} = if\ nf_2\ then\ tn_{3,1}\ else\ rd_2 \ \wedge \\
opt_{3,2} = if\ f1_2\ then\ (if\ cin_{2,2}\ then\ 1\ else\ 0) + opt_{2,1}\ else\ rd_2 \ \wedge \ cin_{3,2} = (hold_{2,1} \wedge \neg nf_2) \ \wedge \\
opt_{3,2} = if\ f1_2\ then\ (if\ cin_{3,2}\ then\ 1\ else\ 0) + opt_{3,1}\ else\ rd_2 \ \wedge \ cin_{3,2} = (hold_{3,1} \wedge \neg nf_2) \ \wedge \\
nor_{4,2} = div(t1_{4,2} + tn_{4,2}, 2) \ \wedge \ t1_{4,2} = if\ f1_2\ then\ t1_{3,1}\ else\ rd_2 \ \wedge \\
tn_{4,2} = if\ nf_2\ then\ tn_{3,1}\ else\ rd_2 \ \wedge \ hold_{4,2} = (f1_2 \wedge \neg hold_{3,1}) \ \wedge \ cin_{4,2} = (hold_{3,1} \wedge \neg nf_2) \ \wedge \\
opt_{4,2} = if\ f1_2\ then\ (if\ cin_{3,2}\ then\ 1\ else\ 0) + opt_{3,1}\ else\ rd_2)) \\
\Rightarrow \\
(nor_{4,2} = opt_{4,2})
\end{array}
$$