Master Thesis

Relevance filters for event-B

Author[s]:
Röder, Jann

Publication Date:
2010

Permanent Link:
https://doi.org/10.3929/ethz-a-006266803

Rights / License:
In Copyright - Non-Commercial Use Permitted
Relevance Filters for Event-B

Jann Röder

April 5, 2010
Abstract

Unnecessary hypotheses, that are not required to find a proof of the goal, often prevent an automated theorem prover (ATP) from finding a proof within reasonable time. In this thesis we examine and compare a number of syntactic relevance filtering techniques from the literature as well as our own relevance filtering idea based on structural similarities between formulas.

The evaluation of the filtering techniques in the context of Event-B shows that relevance filtering provides a significant advantage in both proving times and success rate over not using filtering. Additionally we show that using several different relevance filtering techniques with short prover timeouts improves the success rate significantly more than increasing the prover timeout for a single filtering strategy.

We present a filtering strategy for the Rodin platform that reduces the number of proofs that need help by the user by about 40% compared to pre-existing strategies. Finally we will show that our filtering strategy achieves a success rate that lies within one percentage point of what is possible using a perfect relevance filter that selects exactly the required hypotheses.
# Contents

1 Introduction .................................................. 5  
1.1 Related Work .................................................. 6  

2 Preliminaries .................................................. 8  
2.1 Definitions .................................................... 8  
  2.1.1 Logical Connectives ........................................ 8  
  2.1.2 Types ...................................................... 8  
  2.1.3 Terms and Formulas ........................................ 8  
  2.1.4 Operators .................................................. 10  
  2.1.5 Variables .................................................. 10  
  2.1.6 Symbols .................................................... 11  
  2.1.7 Predicates .................................................. 11  
  2.1.8 Sequent ..................................................... 11  
  2.2 The Rodin Platform ........................................... 12  
  2.2.1 The Provers ............................................... 12  
  2.2.2 Tactics .................................................... 14  
  2.2.3 Proving Interface ........................................... 14  

3 Filtering Techniques ........................................... 16  
3.1 Lasso ......................................................... 17  
3.2 Whitelisting ................................................... 17  
  3.2.1 Details ..................................................... 17  
3.3 Algorithm by Meng and Paulson ................................ 18  
3.4 SInE Filter ..................................................... 20  
  3.4.1 Details ..................................................... 20  
3.5 Direct (Contextual) Relevance Filter .......................... 20  
  3.5.1 Implementation ............................................. 22  
3.6 A Filter based on Subexpression Matching ..................... 22  
  3.6.1 The Algorithm ............................................. 23  
  3.6.2 Similarity Measure ......................................... 24  
  3.6.3 How to Explain Symbols .................................... 28  
  3.6.4 Summary ................................................... 32  
3.7 Multi Filter ................................................... 32
1 Introduction

When using automated theorem provers (ATP) on problems produced by a formal method such as Event-B it is very often the case that the problem contains hypotheses which are not necessary to prove the goal. Unnecessary hypotheses are a problem especially for resolution based provers because they lack efficient techniques to detect those unnecessary hypotheses.

When a human proves a theorem, his biggest problem is most likely not which hypotheses are necessary to infer a proof, but how and in which order to use them. For a resolution based prover there is only one rule: the resolution rule. The order in which the rule is applied to the relevant hypotheses is not that crucial. What is crucial, however, is to only apply it to relevant hypotheses. A resolution prover can apply the resolution rule millions of times without getting closer to proving the goal.

Despite steady progress in the development of automated theorem provers as evident in the annual CASC competitions [25], irrelevant hypotheses are still a problem. In the last CASC competition [26] the new LTB division has been added which focuses on problems with many irrelevant hypotheses. The most successful candidates in the LTB division used the SInE [12] relevance filter in conjunction with various theorem provers.

A relevance filter tries to determine the relevance of a hypothesis for the proof of a goal before the prover is invoked. The expectation is that by invoking the prover only with hypotheses considered relevant by the relevance filter, the negative effects of too many irrelevant hypotheses are reduced.

In this thesis we examine several relevance filtering methods that are available in the literature as well as our own method. We evaluated the methods in the context of Event-B using the Rodin platform. During the evaluation we fine tuned the relevance filtering strategies for the Event-B context and developed an Event-B specific whitelisting method that we use in conjunction with all relevance filters. We also developed our own idea for relevance filtering which is based on structural similarity between formulas. To demonstrate the robustness of our results and dispel arguments about over-fitting filters to a specific test set, we confirmed our results with a second test set that we did not look at when we implemented the filters.

Our main contribution is the discovery that a combination of different relevance filtering techniques with short prover timeouts increases the success rate much
more than a single relevance filtering method with a longer timeout. Additionally we compare a number of published relevance filtering techniques which, to the best of our knowledge, has not been done before, except maybe in [24] where the authors compare two of their own ideas. More Event-B specific contributions include a new proving strategy that reduces the number of manual proofs in our test set by 20 – 40%, depending on how much time the strategy is allowed to use.

**Organization**  The remainder of this thesis is organized in five chapters. In Chapter 2 we provide definitions and give an overview of the Rodin platform. In Chapter 3 we present a number of filtering techniques from the literature as well as our own method. Chapter 4 contains the evaluation and comparison of the filters presented in Chapter 3. Chapter 5 gives a brief overview of the tools we have developed for this thesis. Finally the last chapter concludes the thesis with a summary of the main results.

1.1 Related Work

What distinguishes our work from other work is that we compare a number of relevance filters and combine them. Another difference to most other work is that we evaluate our methods using the Rodin platform which is actually used in practice in contrast to the TPTP library[29], that is used for the evaluation in many papers, whose only purpose is to provide an academic benchmark for theorem provers.

In [16] the evaluation is done for problems generated by the Sledgehammer link-up for the Isabelle [20]. Their method, which is based on symbol overlap between formulas, also turned out to work well in the Event-B context. Another paper that deals with relevance for problems generated by a formal method is [23]. However, their approach assumes a hierarchical theory structure which we do not have in Event-B.

The approach of using multiple proof attempts with different sets of hypotheses to find “difficult” proofs has already been tried in [27]. However, they only order the hypotheses based on their “heaviness”, which is a combination of the number of symbols and the degree of function nesting. Then hypotheses are systematically removed to generate all possible subsets of hypotheses until reaching a timeout. They state in their conclusion that their next step would be to use a relevance ordering to guide the systematic subset generation. We went one step further by generating several hypothesis subsets using different heuristic relevance orderings instead of only one. The usage of multiple filters is more robust because it could happen that a single relevance filter ranks a crucial hypothesis very low so that even a very generous strictness parameter would not select it.
Besides the papers mentioned above there are a number of papers that deal with specific relevance filtering methods in a more academic fashion. In fact the relevance filtering problem is quite old and can be traced back at least until 1980 when Plaisted presented a method that selects clauses using a clause connection graph in which clauses are connected if they contain symbols that can be resolved [21]. Plaisted later re-used his ideas from the 1980 paper in a more recent one [22]. Even though the recent paper is much longer and more elaborate, the basic idea is the same. The only experimental results we could find were [14] from 1984 which can not be compared to modern problems for automated theorem provers. Meng and Paulson [16] claim that they tried Plaisted’s method with their Sledgehammer link-up for Isabelle, however they write it does not work well because it selects too many clauses.

In [24] another syntactical method is presented and compared to a method based on Latent Semantic Analysis (LSA) which is a technique for analyzing relationships between (text) documents. A different approach was taken in [28] where the authors present a method that selects hypotheses which are false in a model that is constructed for the already selected hypotheses and the negated goal. It can however be debated whether their method still is a relevance filter since it can establish theoremhood by itself without the help of a theorem prover.

The authors of [15] use predicate abstraction to filter out irrelevant hypotheses for the concrete proof attempt. Others (for example [9]) have also used abstraction; However, they use abstraction in an iterative process where abstract and concrete proof attempts are interleaved in contrast to the former paper which uses abstraction for a one shot selection.
2 Preliminaries

2.1 Definitions

In this section we give some Definitions that will be used in this thesis. A detailed description of Event-B’s mathematical language can be found in [17]. They, however, use a terminology that is closer to compiler design, whereas our definitions are closer to what is used in books about logic (e.g. [7]).

2.1.1 Logical Connectives

Event-B has the following logical connectives:

| ∧  | Conjuction  | ∨ | Disjunction |
| └───|-------------| └───|-------------|
| ⇒  | Implication | ⇔ | Equivalence |
| ¬  | Negation    | ⊥ | Falsity     |
| T  | Truth       |

2.1.2 Types

Event-B uses a typed logic where every term is associated with a type. The type of a term can be one of the following:

- a basic set, that is a predefined set such as \( \mathbb{Z} \) or \( \text{BOOL} \), or a carrier set provided by the user.
- a power set of another type such as \( \mathbb{P}(\mathbb{Z}) \).
- a cartesian product of two types such as \( \mathbb{Z} \times \text{BOOL} \).

Since for the remainder of this thesis we only need the fact that every term has a type and that types can be compared to each other we will limit our description of types to the above.

2.1.3 Terms and Formulas

A term is inductively defined as:

1. Any variable is a term.
2. If $f$ is an operator with arity $n$, and $f$ maps operands of types $\nu_1, \nu_2, \ldots, \nu_n$ to a value of type $\mu$ and $t_1, t_2, \ldots, t_n$ are terms of types $\nu_1, \nu_2, \ldots, \nu_n$, then $f(t_1, t_2, \ldots, t_n)$ is a term of type $\mu$.

3. If $x$ is a variable of type $\nu$ and $\phi$ is a formula, then $\{x|\phi\}$ is a term of type $\mathbb{P}(\nu)$.

A formula is defined as follows:

1. If $p$ is a predicate with arity $n$ on types $\nu_1, \nu_2, \ldots, \nu_n$ and $t_1, t_2, \ldots, t_n$ are terms of types $\nu_1, \nu_2, \ldots, \nu_n$, then $p(t_1, t_2, \ldots, t_n)$ is an atomic formula.

2. $\bot$ and $\top$ are atomic formulas.

3. If $\Phi$ is a formula, then so is $\neg \Phi$.

4. If $\Phi_1, \Phi_2$ are formulas, then so is $\Phi_1 \circ \Phi_2$, where $\circ$ ranges over $\lor, \land, \Rightarrow, \Leftrightarrow$.

5. If $\Phi$ is a formula and $x$ is a variable, then $\forall x \cdot \Phi$ and $\exists x \cdot \Phi$ are formulas.

For example:

- $1 + 1$ is a term. We wrote the $+$ operator in infix notation for better readability instead of $+(1, 1)$.

- $1 + 1 = 2$ is a formula. (Again we wrote $=$ as infix instead of $=(1 + 1, 2)$).

- $\forall x \cdot x \in \mathbb{N} \Rightarrow x + x = 2x$ is also a formula.

In [17] and in the Rodin source code a term is called expression and a formula is called predicate.

Sub-formula and Sub-Terms We use the $\sqsubseteq$ symbol to denote the sub-formula or sub-term partial order relation which is defined as follows:

1. $\phi \sqsubseteq \phi$, where $\phi$ is a formula.

2. $\phi \sqsubseteq \neg \phi$, where $\phi$ is a formula.

3. $\forall i \cdot i \in \{1, \ldots, n\} \Rightarrow t_i \sqsubseteq p(t_1, t_2, \ldots, t_n)$, where $p$ is a predicate or an operator and the $t_i$ are terms.

4. $\phi \sqsubseteq \phi \circ \psi$ and $\psi \sqsubseteq \phi \circ \psi$, where $\circ$ is one of $\land, \lor, \Rightarrow$ or $\Leftrightarrow$ and $\phi$ and $\psi$ are formulas.

5. $\phi \sqsubseteq \Delta x_1, \ldots, x_n \cdot \phi$, where $\Delta$ is either $\forall$ or $\exists$, the $x_i$ are variables and $\phi$ is a formula.

The relation has the usual properties of a partial order: reflexivity, transitivity and antisymmetry.
Size of a Formula. We define the size of a formula as the cardinality of the set containing all its sub-formulas and sub-terms. For example the formula $\forall a \cdot a \in A \Rightarrow a \geq 2$ has size eight due to its eight sub-formulas and terms: $\forall a \cdot a \in A \Rightarrow a \geq 2$, $a \in A \Rightarrow a \geq 2$, $a \in A$, $a$, $A$, $a$, $2$. Note that sub-terms or sub-formulas that occur in multiple locations in the original formula are counted multiple times. In the example $a$ is counted twice.

2.1.4 Operators

Operators are used to construct new terms; for example the + operator creates a new term out of two existing ones. In general an operators maps $n$ terms, the operands, of types $\nu_1, \nu_2, \ldots, \nu_n$ to a term of type $\mu$. The number $n$ is called **arity**. For example the + operator has arity two because it must be applied to two terms. For a complete description of the operators in Event-B the reader may have a look at Chapter 5 in [1].

Constants. Operators with arity 0 are called **constants**. Event-B provides several pre-defined constants. The most commonly used ones are $\mathbb{Z}$, $\mathbb{N}$ and $\emptyset$. It must be noted that because sets can be defined over any type there is a $\emptyset$ constant for every possible set type.

2.1.5 Variables

Bound and free variables are defined as usual. The following definition was taken from [10].

An occurrence of a variable $x$ in a term $P$ is called

- **bound** if it is in the scope of a variable binder, such as $\forall x$ or $\exists x$.

- **bound and binding**, iff it is the $x$ in $\lambda x$, where $\lambda$ is a binder such as $\forall$ or $\exists$.

- **free** otherwise.

If $x$ has at least one binding occurrences in $P$, we call $x$ a **bound variable** of $P$. If $x$ has at least one free occurrence in $P$, we call $x$ a **free variable**.

Note that a variable can be both, free and bound in a term. For example: $P := f(x) = a \Rightarrow \exists x \cdot f(x) = a$. Here the first occurrence of $x$ is free, whereas the second occurrence is bound. Thus $x$ is a free and a bound variable of $P$.

In the Rodin platform the user can define “constants”, but they are treated internally like variables and therefore our definition of variable includes this kind of “constant”.

10
Axioms & Theorems
Before-After-Predicate
⊢
Modified specific invariant
evt/inv/INV

Figure 2.1: Proof obligation for invariant establishment in an INITIALISATION event.

2.1.6 Symbols
Using the previous definitions symbol is a synonym for free variable. We keep the notion of symbol since many papers (e.g. [16, 27, 22]) about relevance filtering use it. In Event-B symbols are called free identifiers. For example: $\forall i, j \cdot i \in \mathbb{N} \land j \in 1..n \Rightarrow gp(i) < gp(j)$ contains the symbols $n$ and $gp$. The variables $i$ and $j$ are bound and $\mathbb{N}$ is a constant. As one can see “function symbols” like $gp$ are just normal variables that represent a functional relation.

We sometimes use $\text{sym}(f)$ to denote the set of symbols occurring in the formula $f$.

2.1.7 Predicates
Event-B also provides some built-in predicates. They are listed below:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$=$</td>
<td>Equality</td>
</tr>
<tr>
<td>$\subseteq$</td>
<td>Subset or Equal</td>
</tr>
<tr>
<td>$&gt;$</td>
<td>Greater than</td>
</tr>
<tr>
<td>$\in$</td>
<td>Element of</td>
</tr>
<tr>
<td>$\leq$</td>
<td>Less or Equal</td>
</tr>
<tr>
<td>$\geq$</td>
<td>Greater or Equal</td>
</tr>
<tr>
<td>finite</td>
<td>Finiteness of a set</td>
</tr>
</tbody>
</table>

2.1.8 Sequent
A sequent is a formalized statement of provability. A sequent is usually written as $\Gamma \vdash \sigma$, where $\Gamma$ represents a sequence of formulas and $\sigma$, a single formula. Intuitively $\Gamma \vdash \sigma$ means that if all formulas in $\Gamma$ are true, then $\sigma$ is true. We call the formulas occurring in $\Gamma$ hypotheses and the formulas $\sigma$ the goal.

In an Event-B context the term proof obligation, or PO for short, is also frequently used. It refers to a sequent which must be proven by the user. For example for each invariant it must be proven that it holds after a machine’s INITIALISATION event. See Figure 2.1 for the template of a proof obligation of that type.
2.2 The Rodin Platform

The Rodin platform\(^1\) is a software program that implements the Event-B method\(^1\). It consists a modeling part that is used to create Event-B models, and an interactive theorem prover that is used to prove the correctness of the models. Below we describe the aspects of the Rodin platform that are relevant for this thesis.

2.2.1 The Provers

The Rodin tool provides several reasoners which work together to discharge a proof obligation. The word reasoner refers to an entity that modifies the proof tree. The reasoners can be divided into two categories: provers and non-provers.

The non-provers can apply simplification rules or apply a single proof rule, for example:

\[ H, \bot \vdash \phi \]

Every application of a non-prover is visible in the proof tree and usually easy to understand for the user. The purpose of the non-prover reasoners is mainly to assist the user when doing a manual proof, but they also increase the success rate of the provers.

The provers are Automated Theorem Provers (ATP) that apply a proof inference technique, such as resolution, and if they are successful, the Rodin tool will discharge the proof obligation. A prover does not give any output except success (the sequent could be proved), Timeout (the prover ran out of time) or Failure (the sequent can not be proved). Sometimes the prover will also output which hypotheses it used to prove the sequent.

In the following we will present the provers that are available in the Rodin tool. Each of them has its own weaknesses and strengths. The following information is taken from the Rodin wiki\(^2\):

**NewPP**

NewPP is the only prover that is included in the Rodin distribution. It uses a combination of unit resolution and the Davis Putnam algorithm. Its input language is first order logic with the $\in$ predicate. This means that Event-B formulas need to be translated to this form before being passed to the prover. For example $A \subseteq B$ is translated to $\forall x \cdot x \in A \Rightarrow x \in B$. The prover then further translates the formulas to CNF.

\(^1\)http://sourceforge.net/projects/rodin-b-sharp/
\(^2\)http://wiki.event-b.org
Strengths

1. NewPP records the hypotheses it uses for a proof.
2. NewPP has limited support for equational reasoning.
3. NewPP has a search depth parameter in addition to a time limit.

Weaknesses

1. There is no support for arithmetic. That means NewPP can discharge $1 = 1$ but not $1 + 1 = 2$.
2. New PP is unaware of some set theoretical axioms; in particular, $\exists A \cdot \forall x \cdot x \in A \iff x \in B \lor x \in C$, because the union axiom is not available within NewPP. Roughly spoken, NewPP can only reuse sets that already appear in the formula, but it is unable to introduce new sets.

PP

PP is a prover for Atelier B\(^3\) which is a tool for working with the B method\(^2\). Since the B method is considered the predecessor of Event-B, it is also called classical B.

The prover itself is closed source and must be downloaded in binary form from the Atelier B website. The input format is the language of classical B. After the translation from Event-B to classical B has been done, the prover operates in a manner similar to NewPP but with support for arithmetic. On the downside PP does not record the hypotheses it uses.

ML

ML (Mono Lemma prover) is another prover for Atelier B. This means it is also closed source and the input language is also classical B.

In contrast to the other two provers ML is not a resolution based prover. Instead it applies a mix of forward, backward and rewriting rules in order to discharge the goal (or detect a contradiction among hypotheses).

Strengths

1. ML has limited support for equational and arithmetic reasoning.

\( ^3\text{http://www.atelierb.eu} \)
2. ML is more resilient to unnecessary hypotheses than newPP or PP. For that reason the Rodin tool always passes all hypotheses to the ML prover and does not offer the option to only pass the selected ones as for PP and NewPP.

Weaknesses

1. ML does not record the hypotheses it uses for a proof.

2. ML also does not know all set axioms.

2.2.2 Tactics

As mentioned above the Rodin platform uses several reasoners to prove a proof obligation. A set of instructions on how to apply and combine reasoners is called a tactic. Rodin offers a number of pre-defined tactics, most of which simply encapsulate a single reasoner call. However there are also two user configurable tactics: the post-tactic and the auto-tactic. These are combinations of the pre-defined tactics.

The post tactic is applied after every manual proof step in an interactive proof. Since we are mostly interested in automatic proofs we care more about the auto-tactic. This is the tactic used by Rodin if the user tells it to prove a proof obligation automatically. A list of the tactics which can be used in the auto/post tactic can be found in section 4.2.

The way in which the tactics are combined in the auto-tactic is rather simple: The Rodin tool tries to apply the tactics contained in the auto-tactic in the order defined by the user. This is repeated until either the proof obligation is discharged or none of the tactics is applicable anymore.

2.2.3 Proving Interface

The Rodin tool offers an interactive proving environment that the user has to use when the auto-tactic fails to prove a proof obligation. Figure 2.2 shows a screenshot of the proving environment.

On the left hand side we see the proof tree which is a top down view of the proof tree. Each reasoner application is visible there. In the middle you find, from top to bottom, the list of currently selected hypotheses, the goal and the proof control panel. In Rodin a hypothesis can be “selected”, “visible” or “hidden”. If a hypothesis is “visible” it can be selected by the user and then becomes “selected”. Also, new hypotheses that are created by a proof rule are automatically selected: for example when a quantified formula is instantiated. A hypothesis becomes “hidden” when a reasoner decides to hide it. Usually this happens when a simplification rule is
applied to a formula. Since the simplified formula subsumes the unsimplified one, the unsimplified version is hidden by the reasoner.

The purpose of “selected” hypotheses, besides seeing them in an easily accessible window, is that they will be used as input for provers that are run in “restricted” mode. The initial selection of hypotheses at the root of a proof tree is based only on the type of the proof obligation. For example for invariant preservation proofs of events, the proof obligation generator selects the invariant whose preservation has to proved and the guards of the event.
3 Filtering Techniques

In this section we will describe several relevance filtering or techniques. The purpose of a relevance filter is to select hypotheses that are relevant for proving a goal or in other words filter out hypotheses that are not relevant for proving a goal. The filtering methods we found in the literature can roughly be divided into the following three categories:

1. Syntactic methods
   These are methods that try to determine the relevance of a formula by looking at its syntactical features. An example for this category is a filter that chooses formulas based on the number of symbols it shares with the goal.

2. Abstraction based methods
   These are methods that try to obtain a proof for an abstracted version of the sequent and then try to derive the set of relevant formulas from the abstract proof.

3. Adaptive methods
   These are methods that use information from other already completed proofs in the same domain to estimate the relevance of hypotheses for the current proof in the hope that the proofs might be similar.

In this thesis we focus on syntactic methods. Abstraction based methods seem to require very close interaction with a theorem prover. Since none of the provers that are available for Event-B seem to be actively developed any more and only for NewPP the source code is available, it was decided to not further investigate this area.

Adaptive methods have been found to be quite successful in [16]. However, they dismiss them because adaptive methods are hard to evaluate due to their adaptive nature. Hence the results depend a lot on the order in which the proofs are attempted. Also the Event-B context is different to the one examined in [16] because in Event-B any hypothesis is very likely to be relevant for at least one proof, while in [16] there seem to exist many hypotheses that have no relation to the problem at all.

We will now take a closer look at specific filtering techniques.
3.1 Lasso

The Lasso is a syntactic method that is already present in the Rodin platform. It selects all hypotheses that have a symbol in common with either the goal or a selected hypothesis.

3.2 Whitelisting

During the evaluation of the filters we noticed that whitelisting certain types of formulas in the selection process had a tremendous positive effect on filter performance (See Section 4.8.1 on page 52). Our white listing method is dynamic and uses pattern matching to whitelist hypotheses in contrast to what Meng & Paulson describe in [16] where they hand pick certain hypotheses for white- or blacklisting.

The formulas that we decided to whitelist are those that give information about the type of a variable or fix a variable at a concrete value or value range. Examples for this are $x \in A \rightarrow B$, $x \in \text{dom}(f)$ or $x \leq 1$.

3.2.1 Details

For a given symbol $x$ the algorithm tries to find a formula or formulas of the form $x \alpha \beta$ or $\neg x \alpha \beta$.

Allowed values for $\alpha$ are built-in binary predicates such as $\in$, $=$ or $\subset$ while $\beta$ must be one of the following:

- A term of the form $t \delta u$ where $t$ and $u$ are terms and $\delta$ is a set of relation operator. Typical examples are $a \rightarrow b$, $a \rightarrow b$ or $a \rightarrow b$. These operators generate sets of relations:

<table>
<thead>
<tr>
<th>Relation</th>
<th>Total relation</th>
<th>Surjective relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftrightarrow$</td>
<td>$\leftrightarrow$</td>
<td>$\leftrightarrow$</td>
</tr>
<tr>
<td>Surj. total rel.</td>
<td>Partial function</td>
<td>Total function</td>
</tr>
<tr>
<td>Partial injection</td>
<td>Total injection</td>
<td>Partial surjection</td>
</tr>
<tr>
<td>Total surjection</td>
<td>Total bijection</td>
<td></td>
</tr>
</tbody>
</table>

For example $\mathbb{N} \rightarrow \mathbb{N}$ represents the set of all total bijective functions from $\mathbb{N}$ to $\mathbb{N}$.

- A term of the form $\text{dom}(t)$ or $\text{ran}(t)$, where $t$ is a term.
- A symbol.
- A numeric or boolean literal value such as $1$ or TRUE.
- A term of the form $\text{max}(t, u)$ or $\text{min}(t, u)$, where $t$ and $u$ are terms.
- A term of the form $\text{card}(t)$, where $t$ is a term.
• A term of the form $\mathbb{P}(t)$, where $t$ is a term.

The decision what to whitelist is based on experience and more or less ad hoc. Usually the whitelisting is applied to all symbols occurring in any formula in the set of relevant hypotheses output by a relevance filter.

### 3.3 Algorithm by Meng and Paulson

The algorithm from [16] is a simple syntactic filter. When rating a formula for its relevance for proving the goal, it assigns a *clause mark* to it which is based on the symbols contained in a formula and a *set of relevant symbols* which is initialized with the symbols contained in the goal.

The clause mark of a formula $f$ is defined as:

$$\text{clauseMark}(f, \text{relSym}) = \frac{|\text{sym}(f) \cap \text{relSym}|}{|\text{sym}(f)|},$$

where $\text{relSym}$ is the current set of relevant symbols.

The clause mark can take values between 0 and 1 and it expresses how many symbols that are not in the set of relevant symbols occur in a hypothesis. If only symbols from the set of relevant symbols occur in a hypothesis, its clause mark is 1 and is thus considered very relevant. If a hypothesis contains only symbols that are not in the set of relevant symbols, its clause mark will be 0 and thus considered not relevant.

The structure of a hypothesis is not taken into account when calculating the clause mark because a hypothesis is viewed simply as a set of symbols. In contrast to the original paper, where the notion of *symbol* also includes predicates and constants, we use our notion of symbols as defined in Section 2.1.6. This is motivated by the fact that all predicates and constants are pre-defined in Event-B. The corresponding axioms are built directly into the provers and cannot be selected by the user.

The filtering algorithm works by building a *set of relevant hypotheses*. In every iteration of the algorithm each hypothesis’ clause mark is recalculated and then compared to a *pass mark*. If the clause mark of a hypothesis is greater than the pass mark, the hypothesis is added to the set of relevant hypotheses and its symbols are added to the set of relevant symbols. The pass mark is increased in each iteration so that it becomes more difficult for new hypotheses to be added to the set of relevant hypotheses. How fast the pass mark increases depends on the convergence parameter $c$ which must be $\geq 1$. The formula $p' = p + \frac{1}{1+p}$ causes the pass mark to converge geometrically to 1. The larger $c$ is, the slower the convergence.
Algorithm 1 shows the complete algorithm in pseudo code. After the algorithm has computed the set of relevant hypotheses, the whitelisting method from Section 3.2 is used for all symbols in the set of relevant symbols.

Algorithm 1 MengPaulson(passMark, c)

\[
\begin{align*}
\text{passMark}: & \quad \{\text{Input: Initial pass mark}\} \\
c: & \quad \{\text{Input: Convergence parameter}\} \\
\text{relevantSymbols} := & \quad \text{sym(goal)} \\
\text{workingSet} := & \quad \{\text{All hypotheses}\} \\
\text{relevantHypotheses} := & \quad \emptyset \quad \{\text{Output: Set of accumulated relevant hypotheses}\} \\
done := & \quad FALSE \quad \{\text{Boolean variable that indicates when to stop}\} \\
\text{while } & \quad \neg \text{done} \text{ do} \\
& \quad \text{newRelevantSymbols} := \emptyset \\
& \quad \text{done} := TRUE \\
& \quad \text{for all } f \in \text{workingSet} \text{ do} \\
& \quad \quad \text{if } \text{clauseMark}(f, \text{relevantSymbols}) \geq \text{passMark} \text{ then} \\
& \quad \quad \quad \text{done} := \text{FALSE} \\
& \quad \quad \quad \text{workingSet} := \text{workingSet} - \{f\} \\
& \quad \quad \quad \text{relevantHypotheses} := \text{relevantHypotheses} \cup \{f\} \\
& \quad \quad \quad \text{newRelevantSymbols} := \text{newRelevantSymbols} \cup \text{sym}(f) \\
& \quad \quad \text{end if} \\
& \quad \text{end for} \\
& \quad \text{relevantSymbols} := \text{relevantSymbols} \cup \text{newRelevantSymbols} \\
& \quad \text{passMark} := \text{passMark} + \frac{1-\text{passMark}}{c} \\
\text{end while} \\
& \quad \text{for all } s \in \text{relevantSymbols} \text{ do} \\
& \quad \quad \text{relevantHypotheses} := \text{relevantHypotheses} \cup \{\text{whitelisted hypotheses for } s\} \\
\text{end for} \\
& \quad \text{return } \text{relevantHypotheses}
\end{align*}
\]
3.4 SInE Filter

SInE [12] (Sumo Inference Engine) is a filtering strategy developed by Krystof Hoder initially targeted for SUMO [19], but it has also been successful in the CASC-J4 [26] competition.

3.4.1 Details

SInE is a syntactic filtering strategy based on the assumption that for every symbol there exist some axioms that “give the symbol its meaning”. Therefore the filter creates a \( D \)-relation that puts symbols into relation with axioms that “define” it. The algorithm can be divided into two phases:

Setup Phase

In the setup phase the algorithm computes in how many hypotheses a symbol occurs. In the original documentation this is called “generality index”. If the generality index for symbol \( x \) is 5, then symbol \( x \) occurs in 5 hypotheses. Based on this index each hypothesis is put into \( D \)-relation with the least general symbol (the one with the lowest generality index) it contains. Should there be more than one least general symbol, the hypothesis is put into \( D \)-relation with all of them.

Execution Phase

In the execution phase the set of relevant hypotheses (RH) is built. The algorithm starts by putting the symbols occurring in the goal into the set of relevant symbols (RS). The algorithm then adds all hypotheses which are \( D \)-related to the symbols in RS to RH. For every hypothesis \( h \) added to RH the algorithm adds \( sym(h) \) to RS. This is repeated until no more hypotheses are added to RH. As with the Meng & Paulson algorithm we use the whitelisting method from Section 3.2 for all symbols \( s \in RS \).

3.5 Direct (Contextual) Relevance Filter

The notion of direct relevance and direct contextual relevance was used in several papers by Geoff Sutcliffe [28, 24], but he also attributes it to other people.

Direct relevance is another measure about how much symbol overlap there is between two formulas \( f_a \) and \( f_b \). It is defined as

\[
\frac{|sym(f_a) \cap sym(f_b)|}{|sym(f_a) \cup sym(f_b)|}
\]
Figure 3.1: Example for direct relevance.

Edges are annotated with the direct relevance between the formulas.

*Direct contextual relevance* takes rarity of symbols into account. It tries to capture the intuitive notion that if a very common symbol occurs in two formulas it does not necessarily mean they are particularly relevant to each other, while a rare symbol occurring in two formulas might indicate a higher relevance.

The formula for direct contextual relevance replaces the numerator of the above formula such that it becomes

\[
\sum_{s \in (\text{sym}(f_a) \cap \text{sym}(f_b))} \left( 1 - \frac{|\{f, s|f \in S, s \in \text{sym}(f)\}|}{|S|} \right)
\]

where is \( S \) is the set of all hypotheses and the goal. The numerator now sums up the symbol weights of the symbols occurring in both, \( f_a \) and \( f_b \). The symbol weight is 0 for a symbol that occurs in all hypotheses and larger (approaching 1) for symbols that occur in fewer hypotheses.

Besides the direct (contextual) relevance between two formulas there might also be *indirect relevance* between two formulas if they are both directly relevant to a third formula. The indirect relevance between two formulas \( f_a \) and \( f_b \) is determined by examining the direct relevance between intermediate formulas \( f_i \) and \( f_{i+1} \) in all paths \( f_a = (f_1), (f_2), \ldots, (f_n) = f_b \). The relevance measure for such a path is the smallest direct relevance on the path divided by the path length. This captures the fact that a path’s strength is determined by its weakest link. The indirect relevance between two formulas is the maximal indirect relevance over all paths connecting the formulas. For example in figure 3.1 the indirect relevance between \( f_a \) and \( f_b \) along the path \( (f_a), (f_3), (f_b) \) is \( \frac{0.3}{3} = 0.1 \) while the indirect relevance along the path \( (f_a), (f_4), (f_b) \) is \( \frac{0.4}{2} = 0.2 \). This means the effective indirect relevance between \( f_a \) and \( f_b \) is 0.2.
3.5.1 Implementation

Our implementation of the direct relevance filter is as follows:

Let $S$ again be the set of all hypotheses and the goal. Each hypothesis has a field for the relevance to the goal which is initialized with 0. Let $R$ be the set of relevant hypotheses that we are building.

1. Calculate the direct (contextual) relevance between any formula in $S$ and the goal. If a hypothesis has a direct relevance of more than 0 to the goal, store it in the relevance field.

2. Calculate the indirect (contextual) relevance between any hypothesis and the goal using a variant of Dijkstra’s shortest path algorithm. If the indirect relevance happens to be higher than the relevance stored in a formula’s relevance field update the field with the indirect relevance.

3. Add all formulas with a relevance to the goal higher than some threshold $t$ to the set $R$.

4. Use the whitelisting algorithm (see Section 3.2) for any symbol $s \in f : f \in R$ and also add those formulas to $R$.

3.6 A Filter based on Subexpression Matching

This filter is based on an idea by the author. It tries to mimic human behavior when trying to determine the relevance of a hypothesis to the proof of a goal. What the author often does when hand-picking hypotheses that could be relevant for the proof is the following:

1. Look for hypotheses that look similar to the goal and select them.

2. Look at the newly selected formulas and check if they contain symbols that need further “explanation” and select hypotheses that “explain” those symbols.

3. Repeat the previous step until the selection looks about right.

This filter is an attempt to turn this very informal description into an algorithm. What sets this filter apart from the other filters we looked at is that it takes a formula’s syntactical structure into account while the other filters only examine the symbols, regardless of where in a formula they occur. Analyzing the structure of an Event-B formula is more promising than analyzing CNF formulas because CNF formulas have a fixed nesting structure for logical connectives in contrast to Event-B formulas, where arbitrary nesting of connectives is possible.
3.6.1 The Algorithm

This section gives a detailed description of the filtering algorithm. The algorithm uses a similarity measure called *match mark*, denoted $m(h_1, h_2)$, which reflects the structural similarity between two formulas $h_1$ and $h_2$. The match mark outputs a value between 0 and 1, where 0 means not similar at all and 1 means very similar. The match mark is not symmetric. For details refer to Section 3.6.2.

**Initialization Step**

During initialization the algorithm tries to find formulas that are similar to the goal (see Algorithm 2). To do that it selects at most $n$ formulas $f_i$ that have match mark $m(f_i, goal)$ higher than some threshold $t$. We call $n$ the initial selection count and $t$ the initial selection threshold. If there are no such formulas it looks at formulas with a match mark $m(goal, f_i)$ higher than the threshold $t$. The decision to look for formulas that can be matched to the goal first and only after that for formulas that the goal can be matched to is mostly based on experiments.

**Algorithm 2** Initialization($n$, $t$)

| Hyps: {Set of all visible hypotheses} |
| goal: {Goal that has to be proved} |
| $t$: {Threshold parameter} |
| $n$: {Initial selection count} |
| $RH :=$ Set of at most $n$ hypotheses such that $\forall h, g \cdot (h \in RH \land g \in Hyps \setminus RH) \Rightarrow (m(h, goal) \geq m(g, goal) \land m(h, goal) \geq t)$ |
| if $RF = \emptyset$ then |
| $RH :=$ Set of at most $n$ hypotheses such that $\forall h, g \cdot (h \in RH \land g \in Hyps \setminus RH) \Rightarrow (m(goal, h) \geq m(goal, g) \land m(goal, h) \geq t)$ |
| end if |
| return $RH$ |

**Execution Step**

After the initialization step the algorithm performs several execution steps. See Algorithm 3. A step is one execution of the while loop. The initialization step and the subsequent execution steps can also be seen as waves (See figure 3.2). This way of looking at it emphasizes how the algorithm starts out from the goal (wave 0) and adds more and more hypotheses to explain the symbols contained in the hypotheses added in the previous waves.

The selection of hypotheses to add is done by the “explain” function. Abstractly it maps a set of unexplained symbols of a formula to a set of hypotheses that
“explain” those symbols. For details see Section 3.6.3. The set $ES$ is used to focus the selection of new hypotheses on previously unknown symbols.

**Algorithm 3** Subexpression Filter

**goal:** \{Goal that has to be proved\}

**Hyps:** \{Set of all available hypotheses\}

$ES := \emptyset$ \{Set of overall explained symbols\}

$RH := Initialization(n, t)$: \{Output: Set of relevant hypotheses, $n$ and $t$ are parameters for the initialization step\}

$SF := RH \times \{goal\}$ \{Pairs of hypotheses, where the second hypothesis caused the addition of the first hypothesis\}

**while** $SF \neq \emptyset$ **do**

$NewES := \emptyset$

$NewSF := \emptyset$

**for all** pairs $(h_1, h_2) \in SF$ **do**

\{ $A \triangle B$ denotes the symmetric difference: $A \setminus B \cup B \setminus A$ \}

$SU := (sym(h_1) \triangle sym(h_2)) \setminus ES$

$H := explain(h_1, SU, Hyps, RH)$

**for all** $g \in H$ **do**

$newSF := newSF \cup \{(g, h_1)\}$

$newES := newES \cup sym(g)$

**end for**

**end for**

**for all** pairs $(h_1, h_2) \in newSF$ **do**

$RH := RH \cup \{h_1\}$

**end for**

$SF := newSF$

$ES := ES \cup newES$

**end while**

**for all** $g \in RH$ **do**

$RH := RH \cup \{Whitelisted hypotheses for sym(g). See Section 3.2\}$

**end for**

return $RH$

---

**3.6.2 Similarity Measure**

The similarity measure is based on the assumption that hypotheses that are relevant for the proof of a goal have structural similarities with the goal. This can be seen as an extension of the assumption made by other relevance filters that relevant hypotheses share symbols with the goal. As mentioned before, we call the
similarity measure *match mark* and denote it by \( m(f_1, f_2) \), where \( f_1 \) and \( f_2 \) are formulas. The match mark can intuitively be described as the degree to which the first formula \( f_1 \) occurs in the second formula \( f_2 \). This implies that the match mark is not symmetric. The match mark is a value between 0 and 1. A match mark of 0 means that no sub-formula or sub-term of \( f_1 \) occurs in \( f_2 \) whereas a match mark of 1 means that \( f_1 \) is a sub-formula of \( f_2 \), albeit with some relaxations as we will see later. A match mark between 0 and 1 means that a sub-formula or sub-term of \( f_1 \) occurs in \( f_2 \).

Before we continue we need to give some definitions:

\[
\text{pos}(x) = \begin{cases} 
+1 & x > 0 \\
0 & x \leq 0
\end{cases}
\]

\[
\text{sub}(f) := \text{Set of all terms and formulas } \subseteq \text{-related to } f \text{ (See Section 2.1.3)}.
\]

With \( \text{sub}(f) \) we denote the set of all sub-terms and sub-formulas of \( f \), which can easily be generated using the syntax tree of a formula. Figure 3.3 shows an example of such a tree. In this example the formula contains 13 sub-formulas and terms. Every node in the syntax tree is the root of a sub-formula or sub-term.

The match mark is defined as

\[
m(f_1, f_2) := \frac{\max_{u \in \text{sub}(f_1), v \in \text{sub}(f_2)} (mc(u, v))}{|\text{sub}(f_1)|},
\]

where \( mc(a_1, a_2) \) stands for the *match count* which is is the number of sub-formulas or terms of \( a_1 \) that occur in \( a_2 \). It is inductively defined as follows:

1. If \( a_1 \) is a bound variable and \( a_2 \) is term (or vice versa) and \( a_1 \) has the same type as \( a_2 \), then \( mc(a_1, a_2) := 1 \).
Figure 3.3: Syntax tree of the formula $\forall r \cdot r \in R \Rightarrow \text{nxt}(r) \subseteq \text{res}\{r\}$

Note: When computing the set of sub-formulas and sub-terms of a formula $f$ and a sub-formula or sub-term $f_1$ of $f$ contains a variable $x$ that is bound in $f$ but free in $f_1$, then $x$ in $f_1$ is still considered bound for this comparison.

2. If $a_1$ and $a_2$ are both free variables and $a_1 = a_2$, then $mc(a_1, a_2) := 1$.

3. If
   - $a_1$ is of the form $f(s_1, s_2, \ldots, s_n)$, where $f$ is an operator or predicate with arity $n$ and $s_1, s_2, \ldots, s_n$ are terms and
   - $a_2$ is of the form $g(r_1, r_2, \ldots, r_n)$, where $g$ is an operator or predicate with arity $n$ and $r_1, r_2, \ldots, r_n$ are terms and
   - $f \approx g$,
   then
     \[
     mc(a_1, a_2) := \sum_{j=1}^{n} \left( mc(s_j, r_j) + \text{pos}(mc(s_j, r_j)) \cdot \frac{1}{n} \right).
     \]
   The $\approx$ symbol stands for a relaxed equality which will be explained later. The purpose of the $\text{pos}(mc(s_j, r_j)) \cdot \frac{1}{n}$ part of the formula is to make the contribution of the matching operator or predicate proportional to the number of matching operands. This is necessary to avoid matching two formulas only on their operators or predicates. For example if $f_1 := a \subseteq b$ and $f_2 := a \subseteq d$, then $mc(f_1, f_2) = 1.5$. If $f_3 := c \subseteq d$, then $mc(f_1, f_3) = 0$.

4. If
   - $a_1$ is of the form $\{x|\phi_1\}$, where $\phi_1$ is a formula and $x$ is a variable and
   - $a_2$ is of the form $\{x|\phi_2\}$, where $\phi_2$ is a formula and $x$ is a variable,
then
\[
mc(a_1, a_2) := mc(\phi_1, \phi_2) + pos(mc(\phi_1, \phi_2)).
\]

5. If \( a_1 = \Delta \) and \( a_2 = \Delta \), where \( \Delta \) is either \( \top \) or \( \bot \), then \( mc(a_1, a_2) := 1 \).

6. If
   - \( a_1 \) is of the form \( \neg \phi_1 \) and
   - \( a_2 \) is of the form \( \neg \phi_2 \),

then
\[
mc(a_1, a_2) := mc(\phi_1, \phi_2) + pos(mc(\phi_1, \phi_2)).
\]

7. If
   - \( a_1 \) is of the form \( \phi_1 \star_1 \phi_2 \), where \( \phi_1 \) and \( \phi_2 \) are formulas and \( \star_1 \) is one of \( \wedge, \lor, \Rightarrow, \Leftrightarrow \) and
   - \( a_2 \) is of the form \( \psi_1 \star_2 \psi_2 \), where \( \psi_1 \) and \( \psi_2 \) are formulas and \( \star_2 \) is one of \( \wedge, \lor, \Rightarrow, \Leftrightarrow \) and
   - \( \star_1 = \star_2 \),

then
\[
mc(a_1, a_2) := mc(\phi_1, \psi_1) + mc(\phi_2, \psi_2) + \\
\frac{1}{2} \cdot (pos(mc(\phi_1, \psi_1)) + pos(mc(\phi_2, \psi_2))).
\]

8. If \( a_1 \) is of the form \( \Delta x_1, x_2, \ldots, x_k \cdot \phi_1 \) and \( a_2 \) is of the form \( \Delta y_1, y_2, \ldots, y_j \cdot \phi_2 \),
   where \( \Delta \) is either \( \forall \) or \( \exists \) and the \( x_i \) and \( y_i \) are variables, then
\[
mc(a_1, a_2) := mc(\phi_1, \phi_2) + pos(mc(\phi_1, \phi_2)).
\]

In all other cases \( mc(a_1, a_2) := 0 \).

**Examples**

Some examples for the match mark measure:

- \( f_1 : \forall r \cdot r \in R \Rightarrow \text{nxt}(r) \subseteq \text{res} \{ \{ r \} \} \)
- \( f_2 : \text{nxt}(r) \subseteq \text{res} \{ \{ r \} \} \)
- \( f_3 : \forall r \cdot r \in R \Rightarrow \text{lst}(r) \subseteq \text{res} \{ \{ r \} \} \)

Now we have
• \( m(f_1, f_1) = 1 \)
• \( m(f_1, f_2) = \frac{8}{13} \)
• \( m(f_2, f_1) = 1 \)
  All of \( f_2 \) occurs in \( f_1 \)
• \( m(f_1, f_3) = \frac{11.5}{13} \)

Only one node (\( \text{lst vs. nxt} \)) does not match when comparing the full formulas. Note that the function application \( \text{nxt}(r) \) consists of an operator with two children: \( \text{nxt} \) and \( r \).

**Relaxed Equality**

In our definition of the similarity measure we used \( \approx \) to denote an equivalence relation on predicates and operators. The \( \approx \) relation extends the normal equality relation (=) with the following additional relations:

- \( "\leq" \approx "\leq" \)
- \( "\leq" \approx "\geq" \)
- \( "\leq" \approx "\leq" \)
- \( "\geq" \approx "\geq" \)
- \( "\leq" \approx "\leq" \)
- \( "\subset" \approx "\subset" \)

These extensions are based on the observation that for example the hypothesis \( a \subseteq b \) is relevant to a proof of \( a \subset b \), yet using normal equality to compare these two formulas with our match mark measure would yield a match mark of 0. The above extensions of the equality relation fixes this issue.

**3.6.3 How to Explain Symbols**

In the execution step of the algorithm hypotheses that “explain” a set of symbols have to be found. Initially we wanted to use only the similarity measure to decide which hypotheses to use. This, however, was not very successful because too often necessary hypotheses did not have much structural similarity with the formula that contained the unexplained symbols. Therefore we decided to use multiple relevancy measures and select the hypothesis that is considered relevant by most of them.

An important design principle was to select few formulas since the algorithm was supposed to work on “hard” problems which require a relatively accurate selection of relevant hypotheses. Unfortunately, this also means that our algorithm will sometimes not select a required hypothesis, but in turn it is able to make accurate selections for other problems. We will now motivate and define the relevancy
criteria we used and then give an algorithm that maps a set of unexplained symbols that occur in a hypothesis to a set of other hypotheses that explain the symbols.

A hypothesis will only be considered for selection if it contains at least one symbol from the unexplained symbol set. If this condition is satisfied, a hypothesis is assigned credits by a number of relevancy criteria. We define the function credits$(f_c, f_e, SU, Hyps, RH)$, where $f_c$ is the candidate formula for which the credits are calculated, $f_e$ is the formula whose symbols need to be explained, $SU$ is the set of symbols of $f_e$ that need to be explained, $Hyps$ is the set of all hypotheses and $RH$ is the set of hypotheses that are already considered relevant. The criteria used in the credits function are:

1. Highest number of unexplained symbols. (2 credits)
   The credit is given to the hypothesis that contains the highest number of symbols from the unexplained symbols set ($SU$). The motivation behind this criterion is that we want to select few hypotheses and therefore we want to cover as many symbols as possible with as few formulas as possible.

2. Largest match (1 credit)
   When the match mark $m(f_1, f_2)$ is equal to 1, we say that $f_1$ can be fully matched to $f_2$. The largest match for two formulas $f_1$ and $f_2$ is the largest sub-formula or sub-term that can be fully matched to $f_2$. Let $lm(f_1, f_2)$ denote the size of the largest full match between $f_1$ and $f_2$. The credit is given to the hypothesis $f_c$ that satisfies the following condition: $\forall f \cdot f \in Hyps \Rightarrow lm(f, f_e) \leq lm(f_c, f_e)$. This criterion works under the assumption that if a hypothesis contains a large part of another one, they are likely to be relevant to each other.

3. If the largest match has opposite polarity (1 credit)
   Credit is given to any hypothesis $f_c$ whose largest match occurs with opposite polarity in $f_e$ as in $f_e$. See below for a definition of polarity.

4. Highest clause mark (1 credit)
   This is the same calculation as in the Meng & Paulson filter (See Section 3.3). The clause mark $clauseMark(f_c, RH)$ is computed and the credit is given to the formula with the highest clause mark. If two formulas have the same clause mark, credit is given to both. The purpose of this criterion is to prevent the addition of too many new symbols.

5. Highest aggregate match mark (1 credit)
   This is a measure for the overall importance of a formula. It is not goal directed and is defined as: $\sum_{g \in Hyps \setminus f} m(f, g)$, where $f$ is the candidate formula, $Hyps$ is the set of all hypotheses and $m(\cdot, \cdot)$ is the match mark defined
in Section 3.6.2. Again this gives preference to “popular” formulas that have a higher chance of being selected and therefore the overall amount of selected formulas is minimized.

Algorithm 4 explain($h_e, SU, Hyps, RH$)

$Hyps$: {Input: Set of all visible hypotheses}
$RH$: {Input: Set of relevant hypotheses}
$h_e$: {Input: Hypothesis whose symbols need to be explained}
$SU$: {Input: Symbols in $h_e$ that need to be explained}
$Result := \emptyset$ {Output: Set of hypotheses that explain the symbols in $SU$}

while $SU \neq \emptyset$

$h_q := \forall h \cdot h \in Hyps \Rightarrow$

\[\operatorname{credits}(h_q, f_e, SU, Hyps, RH) \geq \operatorname{credits}(h, f_e, SU, Hyps, RH)\]

$Result := Result \cup \{h_q\}$

$SU := SU \setminus \text{sym}(h_q)$

end while

Polarity

Every term and sub-formula of a formula can be assigned a polarity. The polarity is either $\{+, -\}$, $\{-\}$ or $\{+, -, -\}$. The inverse of a polarity value $p$ denoted as $\neg p$ is defined as follows:

1. If $p = \{+\}$, then $\neg p = \{-\}$.
2. If $p = \{-\}$, then $\neg p = \{+\}$.
3. If $p = \{+, -\}$, then $\neg p = \{+, -, -\}$.

We define polarity as a mapping $\text{pol}(h, f) \rightarrow \mathbb{P}(\{+, -, -\})$, where $f$ is a formula and $h$ is a sub-term or sub-formula occurring in $f$. For example $\text{pol}(a, \neg((- a \in A) \Rightarrow ((b \in A) \land \neg(c \in A)))) = \{+\}$. Figure 3.4 shows the syntax tree annotated with polarities.

Definition A term has the same polarity as the surrounding formula. The polarity of a sub-formula $\Phi$ occurring in a formula $\Psi$ (i.e. $\Phi \subseteq \Psi$) is defined as follows:

1. If $\Phi = \Psi$, then $\text{pol}(\Phi, \Psi) := \{+\}$.
2. If $\Phi = \neg \Phi_1$ and $\text{pol}(\Phi, \Psi) = p$, then $\text{pol}(\Phi_1, \Psi) := \neg p$. 
3. If $\Phi = \Phi_1 \circ \Phi_2$, $\text{pol}(\Phi, \Psi) = p$ and $\circ$ is
   - $\Rightarrow$, then $\text{pol}(\Phi_1, \Psi) := \neg p$ and $\text{pol}(\Phi_2, \Psi) := p$.
   - $\Leftrightarrow$, then $\text{pol}(\Phi_1, \Psi) := \{+, -\}$ and $\text{pol}(\Phi_2, \Psi) := \{+, -\}$.
   - $\land$ or $\lor$, then $\text{pol}(\Phi_1, \Psi) := p$ and $\text{pol}(\Phi_2, \Psi) := p$.

4. If $\Phi = \forall x_1, x_2, \ldots, x_n \cdot \Phi_1$ or $\Phi = \exists x_1, x_2, \ldots, x_n \cdot \Phi_1$ and $\text{pol}(\Phi, \Psi) = p$, then
   $\text{pol}(\Phi_1, \Psi) := p$.

Note: we assume that the convenience predicates $a \neq b$, $a \not\subseteq b$, $a \not\subset b$ and $a \not\in b$ are replaced by their definitions $\neg(a = b)$, $\neg(a \subseteq b)$, $\neg(a \subset b)$ and $\neg(a \in b)$ respectively.

Motivation  If a term or sub-formula $x$ occurs with negative polarity in formula $f_1$ and with positive polarity in formula $f_2$, this makes $f_2$ relevant for $f_1$ (and vice versa). The technical reason for this is that if a resolution based prover resolves two formulas $f_1$ and $f_2$, they contain the same symbol(s) in opposite polarity. A more intuitive reason is that if $f_1$ is a formula of the form $A \Rightarrow B$, another formula $f_2 := A$ is needed in order to apply the modus ponens rule and derive $B$. The formula $f_1 := A \Rightarrow B$ can be rewritten as $\neg A \lor B$ which makes it obvious that $A$ has negative polarity in $f_1$, but positive polarity in $f_2$. In a way checking polarity is a general and simple way to determine whether the modus ponens rule can be applied to two formulas.
3.6.4 Summary

The algorithm explained above is an attempt to exploit structural similarities in formulas to estimate their relevance to proving a goal. Unfortunately, it became obvious during the development of the algorithm that the syntactic similarity measure alone would not make for an efficient selection algorithm. We observed that there was often a required hypothesis with high structural similarity to the goal, but other required formulas did not have a significantly larger similarity to the goal or other already selected formulas than not relevant hypotheses. This forced us to introduce a seemingly random selection of measures to extend the relevant set in the execution phase of the algorithm. The assumption is, that when a formula is relevant, then it will satisfy a majority of the criteria.

It must also be said that the presented algorithm should be viewed more as an experiment than as an elaborate filtering algorithm. Its purpose is to explore ways of relevance filtering beyond the examination of shared symbols. We will see in the evaluation (see Chapter 4) that the sub expression based filter can help to solve problems that none of the other filters can solve.

3.7 Multi Filter

An idea that comes to mind when looking at the different heuristic filtering methods explained in this chapter is to combine them all into one filter and let each filter “vote” on the hypotheses. A hypothesis is then considered as relevant if more than \( n \) filters consider it to be relevant.

The implementation of such a filter is trivial and does not need to be explained here.
4 Evaluation

This section contains the results of the evaluation of the filtering techniques presented in Chapter 3. We used seven models to evaluate the various filters during their development and implementation and set aside a number of other models to rebut critique about over-fitting the filters to certain models (see Section 4.7.1).

Our main evaluation criterion is the number of manual proofs. Manual proofs are proofs that require help by the user in order to be discharged. Thus, the lower the number of manual proofs, the better the filter since the user has to spend less time discharging proof obligations. The main measure for our evaluations is the percentage of manual proofs.

4.1 The Models

Table 4.1 gives some information about the models we used to evaluate the filters. These models were more or less randomly selected from the models available on the Event-B wiki\footnote{http://wiki.event-b.org}, the Event-B book [1] and models developed by members of the author’s research group. The test set we used contained a total of 1424 proof obligations. We excluded models that did not have any manually discharged proof obligations with the existing selection strategies because it would not be possible to measure improvements with such models.

The arithmetic column in Table 4.1 indicates whether the model uses arithmetic comparison predicates such as $<$ or $\leq$, or makes use of arithmetic operators such as $+$. We mention this because not all provers support arithmetic reasoning.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>POs</th>
<th>Mch/Ctx</th>
<th>Arithmetic</th>
<th>Avg. problem size</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>A model of a cache coherence protocol for a multi-CPU architecture.</td>
<td>567</td>
<td>16/5</td>
<td>No</td>
<td>50</td>
<td>-</td>
</tr>
<tr>
<td>IEEE</td>
<td>A model of the IEEE 1394 leader election protocol.</td>
<td>67</td>
<td>6/3</td>
<td>No</td>
<td>15</td>
<td>ch. 13 in [1]</td>
</tr>
<tr>
<td>Train</td>
<td>A model of a train system which is supposed to help a train agent manage a train network.</td>
<td>133</td>
<td>5/3</td>
<td>No</td>
<td>29</td>
<td>ch. 17 in [1]</td>
</tr>
<tr>
<td>CtsCtrl</td>
<td>A model of a real-time controller for a water tank.</td>
<td>163</td>
<td>4/3</td>
<td>Yes</td>
<td>37</td>
<td>[4]</td>
</tr>
<tr>
<td>MyCache</td>
<td>Another model of a cache coherence protocol for a multi-CPU architecture developed by the author.</td>
<td>280</td>
<td>6/1</td>
<td>Yes (little)</td>
<td>61</td>
<td>-</td>
</tr>
<tr>
<td>Zer_ess</td>
<td>A proof of the well ordering theorem in Event-B.</td>
<td>55</td>
<td>0/4</td>
<td>No</td>
<td>13</td>
<td>[3]</td>
</tr>
</tbody>
</table>

Avg. problem size is the average number of visible hypotheses at the root of a proof tree.

Table 4.1: Models used for evaluation
4.2 Setup

All our evaluations were done using the Rodin platform version 1.2. Below we present the configuration details.

**Auto Tactic** As explained in Section 2.2.2 Event-B has a configurable auto tactic. Unless stated explicitly we will always use the following configuration:

1. True Goal (Discharge)
2. False Hypothesis (Discharge)
3. Goal in Hypothesis (Discharge)
4. Functional Goal (Discharge)
5. Bounded Goal with finite Hypothesis (Discharge)
6. ML (Discharge)
7. Partition Rewriter (Simplify)
8. Simplification Rewriter (Simplify)
9. Type Rewriter (Simplify)
10. Find Contradictory Hypotheses (Discharge)
11. Shrink implicative Hypotheses (Simplify)
12. Functional overriding in Goal (Split)
13. Clarify Goal (Mixed)
14. Functional overriding in Hypothesis (Split)
15. One point rule in Hypotheses (Split)
16. One point rule in Goal (Split)
17. Provers & Filters (Discharge)
18. Use Equals Hypotheses (Simplify)

The variable part is items number 17 and 6. They represent the prover reasoners which we will evaluate. We will refer to item 17 as “provers” and to item 6 as ML.

The difference between the above configuration and the Rodin default configuration is the addition of items 14, 15 and 16, the addition of the provers (item 17) and a different position for item 18. The reason for the latter will be explained in Section 4.8.3. The first two changes were done to get the maximum available proving capabilities.

For comparison: The Rodin default configuration achieves a manual proof rate of 28.79% in about 17.6 minutes. The above configuration without provers, but with ML, achieves a manual proof rate of 26.69% in 17.9 minutes. This means our configuration is two percentage points better at almost no cost.

**Provers** Unless specified otherwise the provers (item 17) were given a timeout of 2000ms (the Rodin default value). The restricted strategy is usually used with
Table 4.2: Filter shorthands

<table>
<thead>
<tr>
<th>Filter</th>
<th>Shorthand</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unrestricted</td>
<td>unres</td>
<td>-</td>
</tr>
<tr>
<td>Restricted</td>
<td>res</td>
<td>-</td>
</tr>
<tr>
<td>Lasso</td>
<td>lasso</td>
<td>-</td>
</tr>
<tr>
<td>Meng Paulson</td>
<td>RF(pm, c) or RF(pm)</td>
<td>c: convergence parameter; pm: initial pass mark;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if c not given: c = 1.2</td>
</tr>
<tr>
<td>Sub expression</td>
<td>SubExpr(n, t)</td>
<td>n: initial selection count; t: initial selection</td>
</tr>
<tr>
<td>Direct Relevance</td>
<td>DCR(t)</td>
<td>t: threshold</td>
</tr>
<tr>
<td>SInE</td>
<td>SInE</td>
<td>-</td>
</tr>
<tr>
<td>Multi</td>
<td>Multi</td>
<td>-</td>
</tr>
</tbody>
</table>

a prover timeout of 500ms (the Rodin default). The search depth parameter of NewPP was set to 10000. We could not use it with unrestricted search depth due to memory constraints. The default value is 3000. Since our value is more than three times the default, the danger of artificially crippling the prover is low.

The ML prover is run using the default settings; that is a timeout of 500ms and the use of force 0 and 1. The reason for treating ML differently is that it uses a very different proving technique. See Section 2.2.1.

**Hardware** The experiments were run under Microsoft Windows Server 2003 (64bit) on a PC with eight E5410 Intel Xeon processor cores running at 2.33 GHz. Rodin and the external provers were run in 32bit mode. Since the provers do not make use of multiple threads, the number of CPUs should influence the results only slightly.

### 4.3 Filters and Abbreviations

In order to make it easier to refer to a filter with specific settings we introduce shorthands for them. See Table 4.2.

The unrestricted strategy selects all visible hypotheses – hence unrestricted. The restricted strategy limits the prover to the hypotheses that are currently selected; thus it is not really a filtering strategy. All other filtering strategies are explained in Chapter 3. Unless explicitly stated the relevance filters are always used with both provers: PP and NewPP.
Timeouts  Sometimes we use other timeouts for the provers than the default 2000 ms we mentioned above. In our notation this is indicated by a time value in square brackets. For example “res[500ms]” means that a prover timeout of 500 ms is used for each prover when using the restricted filter.

4.4 Effects of Relevance Filtering

In this section we look at the effects relevance filters have in the Event-B context and compare the various filtering techniques described in Chapter 3 to each other. In general all proof obligations fall into one of the following three categories:

1. Can be discharged within the prover timeout in unrestricted mode.

2. Can be discharged within the prover timeout if (a sufficiently small superset of) the necessary hypotheses are selected.

3. Can not be discharged within the prover timeout.

The first and the third category are not interesting for us. We only care how much of the second category can be discharged automatically with the help of relevance filtering. If a filter fails on a proof obligation from the second category it is for one or both of the following reasons:

1. The filter selects too much.

2. The filter does not select a necessary hypothesis.

Heuristic relevance filters always have to find the balance between being strict to avoid the first problem and being relaxed enough to avoid the second problem.

Table 4.3 shows the performance of the relevance filters and the unrestricted strategy. The parameters of the filters were chosen to deliver the best results in this setting. We see that all relevance filters show better performance than the unrestricted strategy. Given those results we can answer the question whether relevance filtering is useful in the Event-B context with yes. Interestingly the Multi filter which lets a number of filters “vote” on each hypothesis shows the best performance which is almost 10 percentage points better than unrestricted.

When judging the quality of a relevance filter, one can of course look at the percentage of manual proofs which in our case shows a clear winner: the Multi filter. Another criterion is the size of the relevant set in comparison to the manual proof rate. The rationale behind this is that if two filters \( f_1 \) and \( f_2 \) achieve the same percentage of manual proofs, but on average \( f_1 \) selects 30\% of the visible hypotheses while \( f_2 \) selects 40\%, then \( f_1 \) provides a more accurate approximation of the set of required hypotheses than \( f_2 \).
Filter | Percentage of manual proofs | Avg. selection percentage
--- | --- | ---
Unrestricted | 28.72% | 100%
Lasso | 27.74% | 98%
SubExpr(4; 0.3) | 24.23% | 32%
DCR(0.15) | 23.46% | 45%
SInE | 21.98% | 45%
RF(0.4; 1.2) | 20.08% | 36%
Multi | 18.75% | 33%

These experiments were run with the auto-tactic configuration described in Section 4.2 on our test set. ML was not used, but NewPP and PP with a timeout of 2000 ms each. The “Multi” filter is the majority based filter from Section 3.7 with the following sub filters: lasso, RF(0.4), SInE, DCR(0.3), SubExpr(3; 0.3).
The “average selection percentage” is the quotient of the average number of visible hypotheses and the average size of the relevant set output by the filter.

Table 4.3: Filter performance

Again the Multi filter looks pretty good here: it has the second smallest selection size and the lowest manual proof rate. Overall the differences in selection size are not that big with the exception of the lasso strategy which seems to select almost everything. This also explains the small difference between lasso and unrestricted.

We will now take a closer look at the effect relevance filtering has on the performance of a single prover. Figure 4.1 shows the effect of relevance filtering on the NewPP and the PP prover for different timeouts. We see that relevance filtering always has a positive effect. Especially for the PP prover the improvement is tremendous compared to the unrestricted strategy. The success rate curves generally become flatter when using a relevance filter and the point after which an increased timeout does not lead to a significant improvement anymore seems to be reached a little bit earlier. For the PP prover this point seems to be at 4000ms in unrestricted mode, but at 3000ms with RF(0.4) and even earlier with the SubExpr filter. NewPP shows similar effects but they are less evident. Generally we note that relevance filtering has the desirable effect of achieving a better success rate with lower timeouts.
Figure 4.1: Effects of relevance filtering on prover performance
### 4.5 Combinations of Relevance Filters

Inspired by the common practice in the Rodin platform to use more than one selection strategy, we experimented with combinations of relevance filters. Table 4.4 shows the performance of a combination of six filters that we found to work well. We notice that removing a single filter from the combination sometimes has surprising effects given the evaluation of the single filters. For example, the removal of the Lasso filter has by far the largest effect even though it showed the worst performance in the individual evaluation. This indicates that a filter has a specific “area of expertise” that is not visible when only looking at the individual overall performance.

We see that the “C1” combination performs significantly better than any single filter. Of course one can argue that the combinations have much more prover time available since it runs the provers six times with a 2000ms timeout instead of only once when using a single strategy. To refute this argument we used the most successful single filter with 12000ms prover timeout – six times more than usual, thus giving the single strategy the same amount of prover time as the “C1” strategy has available.

We can see that even with a longer timeout a single filter does not get anywhere near the performance of a combination of filters. We already know from Figure 4.1 that most proofs are found within the first two seconds of a prover’s runtime. We can now conclude that a better selection of the hypotheses works
much better than a longer prover timeout. The point is that all filters we have looked at are based on some heuristic assumption that works well on some proof obligations but not so well on others. Since it is very difficult or even impossible to know before attempting a proof which selection heuristic works best, we need to try them all. Figure 4.2 shows the filters’ performance on individual models. We see that none of the filters is consistently the best and it is easy to find pairs of models where a filter performs very well on one model and very poorly on the other in comparison to the other filters. For example the SInE filter is the best filter on the “MyCache” model but the worst on the “FileSys” model.

4.6 The Practical Optimum

When evaluating relevance filters, a natural question is: what is the best one can hope for? Phrased differently: If we had a perfect filter that always selects exactly the required hypotheses, how many proofs would still require help by the user?

Luckily we can get an approximate answer to this question. Since the models we work with already have proofs for all proof obligations, we can determine the hypotheses that were used in the existing proof and check whether the auto-tactic, without any filters, is able to prove the proof obligation if only the required hypotheses are selected.

In the following we will present a method which, given an existing proof of a goal \( g \), outputs a set \( R \) of hypotheses that contains exactly the hypotheses that were used in the given proof of \( g \). Of course there might still exist a set of hypotheses \( S \) with \( |S| < |R| \) that is sufficient to find a proof of \( g \), but then the proof tree would be different. Even though it is easy to construct an example to demonstrate this,
we have not seen this behavior in our test set.

Readers only interested in the results can skip right to the Results section on page 43.

4.6.1 Sources of Required Hypotheses

Each node in an Event-B proof tree is associated with a sequent and a rule that has been applied to it. The rules are generated by reasoners. A rule that is associated with a proof tree node provides a set of hypotheses that were necessary to apply it to the associated sequent. Thus all hypotheses that are necessary for a rule, that is used somewhere in the proof tree, are required for the proof as a whole.

Unfortunately, the PP prover and ML do not output an accurate set of required hypotheses. That means the set of required hypotheses output by PP or ML almost always contains hypotheses that are not necessary to prove the goal. This is of course not good enough for our purposes.

For this reason we implemented an algorithm that can compute a more accurate set of required hypotheses (see Algorithm 5). The algorithm constructs the set of required hypotheses $N$ by removing hypotheses from the sequent. If the prover cannot prove the sequent anymore after a hypothesis $f$ has been removed, $f$ is added to $N$. If the prover can still discharge the proof obligation without $f$, then $f$ is obviously not a necessary hypothesis.

It should be noted though that this is only an approximation since it is possible that the algorithm finds a local minimum.

4.6.2 Tracing Hypotheses

Another problem is that a hypothesis available at a leaf node in the proof tree might not be available at the root node because it has been modified or created by a proof rule somewhere between the root and the leaf node.

Figure 4.3 shows a very simple proof tree for the goal $a = b$. The boxes show the available hypotheses at each proof node. The edges are annotated with the rules that are applied and the hypotheses that are necessary for the application of the rule. The “Hyp” rule discharges the sequent since there is a hypothesis equal to the goal at that point. As one can see the whole proof requires two hypotheses: $a = b$ and $c = b$. The problem is now that $a = b$ is not a hypothesis at the root node. This is because $a = b$ is the result of $a = c$ being rewritten as $a = b$ by the “eqHyp” rule. So we conclude that at the root level the hypotheses $a = c$ and $c = b$ are required.

Fortunately it is possible to determine which root-level hypotheses were involved in the construction of a hypothesis further down in the tree by looking at the the
Algorithm 5 Hypothesis minimizer

\[ S := \{ \text{Input: Set of needed hypotheses as output by the prover} \} \]

\[ g : \{ \text{Input: Goal to prove} \} \]

\[ H := \emptyset : \{ \text{Variable: Current prover input} \} \]

\[ N := \emptyset : \{ \text{Output variable: Set of needed hypotheses} \} \]

\[ K := \emptyset : \{ \text{Output variable: Set of not needed hypotheses} \} \]

while \( S \neq \emptyset \) do

\[ h := \{ \text{an arbitrary hypothesis from } S \} \]

\[ S := S \setminus \{ h \} \]

\[ H := H \cup S \]

\{ \text{pr}(H, g) \) is true if the prover can prove } g \) from } H \}

if \( \text{pr}(H, g) \) then

\[ K := K \cup \{ h \} \]

else

\[ N := N \cup \{ h \} \]

end if

end while

“actions” associated with the rules at the proof tree nodes. Since the details are quite technical and specific to the Rodin platform, we omit them here.

4.6.3 Results & Summary

The methods described above approximate a perfect relevance filter that selects the minimum set of hypotheses required to prove a goal. With its help we can determine a lower bound for the percentage of manual proofs that is achievable with relevance filtering.

The only drawback of the method is that we select the union of required hypotheses from all sub-goals at the root of the proof tree. The relevance filters that we implemented, however, can make a different selection for each sub-goal. In fact, there are some cases where a relevance filter can automatically discharge a PO that is not recognized as potentially automatic by the method presented above.

For example in Figure 4.4, node B requires the two hypotheses \( h_1 \) and \( h_2 \), and node C requires the two hypotheses \( h_1 \) and \( h_3 \). If now the union of those two sets of required hypotheses is selected at the root node A, then it might happen that the prover cannot discharge the sequent at node B because it is confused by \( h_3 \). Unfortunately, this problem is difficult to solve because the alternative, which is to remove manual proof steps from the proof tree bottom up until only automatic proof steps are left, is infeasible since it is currently not possible to decide whether a specific reasoner has been applied manually or automatically.
Figure 4.3: Sample proof tree

Figure 4.4: Proof tree
Table 4.5: Practical minimum for the percentage of manual proofs

<table>
<thead>
<tr>
<th>Overall</th>
<th>Cache</th>
<th>IEEE</th>
<th>train</th>
<th>CtsCtrl</th>
<th>MyCache</th>
<th>FileSys</th>
<th>Zer_ess</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.18%</td>
<td>5.11%</td>
<td>8.96%</td>
<td>6.02%</td>
<td>6.75%</td>
<td>3.21%</td>
<td>8.18%</td>
<td>21.82%</td>
</tr>
</tbody>
</table>

Even though this problem does occur, we do not think that it influences the result much. The percentage of proof obligations that could be discharged with a relevance filter but not with the method presented in this chapter is less than 0.2%. These POs have already been accounted for in Table 4.5. Keeping those restrictions in mind, the practical minimal percentage of manual proofs using a perfect relevance filter for the provers NewPP and PP together with ML (unrestricted) seems to be around 6% for our test set.

4.7 A Better Strategy for Event-B

In the Event-B context we are obviously interested in the improvement we can achieve using relevance filtering compared to the status quo. The numbers presented in Section 4.5 are not representative for a typical setup of the Rodin platform because ML was not used. We did not use ML for those experiments because ML is more resilient to irrelevant hypotheses than the other provers and therefore relevance filters only have little effect on its performance, at least in our test setup. See Section 4.8.2 for more information about this. In the following ML is always used in unrestricted mode without any relevance filtering.

We will now present a filtering combination that gets within one percentage of the practical optimum that we determined in Section 4.6. The configuration consists of the filter combination presented in Section 4.5 with the addition of the restricted strategy with a prover timeout of 500ms and the use of ML. We will refer to this configuration as “C1e” (see Figure 4.5).

Now why is this particular configuration the best? Due to the extremely large number of possible combinations of filter settings and filter orders, this question is impossible to answer with complete certainty. One thing that we can verify though, is the filter order. The order of the filters is important because every prover invocation takes time. Obviously we want that a proof obligation is discharged with as few prover invocations as possible. Table 4.6 shows which filter is responsible for
Configuration “C1e”:

- Use ML (unrestricted) – 500ms timeout, force 0 and 1
- Use filters in this order:
  1. Restricted – 500 ms prover timeout
  2. RF(0.4; 1.2) – 2000 ms prover timeout
  3. SubExpr(3; 0.3) – 2000 ms prover timeout
  4. RF(0.8; 1.2) – 2000 ms prover timeout
  5. SInE – 2000 ms prover timeout
  6. Lasso – 2000 ms prover timeout
  7. DCR(0.3) – 2000 ms prover timeout

Figure 4.5: Configuration “C1e”

how many successful proofs when they are tried in the order given in configuration “C1e”. The restricted strategy is not considered in the table due to the different prover timeout. We can see that the majority of proofs are found with the help of the RF(0.4) filter and the percentage of proofs that require more than one attempt is (more or less) steadily declining with each additional filter that is tried.

**Performance** Since we use seven filters in the “C1e” configuration, it can happen that the provers are called seven times for the seven filters without success. This can obviously increase the proving time. To limit the effect of this problem we

<table>
<thead>
<tr>
<th># Filter</th>
<th># proofs</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF(0.4)</td>
<td>113</td>
<td>61%</td>
</tr>
<tr>
<td>SubExpr(3; 0.3)</td>
<td>28</td>
<td>15%</td>
</tr>
<tr>
<td>RF(0.8)</td>
<td>14</td>
<td>8%</td>
</tr>
<tr>
<td>SInE</td>
<td>14</td>
<td>8%</td>
</tr>
<tr>
<td>Lasso</td>
<td>7</td>
<td>4%</td>
</tr>
<tr>
<td>DCR(0.3)</td>
<td>8</td>
<td>4%</td>
</tr>
</tbody>
</table>

Table 4.6: Number of proof attempts before a proof is successful with the “C1e” configuration
implemented an option to limit the time spent on a proof attempt for all filters together. Effectively this means that filters that appear towards the end are not called anymore when time runs out. See Tables 4.7 for the effectiveness of the “C1e” strategy for various time limits. Table 4.8 shows the same for a slightly different auto-tactic configuration concerning the “eqHyp” tactic. See Section 4.8.3 for details about this problem.

We see that given unlimited time, the manual proof rate drops to within one percentage point of the 6.18% that are possible using an optimal relevance filter. We think this is a very good result, especially considering the relatively simple nature of the filtering methods we used.

Comparison
Is it worthwhile to use relevance filtering in Event-B? The answer to this question is a clear yes as we will see shortly. First we need to ask ourselves what do we compare the “C1e” configuration to? We see the following options:

1. Only the unrestricted strategy with a timeout of 12500ms to match the time budget available to the provers in “C1e”. This emphasizes on the filter vs. no filter question.

2. Unrestricted with 12000 ms timeout plus the restricted strategy (with the default 500ms timeout). This takes into account that the restricted strategy is not really a filter but a fixed selection created by the Rodin platform.

3. Unrestricted and lasso with the default 2000ms timeout and restricted with the default 500ms timeout. This represents the optimum that was possible with Rodin before the implementation of the relevance filters presented
<table>
<thead>
<tr>
<th>Time limit</th>
<th>Percentage of manual proofs</th>
<th>Proving time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 s</td>
<td>9.13%</td>
<td>46 min</td>
</tr>
<tr>
<td>10 s</td>
<td>8.36%</td>
<td>61 min</td>
</tr>
<tr>
<td>15 s</td>
<td>8.08%</td>
<td>65 min</td>
</tr>
<tr>
<td>20 s</td>
<td>7.79%</td>
<td>76 min</td>
</tr>
<tr>
<td>no limit</td>
<td>7.79%</td>
<td>86 min</td>
</tr>
</tbody>
</table>

For these experiments the restricted strategy was placed before the “eqHyp” tactic and the rest of the filters was placed after the “eqHyp” tactic in the auto-tactic configuration. The “time limit” column refers to the time limit per proof obligation. “Proving time” is the time it takes to prove the test set.

Table 4.8: Manual proof rate and proving time with different time limits for the “C1e” configuration. Alternative eqHyp position.

in this thesis. Comparing to this setup disregards the fact that the “C1e” configuration potentially calls the provers three times more often.

4. Like the above but with a prover timeout of 6000ms for unrestricted and lasso to offset the longer potential prover time available in the “C1e” configuration.

Figure 4.6 shows the trade offs between success rate and proving time for the “C1e” configuration with different time limits as well as for the various comparison options listed above. We see that the “C1e” configuration is always faster (using a time limit) and more successful than any of the comparison options. How much lower the manual proof rate is depends on the time limit that is used for the “C1e” configuration.

4.7.1 Re-evaluation & Summary

In order to verify our results about the effectiveness of our filtering strategies and rebut critique about over-fitting, we re-evaluated the “C1e” strategy with a new set of Event-B models. Our new test set consists of six model with a total of 4276 proof obligations. This is more than twice the number of proof obligations in our original test set.
The numbers in the square brackets indicate the prover timeout for NewPP and PP.

Cu: unrestricted[12500ms]; ML
CuR: restricted[500ms], unres[12s]; ML.
C0: restricted[500ms], lasso[2s], unrestricted[2s]; ML.
C0a: restricted[500ms], lasso[6s], unrestricted[6s]; ML.
C1e: See Figure 4.5
C1e, alt eqHyp: See Table 4.8
Optimum: Practical optimum (see Section 4.6).

Figure 4.6: Proving time vs. manual proof rate
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>POs</th>
<th>Mch/Ctx</th>
<th>Arithmetic</th>
<th>Avg. problem size</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Birthday</td>
<td>A model of a birthday book</td>
<td>61</td>
<td>3/1</td>
<td>Yes</td>
<td>12</td>
<td>–</td>
</tr>
<tr>
<td>BepiColombo</td>
<td>A model of BepiColombo SIXS (Solar Intensity X-ray Spectrometer)</td>
<td>1000</td>
<td>9/10</td>
<td>No</td>
<td>444</td>
<td>[13]</td>
</tr>
<tr>
<td>DepSetSpec</td>
<td>A model of an Attitude and Orbit Control System</td>
<td>1858</td>
<td>14/2</td>
<td>Yes</td>
<td>207</td>
<td>[30]</td>
</tr>
<tr>
<td>LSR</td>
<td>A model of a distributed topology discovery algorithm</td>
<td>393</td>
<td>7/1</td>
<td>Yes</td>
<td>47</td>
<td>[11]</td>
</tr>
<tr>
<td>Elevator</td>
<td>A model of an elevator control system</td>
<td>242</td>
<td>12/4</td>
<td>Yes</td>
<td>25</td>
<td>–</td>
</tr>
<tr>
<td>FlashFileSystem</td>
<td>A model of a flash-based filestore</td>
<td>722</td>
<td>14/8</td>
<td>Yes</td>
<td>76</td>
<td>[5]</td>
</tr>
</tbody>
</table>

Avg. problem size is the average number of visible hypotheses at the root of a proof tree.

Table 4.9: Models used for re-evaluation
<table>
<thead>
<tr>
<th>Strategy</th>
<th>Percentage of manual proofs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>21.02%</td>
</tr>
<tr>
<td>C1e</td>
<td>7.97%</td>
</tr>
<tr>
<td>Opt.</td>
<td>7.18%</td>
</tr>
</tbody>
</table>

Both configurations use the NewPP and PP strategies with the following filters:
“C0”: Restricted [500ms], lasso[2s] and unrestricted[2s]; ML
“C1e”: See Figure 4.5
“Opt.”: Practical optimum (see Section 4.6).

Table 4.10: Performance with new models

As we can see in Table 4.10 the performance of our filtering strategy again lies within less than one percentage point of what is possible with an optimal filter. Surprisingly though, the improvement achieved with relevance filtering is much greater than with the old test set. The improvement is now more than 13 percentage point compared to only slightly more than 4 percentage points with the old test set. The reason for this might be that the new test set contains two very large models with 1000 POs or more and average problem sizes in the hundreds. Unfortunately we could not do accurate time measurements due to a memory problem in the NewPP prover which forced us to implement measures that affect the overall proving time. Keeping this in mind, the proving time for the new test set was around 22 hours for both configurations.

Given the results of the re-evaluation we can conclude that our results are solid. Our “C1e” filter combination achieves a result close to optimal for both test sets.
Table 4.11: Effects of white listing

<table>
<thead>
<tr>
<th>Filter</th>
<th>Percentage of manual proofs</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF(0.2; 1.2)</td>
<td>21.28%</td>
</tr>
<tr>
<td>RF(0.4; 1.2)</td>
<td>20.08%</td>
</tr>
<tr>
<td>RF(0.2; 1.2), no WL</td>
<td>21.00%</td>
</tr>
<tr>
<td>RF(0.4; 1.2), no WL</td>
<td>33.01%</td>
</tr>
<tr>
<td>SinE</td>
<td>21.98%</td>
</tr>
<tr>
<td>SinE, no WL</td>
<td>28.44%</td>
</tr>
</tbody>
</table>

WL = Whitelisting.
These experiments were performed using the NewPP and PP provers with a 2000ms timeout each. The filter listed in the “Filter” column was the only filter used. ML was not used.

4.8 Other Discoveries

In this section we present some smaller results that support some design decisions we made or that are interesting for the Event-B community.

4.8.1 Effects of White Listing

In Chapter 3 we presented a dynamic, heuristic whitelisting method that we added to all our filters as a post-processing step. We will now have a closer look at the effects of white-listing. Table 4.11 shows the performance of some filters with and without our dynamic white listing method. We see that the effect is huge: Almost 13 percentage point for the RF(0.4; 1.2) filter method. Our whitelisting heuristic mainly adds hypotheses that give information about the types of variables which seem to be quite important for proofs in Event-B. Adding those hypotheses as a post processing step seems to be better than using a less strict filter to get them (compare RF(0.4) to RF(0.2)) without whitelisting in Table 4.11.

4.8.2 Is ML Resilient Against Irrelevant Hypotheses?

The fact that by default the Rodin platform does not allow the user to invoke the ML prover in restricted mode indicates that ML might be resilient to irrelevant hypotheses. In this section we investigate if this is true. To do this, we use the algorithm from Section 4.6 to find the set of required hypotheses for a proof and then run ML restricted to only this set. The results can be seen in Table 4.12. The
<table>
<thead>
<tr>
<th>Prover mode</th>
<th>Percentage of manual proofs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML unrestricted</td>
<td>26.97%</td>
</tr>
<tr>
<td>ML restricted to req. hyps</td>
<td>24.30%</td>
</tr>
<tr>
<td>ML with 1000ms timeout &amp; all force</td>
<td>24.86%</td>
</tr>
<tr>
<td>NewPP restricted to req. hyps</td>
<td>21.07%</td>
</tr>
<tr>
<td>NewPP unrestricted</td>
<td>33.78%</td>
</tr>
</tbody>
</table>

Table 4.12: ML prover with different selection methods.

difference to ML in unrestricted mode is about 2.6\% which is not much compared to NewPP where the difference is more than 12\%. Another hint that ML is resilient against irrelevant hypotheses is the fact that increasing the timeout from 500ms to 1000ms and setting the force setting to “all” brings the manual proof rate to 24.86\% again.

Another interesting point is how prover performance varies on different problem sizes. This is shown in Figure 4.7. Problem size is defined here as the number of visible hypotheses. The first chart shows the success rate of ML for different problem sizes. We compared ML in its default configuration (500ms timeout, unrestricted), ML (unrestricted) with a timeout of 2000ms and ML (500ms timeout) restricted to only the necessary hypotheses as determined by the method described in Section 4.6. Assuming there are 80 visible hypotheses for some problem and 10 of them are necessary, then ML will be given all 80 hypotheses in unrestricted mode, but only the 10 necessary ones in restricted mode. Both data points will be added to the 80-89 bin, though.

We see that ML’s performance is very consistent up to a problem size of about 80. After that the success rate for all three settings drops significantly. For the default mode it drops to almost zero while the version restricted to only the necessary hypotheses shows by far the best performance.

It seems that up to a problem size of 50 the selection of hypotheses is not important since the performance is almost identical for all three settings. For problem sizes between 50 and 80 the short timeout in the default configuration seems to be a problem since it loses ground to the other two configurations. For problems of size over 80 relevance filtering also seems to become relevant for ML since even the configuration with a 2s timeout is significantly worse than the one restricted to only the required hypotheses for those problems.

When interpreting the results one should consider that the statistics for problems of size >80 must be taken with a grain of salt since there are much less problems of size >80 than <80 as can be seen in the third chart, which shows the average number of examined proof obligations of each size. We show the average size because the number of examined proof obligations is not exactly the same for the
Figure 4.7: Prover success rate for different problem sizes
different prover settings since the auto-tactic tends to increase the problem size when the prover is not successful.

It must also be noted that the position of the ML prover in the auto-tactic had to be changed for this experiment since a comparison to NewPP would not have been possible otherwise. It was now used at the same position as NewPP (item 17 in Section 4.2).

The second chart shows the same as the first chart, but for the NewPP prover (with the 2000ms default timeout). It is immediately visible that the difference between the case where we restrict NewPP to only the necessary hypotheses and the unrestricted case is much larger for NewPP than for ML. Therefore we can conclude that ML is significantly more resilient to irrelevant hypotheses than NewPP and up to a problem size of 80 almost indifferent to the selection of hypotheses.

### 4.8.3 Position of the “Use equals Hypotheses” Tactic

Rodin contains a tactic called “Use equals Hypotheses” or “eqHyp” for short. The “eqHyp” tactic replaces variables by their definition. For example if there are the hypotheses $h_1 := f(a) \land g(a) \Rightarrow k(a)$ and $h_2 := a = 1$, the “eqHyp” tactic will rewrite $h_1$ as $f(1) \land g(1) \Rightarrow k(1)$. This tactic can certainly have positive effects on the performance of the auto-tactic, but it also decreases the effectiveness of our relevance filters. We think the reason for this is that “eqHyp” does not only replace variables with literals as in the example above but also larger terms like $a = A \triangleleft k(x)$. Doing this introduces more symbols into a formula which makes it more likely for a formula to be considered relevant by a relevance filter. This leads to more irrelevant hypotheses being selected by a filter and thus decreases prover performance.

Experiments (see Table 4.13) have shown that if the “eqHyp” tactic is placed after the provers in the auto-tactic configuration, the percentage of manual proofs is about one percentage point lower than if the “eqHyp” tactic is placed before the provers. Unfortunately, placing the “eqHyp” tactic after the provers also increases the time required to prove the test set by about 50%. To understand the reason for that we need to recall that Rodin tries to apply the tactics contained in the auto-tactic configuration in the given order. If a tactic is successful, it starts over with the first tactic. Thus, if the “eqHyp” tactic appears before the provers in the

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Percentage of manual proofs</th>
<th>Proving time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1e after eqHyp</td>
<td>8.01%</td>
<td>72 min</td>
</tr>
<tr>
<td>C1e before eqHyp</td>
<td>7.02%</td>
<td>105 min</td>
</tr>
</tbody>
</table>

Table 4.13: Effect of different positions of the “eqHyp” tactic
auto-tactic configuration, it is applied as many times as possible and only then the provers are invoked. On the other hand, if the “eqHyp” tactic appears after the provers, the provers are invoked before every invocation of the “eqHyp” tactic. Since a prover invocation is relatively time consuming, the overall proving time is longer in that case.
5 Tools

During the work on this thesis several tools were developed to facilitate the evaluation and the integration of the filtering strategies into the Rodin platform. Below is a list of the most notable ones.

5.1 The Meta Prover

The Meta Prover acts as an umbrella for the various filtering strategies and theorem provers. It appears as a single entry in the auto tactic preferences and can be configured using the configuration dialog we implemented. See Section 5.2. The main features of the Meta Prover are listed below.

**Timelimit**  The meta prover allows the user to set a limit on the total time spent on a proof obligation. In addition it allows the user to configure the prover timeouts which was not possible before.

**Filter Ordering**  Since all out attempts at adaptively ordering the relevance filters have been unsuccessful the meta prover simply applies the filters in the user specified order. However, there is one optimization that uses the prover output to decide if a set of hypotheses is sufficient to prove a proof obligation. Namely, if the prover outputs “Fail”, which means that the goal cannot be derived from the given hypotheses, we know that a crucial hypothesis is missing – therefore selections which are a subset of the already tried one do not have to be tried anymore.

5.2 Filter Configuration Dialog

In order to make the relevance filters available and configurable to the end user we implemented a preferences dialog that allows the user to configure the relevance filters as well as the provers. See Figure 5.1 to see what it looks like.
5.3 Proof Shortener

When doing manual proofs it often happens that the user performs more manual steps than necessary. Initially we wanted to evaluate how much relevance filtering can shorten proofs. It turned out that this test was too time consuming and also not objective because it depends on the size the proof had before which in turn depends on the human who proved it in the first place. Nevertheless we think this feature might be valuable for some people who want to shorten their proofs for better readability.

In order to make a given proof as short as possible, the proof shortener tries to apply the auto-tactic (or only the provers with relevance filtering) as close as possible to the proof tree root. This often allows it to get rid of manual proof steps and thus leads to a shorter proof.

5.4 Hypothesis Minimizer

To better understand how the automated theorem provers work it helps a lot to know which hypotheses are needed for a proof. To provide this information we developed a tool that uses the algorithm described in Section 4.6 to determine
the set of required hypotheses and then select exactly the needed hypotheses immediately before the prover application in the proof tree. That way the required hypotheses for a proof obligation are easily accessible in the proof tree.
6 Conclusion

All relevance filtering strategies we implemented are based on relatively simple assumptions which may or may not be correct for a specific problem. When we implemented our own idea for a relevance filter it became apparent that it is not possible for one particular heuristic to always find a good approximation of the set of required hypotheses. Any attempt to make the filter work for a particular case resulted in over-fitting the filter to a specific problem where we could not justify our design decisions any more. It seems there is no middle ground between simple heuristics that produce a very rough guess of the set of required hypotheses, which might even lack a required hypothesis, and actually proving the proof obligation to get a very accurate set of required hypotheses. Initially we thought predicate abstraction might be that middle ground. Unfortunately, formulas in the Event-B context make extensive use of quantification which makes predicate abstraction difficult. There have been attempts to use predicate abstraction with formulas that contain quantifiers [8, 18], but they go all the way to build a theorem prover. Also their method is not complete and it seems that in their setting a significant portion of hypotheses do not contain quantifiers which is not true in the Event-B context.

This brings us to the question how general our results are. With the exception of the sub-expression filter all filtering techniques discussed in this thesis work on arbitrary theorem proving problems since they view a logical formula simply as a set of symbols. Since the sub-expression filter tries to exploit structural similarities, it might not work well for problems formulated in CNF, which has a fixed nesting structure for logical connectives.

The relative performance of the relevance filters depends very much on the problem set and can therefore not be generalized. Even within our test set, there is significant variation between different models (see Figure 4.2). For our main result, that combining different filters with short prover timeouts works better than increasing the prover timeout of an individual filter, we don’t see any reason why it should not apply to other settings.

For the Event-B context we showed on two independent test sets that a combination of rather simple syntactic filters can achieve a manual proof rate that lies within one percentage point of what is practically possible with relevance filtering. In order to further decrease the manual proof rate the provers themselves would have to be improved or tactics would have to be added which perform proof steps,
that currently have to be done by hand, automatically. Regarding these results we believe that the limits of syntactic relevance filtering in Event-B have been reached. Even a perfect relevance filter could only achieve a minimal additional decrease of the manual proof rate.
Bibliography


