Implicational Rewriting Tactics in HOL

Vincent Aravantinos[1] and Sofiène Tahar[2]

[1] Software & Systems Engineering, Fortiss GmbH,
Guerickestraße 25, 80805, Munich, Germany
vincent.aravantinos@fortiss.org
http://www.fortiss.org/en
[2] Electrical and Computer Engineering Dept., Concordia University,
1455 De Maisonneuve Blvd. W., Montreal, Canada
tahar@ece.concordia.ca
http://hvg.ece.concordia.ca

Abstract. Reducing the distance between informal and formal proofs in interactive theorem proving is a long-standing matter. An approach to this general topic is to increase automation in theorem provers: indeed, automation turns many small formal steps into one big step. In spite of the usual automation methods, there are still many situations where the user has to provide some information manually, whereas this information could be derived from the context. In this paper, we characterize some very common use cases where such situations happen, and identify some general patterns behind them. We then provide solutions to deal with these situations automatically, which we implemented as HOL Light and HOL4 tactics. We find these tactics to be extremely useful in practice, both for their automation and for the feedback they provide to the user.

1 Introduction

Interactive theorem proving has well-known benefits: it allows to build a formal proof with the help of a computer ensuring the proof is correct. It does not have the restrictions of automated theorem proving since it can appeal to the user's creativity through interaction. But this interaction is also the shortcoming of interactive theorem proving: the user is forced to make explicit some formal steps that are obvious to a human. Thus, automation of these steps, when possible, relieves the user from many tedious manipulations. These complex manipulations are one of the essential reasons why interactive theorem proving is not so popular in practical applications. Thus automation is a key ingredient to bring formal reasoning to a wider audience by making it closer to intuitive reasoning.

In interactive theorem proving, the main tool for automation of reasoning is *decision or semi-decision procedures* [18,25]: e.g., for propositional reasoning [10], linear arithmetic reasoning [8], reasoning modulo various theories [5], or even first-order [12] or higher-order reasoning [16]. These have been an independent subject of research for many years with several of the corresponding progresses being transfered to interactive theorem proving, either by direct implementation or by call to external tools [24]. But decision procedures are useful only to conclude goals: the user must resort to interaction with the theorem prover if the goal is too complex to be proven by a decision procedure (which is the case of most goals). (S)he still has access however to another kind of automation: rewriting.

G. Klein and R. Gamboa (Eds.): ITP 2014, LNAI 8558, pp. 45–60, 2014.

Rewriting enables automation of equality reasoning: given a theorem stating an equality $t = u$ and a term v which contains a subterm matching t, rewriting replaces this subterm by the corresponding instantiation of u (we refer to [3,21] for details). An extremely useful generalization is *conditional rewriting* [9] (often called "simplification" in the HOL Light [14] and HOL4 [29] communities): given a theorem whose statement has the form $p \Rightarrow l = r$, conditional rewriting replaces, in a term t, any subterm matching l, *if the condition p can be proven automatically for the corresponding instantiation*. (Conditional) rewriting may not prove goals as complex as the ones proven by some decision procedure, but it can be used at any step of a proof, even if it does not terminate this proof: it simply allows to *make progress* in the proof, which is an extremely useful feature in an interactive context. However, (conditional) rewriting still requires regularly some user explicit input could be automated as we show now.

Example 1. Consider the following goal (for clarity, we use mathematical notations instead of HOL Light ASCII text to represent mathematical expressions): $\forall x, y, z.\ \mathrm{prime}\ y \wedge xy > z \wedge x - y = 5 \wedge x^2 - y^2 = z^2 \wedge 0 < z \wedge 0 < y \Rightarrow \frac{x}{x} y = y$ where $\mathrm{prime}\ y$ indicates that y is a prime number.

Assume that the immediate objective of the user is to rewrite $\frac{x}{x}$ into 1. To do so, (s)he calls the rewriting tactic with the theorem $\vdash \forall x.\ x \neq 0 \Rightarrow \frac{x}{x} = 1$. This of course does not work since this theorem is not purely equational: one must here use *conditional* rewriting in order to get rid of the condition $x \neq 0$. However which theorems should be provided here in order to prove $x \neq 0$ automatically? In this goal, this follows from the fact that $x.y > z$ and $0 < z$: therefore $x.y > 0$; thus $x.y \neq 0$, and hence $x \neq 0$. This proof is not mathematically difficult, however it requires a lot of thought from the user before being able to come up with the tactic call which will accomplish the intended action, since (s)he basically needs to mentally build a formal proof. Once this is done, one can apply conditional rewriting with the following theorems: $\vdash \forall x.\ x \neq 0 \Rightarrow \frac{x}{x} = 1$, $\vdash \forall x, y.\ x > y \Leftrightarrow y < x$, $\vdash \forall x, y.\ x < y \wedge y < z \Rightarrow x < z$, $\vdash \forall x.\ 0 < x \Rightarrow 0 \neq x$, $\vdash \forall x, y.\ x.y < 0 \wedge 0 < y \Rightarrow 0 < x$.

The whole process is extremely tedious. Even more embarrassing: this process is about building mentally a formal proof whereas helping such a task is precisely what an interactive theorem prover is made for! Therefore, there can of course be mistakes in this proof, or omission of some intermediate theorems when calling the tactic. In addition, this is a situation where the user is forced to interrupt the flow of his/her proof in order to adapt this proof to the tool at use: that is precisely the sort of situation leading a user to conclude that interactive theorem proving is counter-intuitive, tedious to use, and therefore to maybe give up on using it. Finally, notice that the simplifier might also simply not be able to deal with the sort of reasoning involved in the proof. In all these cases, *nothing* happens: the tactic does not apply any change to the goal and the user is left with no clue which of the above flaws is the reason for the lack of progress.

To avoid this, a simpler and more frequently used approach is to simply assert $x \neq 0$, and prove it as an independent subgoal. This allows to get rid of all the above flaws: proving this condition is done under the control of the theorem prover, which helps the user with the goal and tactic mechanism, thus providing some useful feedback while avoiding any mistake in the formal proof. In addition, one can use complex reasoning that is out of reach for the conditional rewriter of HOL Light or HOL4. However, this approach forces the user to write *manually* the subgoal $x \neq 0$. Manually writing explicit information in a script is extremely fragile with respect to proof change: if ever x is renamed in the original goal, then the proof script has to be updated; similarly if an earlier modification changes the situation into the

exact same one, but with a non-trivial expression instead of x; in other cases, some change of prior definitions, or the removal of some assumptions might all as well lead to necessary updates of the subgoal. Finally, all these updates are even more tedious when the subgoal is big (also entailing more potential typing errors), which happens frequently in real-life situations.[1]

In this paper we target precisely this sort of problem. We characterize frequent situations presenting similar problems and provide solutions to them. This includes the problem mentioned in the above example and others, more or less frequent where the user also has to explicitly write some information which could be derived automatically by the theorem prover. Note that the forms of reasoning which we address in this paper are actually simple: they are much simpler than any decision procedures. But it is precisely because they are simple, that it is necessary and useful to automatize them.

The rest of the paper is organized as follows: Section 2 presents our solution to the problem of Example 1. Section 3 describes the underlying algorithm. Section 4 proposes several refinements to our solution. Section 5 presents solutions to other situations presenting the same sort of proof-engineering defects. Finally Section 6 discusses the related work and Section 7 concludes the paper.

This work is entirely implemented in HOL Light and HOL4. The HOL Light version has been integrated in the official distribution of HOL Light and the sources for HOL4 can be publicly found at [1]. Both implementations come with a manual providing technical details to use the tactics.

2 Implicational Rewriting

This section deals precisely with the situation presented in Example 1. Terms and substitutions are defined as usual, with $t\sigma$ denoting the application of the substitution σ to the term t. For terms t, u, v, the notation $t[u/v]$ denotes the term obtained from t by replacing all occurrences of v by u. A *formula* is any term of type boolean. For a formula of the form $\phi \wedge \psi$ (resp. $\neg\phi$, $\forall x.\phi$) we say that ϕ and ψ (resp. ϕ, resp. ϕ) are *direct subformulas* of the formula, and similarly for the other connectives and quantifier. The *subformulas* of a formula are defined by transitive closure of the relation "is a direct subformula". An *atomic subformula* is a subformula whose head symbol is not a logical connective or quantifier. Given a formula ϕ, a subformula ψ occurs *positively* (resp. *negatively*) in ϕ if it occurs in the scope of an even (resp. odd) number of negations or implication premisses[2]. The fact of occurring positively or negatively is called the *polarity* of the subformula.

Consider again the Example 1. In this example, implicational rewriting consists in replacing in the conclusion the atom $x \neq 0 \wedge \frac{x}{x}y = y$ (and only this atom, not the top formula) by $1.y = y$, i.e., $\frac{x}{x}$ is replaced by 1, and the conjunction with $x \neq 0$ is added to the atom. In case this atom was occurring negatively in the goal (e.g., the same goal but with $\frac{x}{x}y \neq y$) then $\frac{x}{x}y \neq y$ would have been replaced by $\neg(x \neq 0 \Rightarrow \frac{x}{x}y = y)$. Formally, this is generalized as follows:

[1] We do not claim that *every* explicitly-written subgoal is a bad practice w.r.t. proof engineering: many subgoals provide important high-level information which is out of reach for automation. However, the problem here is that *this subgoal could be automatically generated*. So the user should not be left with the burden of writing it.

[2] For simplicity, we do not consider formulas with equivalences, even though it is easily handled in practice, e.g., by rewriting $\phi \Leftrightarrow \psi$ into $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$.

Definition 1 (Implicational Rewriting). *Consider a goal with conclusion c. Then, we call* implicational rewriting *by th any tactic replacing one or more atomic subformula A of c by:*

$$p\sigma_1 \wedge \cdots \wedge p\sigma_k \triangleright A[r\sigma_1/l\sigma_1]\ldots[r\sigma_k/l\sigma_k] \quad (1)$$

where $\sigma_1, \ldots, \sigma_k$ are the matching substitutions of some subterms of A matching l and where $\triangleright = \wedge$ (resp. $\Rightarrow$) if A occurs positively (resp. negatively) in c.

Note that the definition leaves a lot of freedom about the strategy to use for the rewrite: not all subterms need to be rewritten, and we do not specify which subterms should be rewritten in priority. In the following, $\mathcal{IR}_{th}(c)$ denotes some function implementing the specification of Definition 1.

The first property of $\mathcal{IR}$ is that *it replaces indeed some term matching l by the corresponding instantiation of r.* But the tactic would be useless if it was not sound:

Theorem 1 (Soundness). *For every goal of conclusion c and every theorem th, it holds that $\mathcal{IR}_{th}(c) \Rightarrow c$.*

Contrarily to (conditional) rewriting, the resulting goal only entails the initial goal, but is not equivalent to it: this can be seen both as a weakness (one needs to backtrack if the result becomes not provable anymore) or as a strength (many more possible inferences are accessible for reasoning). Note that the case distinction about the polarity in Definition 1 is capital for this theorem to hold.

An essential property of implicational rewriting, as opposed to conditional rewriting, is that it provides *feedback* to the user about the condition that is required to be proven: where the user is left with a simply unchanged goal when conditional rewriting cannot prove the side condition, implicational rewriting provides instead the precise condition instantiation which has to be proven. In addition, since this condition now appears in the goal, the the theorem prover can be used to prove it, exactly as if the condition had been stated explicitly as a subgoal by the user.

Note that there exists an easier solution to the problem presented in Example 1: given a goal having a subterm matching l, replace $l\sigma$ by $r\sigma$ and introduce automatically a new *subgoal* stating $p\sigma$. This is the approach of [30] in HOL Light, or of the `force` function of ACL2. It is also very similar to [15] in HOL4: the only difference with the latter is that $p\sigma$ is added as a conjunction *on top of the overall formula* (and not at the level of the atom as implicational rewriting does) instead of being stated as a separate subgoal. Similar tactics are also available in Coq and Isabelle. This approach will be called *dependent rewriting* in the following, according to [15]. Table 1 sums up the different approaches (with only one rewriting for presentational reasons), where $g \rightsquigarrow g'$ denotes a tactic turning a goal g into g', an expression $\phi[t]$ in g means that t occurs in ϕ, then the expression $\phi[t']$ in g' denotes ϕ in which this occurrence is replaced by t'.

So the major difference between implicational and dependent rewriting is the fact that the latter applies *deeply*. We argue now that this is not a cosmetic feature but actually has a high impact on the *compositionality* of the tactic.

Example 2. Consider a goal $g : \forall x, y.\ P\ x\ y \Rightarrow \frac{x}{x} * y = y$, where $P\ x\ y$ is a big expression entailing in particular that $x \neq 0$. With implicational rewriting, using the theorem $\vdash \forall x.\ x \neq 0 \Rightarrow \frac{x}{x} = 1$, we obtain immediately the goal: $\forall x, y.\ P\ x\ y \Rightarrow x \neq 0 \wedge 1 * y = y$. Instead, with *dependent* rewriting, the tactic will try to replace the goal by $g' : x \neq 0 \wedge \forall x, y.\ P\ x\ y \Rightarrow 1 * y = y$. But this does not work since $x \neq 0$ is not in the scope of $\forall x$. Therefore $g' \not\Rightarrow g$ and the tactic application is not valid. Consequently, with dependent rewriting, one must first

Table 1. Definitions of the different sorts of rewriting

	Definition	Condition
Rewriting	$c[l\sigma] \rightsquigarrow c[r\sigma]$	if $\vdash l = r$
Conditional rewriting	$c[l\sigma] \rightsquigarrow c[r\sigma]$	if $\vdash p \Rightarrow l = r$ and $\vdash p$
Dependent rewriting	$c[l\sigma] \rightsquigarrow p\sigma \wedge c[r\sigma]$	if $\vdash p \Rightarrow l = r$
Implicational rewriting	$c[A[l\sigma]] \rightsquigarrow c[p\sigma \wedge / \Rightarrow A[r\sigma]]$	if A occurs pos./neg. in c

remove the quantifiers using the adequate tactics (`GEN_TAC` in HOL4 and HOL Light), and then only can apply dependent rewriting. We then obtain the goal $x \neq 0 \wedge (P\ x\ y \Rightarrow 1 * y = y)$. However, this is still not satisfying because the new goal is not provable: indeed $x \neq 0$ derives from $P\ x\ y$, but since $x \neq 0$ is not "in the scope" of $P\ x\ y$, it cannot be proven. Therefore, even though the tactic is valid, it is of no help for the proof. Consequently, one must first discharge the hypothesis $P\ x\ y$ in the assumptions before applying the dependent rewrite, yielding finally the goal $x \neq 0 \wedge 1.y = y$.

As the example demonstrates, a lot of book-keeping manipulations are necessary with dependent rewriting but not with implicational rewriting. But it gets even worse when one starts to try proving $x \neq 0$: if P is complex, it will also itself probably require some rewrites, however this is not possible in a simple way since $P\ x\ y$ is now in the assumptions. So a first option is to put the assumption back in the goal (tedious when goals have many assumptions), which actually amounts to manually doing what implicational rewriting does automatically.

A second option is to rethink the flow of the proof and give up on using dependent rewriting. This is what happens most commonly in practice: one will try, from the initial goal, to find a proof of $x \neq 0$ from $P\ x\ y$, apply the corresponding tactics, and finally use conditional rewriting. Note in addition that the proof of $x \neq 0$ must be done in forward reasoning since $x \neq 0$ does not appear in the goal: this makes the process even harder since interactive provers do not emphasize this type of reasoning. This is unless the user decides to set manually the subgoal $x \neq 0$ with the flaws already mentioned in Example 1. As we can see, all the intended benefits of using dependent rewriting are lost whatever is the chosen option and we get back precisely to the situation that was described in Example 1: the user is forced to rethink his/her proof against his/her original intuition; (s)he cannot use automation and has to input data manually, with all the proof-engineering problems already mentioned.

Even though this is a toy example, it is representative of an *extremely frequent* situation when using dependent rewrite. Actually this tactic seems to be seldom used in HOL4, maybe showing that these flaws prevent it from being useful in practice. Table 3 sums up the advantages of the different approaches.

3 Implementation

Let th be a theorem of the form $\vdash \forall x_1, \ldots, x_k.\ p \Rightarrow l = r$. In this section, we provide an implementation of implicational rewriting by th, called IR. This implementation requires the following steps:

1. go through the atoms of the goal, keeping track of their polarity;
2. for each atom, go through its subterms;
3. for each subterm t_i matching l with substitution σ_i, replace it by $r\sigma_i$ while keeping track of the matching substitution σ_i;

Table 2. Pros & cons of the different sorts of rewriting

	replaces $l\sigma$ by $r\sigma$	conditional equations	no need to prove the condition	compositionality
Rewriting	✓			
Cond. rewr.	✓	✓		
Dep. rewr.	✓	✓	✓	
Imp. rewr.	✓	✓	✓	✓

4. reconstitute the atom with the replaced subterms;
5. add the conjunction $p\sigma_1 \wedge p\sigma_2 \wedge \ldots$;
6. reconstitute the complete goal.

In addition, to obtain a valid tactic, the process should not just generate a formula ϕ, but also a *proof* that this formula entails the conclusion of the initial goal c. To do so, we actually generate a theorem $\vdash \phi \Rightarrow c$ (which holds if the implementation satisfies the specification of Definition 1, by Theorem 1).

Steps 2, 3, and 4 are implemented by a function IRC_{th} (for Implicational Rewriting Conversion) which takes an atom A as input and returns a theorem $p\sigma_1, \ldots, p\sigma_k \vdash A = A[r\sigma_1/l\sigma_1] \ldots [r\sigma_k/l\sigma_k]$[3]: IRC_{th} is a recursive function defined by case analysis on the structure of A:

$$\mathrm{IRC}_{th}(t) \stackrel{\text{def}}{=} \begin{cases} \dfrac{\vdash p \Rightarrow l = r}{p\sigma \vdash l\sigma = r\sigma} & \text{if } t \text{ has the form } l\sigma \\ & \text{for some substitution } \sigma \\ \dfrac{}{\vdash t = t} & \text{if } t \text{ is a variable or a constant} \\ \dfrac{\Gamma \vdash u = v}{\Gamma \vdash t = \lambda x.v} & \text{if } t \text{ has the form } \lambda x.u \\ & \text{and } \mathrm{IRC}_{th}(u) = \Gamma \vdash u = v \\ \dfrac{\Gamma_1 \vdash t_1 = u_1 \qquad \Gamma_2 \vdash t_2 = u_2}{\Gamma_1 \cup \Gamma_2 \vdash t_1 t_2 = u_1 u_2} & \text{if } t \text{ has the form } t_1 t_2, \\ & \mathrm{IRC}_{th}(t_1) = \Gamma_1 \vdash t_1 = u_1 \\ & \text{and } \mathrm{IRC}_{th}(t_2) = \Gamma_2 \vdash t_2 = u_2 \end{cases}$$

We impose that when the first rule and another can be applied, the first rule always has priority, and, if necessary, that th is renamed in order to avoid captures (i.e., so as not to contain any variable which is bound in the rewritten term). Since IRC_{th} must return a theorem, we present not only the resulting theorem but also the proof that allows to obtain it: for instance, the third rule means that $\mathrm{IRC}_{th}(\lambda x.u)$ calls first $\mathrm{IRC}_{th}(u)$; this recursive call must have the form $\Gamma \vdash u = v$, from which can thus be deduced $\Gamma \vdash \lambda x.u = \lambda x.v$. Note that, in the end, the function only returns this latter theorem, i.e., the result of the inference and not the inference itself, contrarily to what the definition of IRCmight suggest: this is done only for explanation purposes. The provided inferences are intended to give a clue to the reader about the way to obtain the result, but do not correspond to some precise inference rule: however all of them can be easily implemented using functions provided by both HOL Light and HOL4.

IRC_{th} can actually be seen as a *usual* rewriting by th. In practice, we therefore make use of *conversions* [23] to implement these steps (i.e., functions which, given a term t returns a theorem of the form $\vdash t = u$).

[3] Note that, in HOL, equality among booleans is just the same as equivalence.

Step 5 is then easily obtained from this result by propositional reasoning. The corresponding function is called AIR (for Atomic Implicational Rewriting) and exists both in a positive form AIR^+ and in a negative form AIR^-: it is up to the function which calls AIR to determine the adequate form according to the context (positive or negative atom). [4]

$$\mathrm{AIR}^+_{th}(A) \stackrel{\text{def}}{=} \frac{\Gamma \vdash A = A'}{\vdash (\bigwedge_{\phi \in \Gamma} \phi) \wedge A' \Rightarrow A} \qquad \mathrm{AIR}^-_{th}(A) \stackrel{\text{def}}{=} \frac{\Gamma \vdash A = A'}{\vdash A \Rightarrow \big((\bigwedge_{\phi \in \Gamma} \phi) \Rightarrow A'\big)}$$

where $\Gamma \vdash A = A'$ be the result of $\mathrm{IRC}_{th}(A)$.

Finally, steps 1 and 6 are achieved by $\mathrm{IR}^\pi_{th}(\phi)$, where ϕ is the intended conclusion of the goal to be implicationally rewritten and $\pi \in \{+, -\}$ is a polarity:

$$\mathrm{IR}^\pi_{th}(\phi) \stackrel{\text{def}}{=} \begin{cases} \mathrm{AIR}^\pi(\phi) & \text{if } \phi \text{ is atomic} \\ \dfrac{\vdash \phi_1 \Rightarrow \psi_1}{\vdash \neg\psi_1 \Rightarrow \neg\phi_1} & \text{if } \pi = + \text{ (resp. } -\text{),} \\ & \phi \text{ is a negation } \neg\phi_1 \text{ (resp. } \neg\psi_1\text{),} \\ & \text{and } \mathrm{IR}^{\overline{\pi}}_{th}(\phi_1)(\text{resp. } \psi_1) = \vdash \phi_1 \Rightarrow \psi_1 \\ \dfrac{\vdash \psi_1 \Rightarrow \phi_1 \qquad \vdash \psi_2 \Rightarrow \phi_2}{\vdash \psi_1 \wedge \psi_2 \Rightarrow \phi_1 \wedge \phi_2} & \text{if } \pi = + \text{ (resp. } -\text{),} \\ & \phi \text{ is a conjunction } \phi_1 \wedge \phi_2 \text{ (resp. } \psi_1 \wedge \psi_2\text{),} \\ & \mathrm{IR}^\pi_{th}(\phi_1)(\text{resp. } \psi_1) = \vdash \psi_1 \Rightarrow \phi_1\text{,} \\ & \text{and } \mathrm{IR}^\pi_{th}(\phi_2)(\text{resp. } \psi_2) = \vdash \psi_2 \Rightarrow \phi_2 \\ \dfrac{\vdash \psi_1 \Rightarrow \phi_1}{\vdash \forall\psi_1 \Rightarrow \forall\phi_1} & \text{if } \pi = + \text{ (resp. } -\text{),} \\ & \phi \text{ is a quantifed formula } \forall\phi_1 \text{ (resp. } \forall\psi_1\text{),} \\ & \text{and } \mathrm{IR}^\pi_{th}(\phi_1)(\text{resp. } \psi_1) = \vdash \psi_1 \Rightarrow \phi_1 \\ \dfrac{\vdash \phi_1 \Rightarrow \psi_1 \qquad \vdash \psi_2 \Rightarrow \phi_2}{\vdash (\psi_1 \Rightarrow \psi_2) \Rightarrow (\phi_1 \Rightarrow \phi_2)} & \text{if } \pi = + \text{ (resp. } -\text{),} \\ & \phi \text{ is an implic. } \phi_1 \Rightarrow \phi_2 \text{ (resp. } \psi_1 \Rightarrow \psi_2\text{),} \\ & \mathrm{IR}^{\overline{\pi}}_{th}(\phi_1)(\text{resp. } \psi_1) = \vdash \phi_1 \Rightarrow \psi_1\text{,} \\ & \text{and } \mathrm{IR}^\pi_{th}(\phi_2)(\text{resp. } \psi_2) = \vdash \psi_2 \Rightarrow \phi_2 \\ \text{Disjunction is handled like conjunction, simply replacing } \vee \text{ by } \wedge, & \\ \text{and exist. quantification like univ. quantification, replacing } \forall \text{ by } \exists. & \end{cases}$$

where $\overline{\pi}$ is defined as $\overline{+} \stackrel{\text{def}}{=} -$ and $\overline{-} \stackrel{\text{def}}{=} +$. As in the definition of IRC, not only we give the conclusion of the resulting theorem but also the (big-step) inference rule used to derive this theorem from the recursive calls. We call IRC "implicational conversions" because the rules are very similar to conversions, except that implication is used instead of equality.

At the top-level, only the positive polarity is used: in the end, IR^+_{th} returns a theorem of the form $\vdash \phi' \Rightarrow \phi$ where ϕ' is the implicationally rewritten version of ϕ. So, given a goal of conclusion c, one can call IR^+_c and apply the Modus Ponens tactic – i.e., the tactic which, given a goal of conclusion c and a theorem $c' \Rightarrow c$, turns c into c' – to the result. We can prove that this tactic indeed implements implicational rewriting:

[4] In practice, a distinction has to be made between the assumptions introduced by IRC and the assumptions that come from the original goal. This is easy to achieve but not presented for readability.

Theorem 2. *For every theorem th and every formula ϕ, the tactic consisting in applying Modus Ponens to $\mathrm{IR}^{+}_{th}(\phi)$ implements implicational rewriting by th.*

4 Refinements

In this section, we investigate a few refinements of implicational rewriting.

4.1 Theorems Introducing Variables

A usual problem with rewriting is how to handle theorems that introduce new variables when applying the rewrite, i.e., theorems of the form $\vdash l = r$, where r contains variables not occurring in l. Consider for instance the rewriting of the term $y + 0$ by the theorem $\vdash 0 = x - x$: since x does not appear in $y+0$, what sense would it have to replace 0 by $x-x$? And what if the original term indeed contains x, e.g., if we rewrite $x + 0$ instead of $y + 0$? Shall the rewrite replace 0 by $x-x$ or rename x to avoid a possibly unintended capture? This problem is usually considered to be rare enough that it is not worth considering: since x does not occur in the original term, the user simply should avoid rewriting with these theorems. However, it can make sense to apply such a rewrite, e.g., it can be useful to replace 0 by $x - x$ if, in the context of the proof, one wants to apply a theorem about substractions.

With implicational rewriting, this situation is even more frequent since new variables can also come from the condition of the theorem:

Example 3. Consider for example that one wants to prove $\forall s, \cdot, x, y, z, t.\ P \Rightarrow ((x \cdot y) \cdot z) \cdot t = x \cdot (y \cdot (z \cdot t))$, where P is an irrelevant premise. We assume that P entails in particular the predicate $group\ (s, \cdot)$, which states the usual group axioms for the operation $\cdot$ over s. A first step is to (implicationally) rewrite the goal with the associativity theorem $\vdash \forall g, op, x, y, z.\ group\ (g, op) \wedge x \in g \wedge y \in g \wedge z \in g \Rightarrow op\ (op\ x\ y)\ z = op\ (x (op\ y\ z))$. This yields: $\forall s, \cdot, x, y, z, t.\ P \Rightarrow group\ (g, \cdot)\ \wedge x \in g \wedge y \in g \wedge z \in g \wedge (x \cdot (y \cdot z)) \cdot t = x \cdot (y \cdot (z \cdot t))$. Here, the new variable g has been introduced in the goal whereas it does not have any meaning there. As a consequence, the goal is not provable anymore.

What happens is that matching $op\ (op\ x\ y)\ z$ (the l.h.s. of the associativiy theorem) indeed provides instantiations for op, x, y and z, but it does not provide any instantiation for g. But in practice, the user usually knows the instantiation of g, or will find it out later on in the course of the proof. Therefore, a satisfying solution would apply the rewrite but would still leave to the user the possibility to instantiate g manually. We achieve this by detecting automatically variables that are introduced by the rewrite, then applying the rewrite, and finally *quantifying existentially over the introduced variables.* In the above example, one would obtain the goal: $\forall s, \cdot, x, y, z, t.\ P \Rightarrow \exists g.\ group\ (g, \cdot)\ \wedge x \in g \wedge y \in g \wedge z \in g \wedge (x \cdot (y \cdot z)) \cdot t = x \cdot (y \cdot (z \cdot t))$. In our first example, $y + 0$ would be replaced by $\exists x.\ y + (x - x)$. This solution allows to maintain the provability of the goal, while preserving the advantages of (implicational) rewriting.

Formally, this consists simply in replacing the expression (1) of Definition 1 by $\exists x_1, \ldots, x_k.\ p\tau\sigma_1 \wedge \cdots \wedge p\tau\sigma_k \triangleright A[r\tau\sigma_1/l\sigma_1] \ldots [r\tau\sigma_k/l\sigma_k]$, where $x_1, \ldots, x_k$ denote all the variables introduced by the theorem $\vdash p\tau \Rightarrow l\tau = r\tau$ (i.e., formally, variables occurring in $r\tau$ and $p\tau$ but not in $l\tau$) and where τ is a renaming substitution used to avoid potential captures between the variables of the goal and the variables that occur in r and p but not in l. The proof of Theorem 1 still applies: only the base case of the induction slightly changes.

This refinement is implemented by modifying IRC so that the introduced variables are renamed to avoid possible captures; then by modifying AIR so that the function adds the required existential quantifications over these newly introduced variables. Note that the quantification's scope is only the premise (resp. conclusion) of the resulting theorem in the positive (resp. negative) case.

4.2 Preprocessing and Postprocessing

Preprocessing the input theorems to allow more theorems than just those of the form $\forall x_1, \ldots, x_k.\ p \Rightarrow l = r$ is a simple way to allow many improvements. For instance, purely equational theorems $\forall x_1, \ldots, x_k.\ l = r$ can be turned into $\forall x_1, \ldots, x_k.\ \top \Rightarrow l = r$, which entails that implicational rewrite can also be used as a substitute to standard rewriting.

In addition, some further preprocessing allows to accept theorems of the form $p \Rightarrow c$ (resp. $p \Rightarrow \neg c$) by turning them into $p \Rightarrow c = \top$ (resp. $p \Rightarrow c = \bot$). Note that this is commonly used in HOL4 and HOL Light for usual rewriting. For implicational rewriting, it also means that it can be used as a substitute for the (matching) Modus Ponens tactic, i.e., the tactic which, given a theorem of the form $\forall x_1, \ldots, x_k.\ p \Rightarrow c$ and a goal of the form $c\sigma$ for some substitution σ, generates the goal $p\sigma$. Indeed, in such a situation, implicational rewriting turns the goal $c\sigma$ into $\top$ (thanks to the described preprocessing) and adds the conjunct $p\sigma$. We thus obtain the goal $p\sigma \wedge \top$ which is equivalent to what is obtained by the matching Modus Ponens. However, implicational rewriting is then even more powerful than this tactic since it is able to apply this sort of reasoning *deeply* (the lack of this feature is a common criticism).

However, this latter solution is a little bit unsatisfying since it yields $p\sigma \wedge \top$ instead of the expected $p\sigma$. Of course it is trivial to get rid of $\top$ here, but it would obviously be better to achieve this automatically. This is easily done by maintaining a set of basic rewriting theorems containing commonly used propositional facts like $\forall p.\ p \wedge \top \Leftrightarrow p$, or $\top \wedge \top \Leftrightarrow \top$. Again, similar solutions are used for rewriting in HOL4 and HOL Light. In order to do this rewriting on the fly, implicational rewriting must be adapted to be able to rewrite with several theorems. Since the current definitions already handle multiple rewrites with the same theorem, it is trivial to extend them to deal with several theorems simply by considering a set of theorems instead of just one.

Finally, theorems of the form $\forall x_1, \ldots, x_k.\ p \Rightarrow \forall y_1, \ldots, y_n.\ l = r$ (i.e., additional quantifiers appear before the equation) can also be handled at no cost (all definitions adapt trivially).

4.3 Taking the Context into Account

Example 4. Consider the goal $\forall x, y.\ x \neq 0 \Rightarrow \frac{x}{x} * y = y$. Applying implicational rewriting with $\vdash \forall x.\ x \neq 0 \Rightarrow \frac{x}{x} = 1$ yields the goal $\forall x, y.\ x \neq 0 \Rightarrow x \neq 0 \wedge 1 * y = y$. The context obviously entails the inner $x \neq 0$, but one still needs additional manual reasoning to obtain $\forall x, y.\ x \neq 0 \Rightarrow 1 * y = y$.

Therefore a further refinement is to modify implicational rewriting so that the context is taken into account. This can be handled by adding the contextual hypotheses to the set of usable theorems, while going down the theorem. This means that, e.g., in the positive case of the implicational rule of IR's definition (fourth case), the formula ϕ_1 is added to the set of theorems that can be used by the recursive call to compute ψ_2 from ϕ_2. Similar treatments

can be applied in the negative case and for conjunction. This is overall similar to what is done in usual rewriting to handle the context, see, e.g., [23] for a sketch of these ideas. However these additions require a particular care in order, in particular, to avoid recomputing the same contexts several times, see. Many solutions exist, see, e.g., the source codes of HOL4, HOL Light or Isabelle. Note that this allows to get rid of many of the introduced conditions automatically, exactly like conditional rewrite does. Therefore implicational rewriting can also often be used as a replacement for conditional rewriting.

In the end, all these refinements (including efficient use of the context) are implemented in one single tactic called `IMP_REWRITE_TAC`: this tactic takes a list of theorems and applies implicational rewriting with all of them repeatedly until no more application is possible. Note that all these refinements are not just small user-friendly improvements: they also improve again the compositionality of the tactic by allowing to chain seamlessly implicational rewrites of many theorems. Since, as shown in the above refinements, this tactic subsumes rewriting, conditional rewriting, and Modus Ponens tactics, it integrates very well in the usual tactic-style proving workflow: one can use just one tactic to cover all the other cases, plus the ones that were not covered before. In practice, this combination happens to be *extremely powerful*: many proof steps can be turned into only one call to this tactic with several theorems. For instance, the tactic has been extensively used for months to develop in particular the work presented in [19,20,27]. In particular, the library presented in [20] was completely rewritten using implicational rewriting, which showed a dramatic reduction of its code size. In addition the time taken to prove new theorems was also much reduced due to the relevant feedback provided by the tactic.

The main improvements we foresee for the above refinement process are regarding performance. This could probably benefit from all the optimizations that already exist for usual rewriting, e.g., [22]. Another easy possible refinement would be to allow implicational rewriting in the assumptions instead of the conclusion of the goal. This should be easily achieved simply by considering the reverse implication.

5 Other Interaction-Intensive Situations

In this section, we tackle the automation of situations that present the same loopholes as the ones that motivated our development of implicational rewriting, i.e., situations where the user has to input manually some information that could be computed automatically by the theorem prover, leading to fragility of proof scripts and tediousness of the interaction.

5.1 Contextual Existential Instantiation

Consider a slight variation of Example 3:

Example 5. Let be the goal $\forall s, \cdot, x, y, z, t.\ group\ (s, \cdot) \Rightarrow ((x \cdot y) \cdot z) \cdot t = x \cdot (y \cdot (z \cdot t))$. Applying implicational rewriting now yields: $\forall s, \cdot, x, y, z, t.\ group\ (s, \cdot) \Rightarrow \exists sg.\ group\ (sg, \cdot) \wedge (x \cdot (y \cdot z)) \cdot t = x \cdot (y \cdot (z \cdot t))$. The usual solution is then to strip the quantifiers and discharge $group\ (s, \cdot)$ to obtain $\exists sg.\ group\ (sg, \cdot) \wedge (x \cdot (y \cdot z)) \cdot t = x \cdot (y \cdot (z \cdot t))$ as a conclusion. Then one can provide explicitly the witness s for sg in order to obtain: $group\ (s, \cdot) \wedge (x \cdot (y \cdot z)) \cdot t = x \cdot (y \cdot (z \cdot t))$.

Here, the user must provide explicitly a witness. In this example, the witness is just the one character s, but in other situations it can be big complex terms. Therefore, the same reproaches can be applied to this situation as those that gave raise to implicational rewriting (fragility of the script, tediousness for the user, etc.). However, once again, it is a situation where the context could be used by the theorem prover to find the witness by itself: it is easy to go through all the assumptions and find an atom which matches an atom of the conclusion (e.g., in the above example, $group\ (s, \cdot)$ matches $group\ (sg, \cdot)$).

One can even instantiate existential quantifications that appear *deep* in the goal instead of just as the top connective. Then, not only the assumptions, but also the context of the formula can be used. This was implemented in a tactic called `HINT_EXISTS_TAC`, of which a preliminary version has already been integrated into HOL4. As shown in the above example, this sort of situation can also happen when using implicational rewriting, therefore this tactic is also integrated transparently in `IMP_REWRITE_TAC`.

Note that the underlying algorithm shares some common structure with the one of implicational rewriting. More precisely, IR can actually be reused only changing the base call to AIR. For space reasons, we refer to the source code in [1] for the details of the implementation.

5.2 Cases Rewrite

Implicational rewriting can be seen as a situation where the user has a at his/her disposal a theorem $\vdash \forall x_1, \ldots, x_k.\ p \Rightarrow c$ *and is ready to accept whatever it takes* to make use of the information provided by c. Since this cannot be done at no cost, implicational rewriting accepts to do it, at the condition to add the necessary instantiation of p. When one uses implicational rewriting, one makes the underlying assumption that (s)he will have the ability to prove p later on.

But, sometimes, we do not want to make such a strong assumption; instead, we want to split the proof by considering both what happens if p holds and what happens if p does not hold. In such cases, the usual solution is to explictly use a case-split tactic to consider two branches of the proof: one where p holds, and one where $\neg p$ holds. But, once again, the user has to explicitly state some information which is possibly verbose, fragile and tedious, when the prover could do the same automatically by retrieving the relevant information from the theorem $\vdash \forall x_1, \ldots, x_k.\ p \Rightarrow c$.

This yields *cases rewriting* which requires also a theorem of the form $\vdash \forall x_1, \ldots, x_k.\ p \Rightarrow l = r$ (or $\vdash \forall x_1, \ldots, x_k.\ p \Rightarrow c$ with preprocessing) and looks for an atom A with a subterm matching l (say with substitution σ). However, unlike implicational rewriting, it does not replace A by $p\sigma \wedge A[r\sigma/l\sigma]$ or $p\sigma \Rightarrow A[r\sigma/l\sigma]$, but rather by $(p\sigma \Rightarrow A[r\sigma/l\sigma]) \wedge (\neg p\sigma \Rightarrow A)$. This was implemented in the tactic `CASES_REWRITE_TAC`. We refer to the manual of [1] for more details.

5.3 Target Rewrite

Example 6 ([2]). Consider the goal $\forall n, m.\ SUC\ n \leq SUC\ m \Rightarrow n \leq m$. Assume we already proved that the predecessor function is monotonic: $\vdash \forall n, m.\ n \leq m \Rightarrow PRE\ n \leq PRE\ m$ and that the predecessor is the left inverse of the successor: $\vdash \forall n.\ PRE\ (SUC\ n) = n$. A natural proof would start by replacing $n \leq m$ by $PRE\ (SUC\ n) \leq PRE\ (SUC\ m)$. But rewriting with $\vdash \forall n.\ n = PRE\ (SUC\ n)$ will obviously not terminate. So we should rewrite the goal only once (as allowed by the HOL4 and HOL Light tactic `ONCE_REWRITE_TAC`). But

this rewrites everywhere, yielding: $\forall n, m.\ PRE\,(SUC\,(SUC\,n)) \leq PRE\,(SUC\,(SUC\,m)) \Rightarrow PRE\ (SUC\ n) \leq PRE\ (SUC\ m)$. Instead, we have to use a special tactic allowing to state precisely that we want to rewrite only in the conclusion of the implication, and only once (e.g., `GEN_REWRITE_TAC` in HOL Light and HOL4), or more elaborate solutions like [13] in Coq). This requires to provide explicitly to the prover which part of the goal has to be rewritten: so, exactly as in the previous situations, the user has to provide explicitly some information which is very unintuitive and very dependent on the current shape of the goal. Thus, once again, this solution is both fragile and tedious to the user.

In this example, the important information is actually not the *location* of the rewrite, but the *objective* that the user has in mind. And this objective is to rewrite the goal *in order to use* the theorem $\vdash \forall n, m.\ n \leq m \Rightarrow PRE\ n \leq PRE\ m$. Generally, when one wants to apply a precisely located rewrite with a theorem, the underlying objective is to get a goal which allows the use of *another* theorem. So we define *target rewriting* which takes *two theorems* as input: the one used for the rewrite (called *supporting* theorem) and the one that we intend to use after the rewrite (called *target* theorem). Very naively, the tactic simply explores all the possible rewrites of the goal using the supporting theorem until one of these rewrites yields a term which can be rewritten by the target theorem.

Example 7. In Example 6, the supporting theorem is $\vdash \forall n.\ n = PRE\ (SUC\ n)$ and the target theorem is $\vdash \forall n, m.\ n \leq m \Rightarrow PRE\ n \leq PRE\ m$. The list of all possible 1-step rewrites explored by the tactic is the following:

1 $\forall n, m.\ PRE\ (SUC\ (SUC\ n)) \leq SUC\ m \Rightarrow n \leq m$
2 $\forall n, m.\ SUC\ n \leq PRE\ (SUC\ (SUC\ m)) \Rightarrow n \leq m$
3 $\forall n, m.\ SUC\ n \leq SUC\ m \Rightarrow PRE\ (SUC\ n) \leq m$
4 $\forall n, m.\ SUC\ n \leq SUC\ m \Rightarrow n \leq PRE\ (SUC\ m)$

Then the list of possible 2-step rewrites is enumerated as follows:

1 $\forall n, m.\ PRE\ (SUC\ (SUC\ n)) \leq PRE\ (SUC\ (SUC\ m)) \Rightarrow n \leq m$
2 $\forall n, m.\ PRE\ (SUC\ (SUC\ n)) \leq SUC\ m \Rightarrow PRE\ (SUC\ n) \leq m$
3 $\forall n, m.\ PRE\ (SUC\ (SUC\ n)) \leq SUC\ m \Rightarrow n \leq PRE\ (SUC\ m)$
4 $\forall n, m.\ SUC\ n \leq PRE\ (SUC\ (SUC\ m)) \Rightarrow PRE\ (SUC\ n) \leq m$
5 $\forall n, m.\ SUC\ n \leq PRE\ (SUC\ (SUC\ m)) \Rightarrow n \leq PRE\ (SUC\ m)$
6 $\forall n, m.\ SUC\ n \leq SUC\ m \Rightarrow PRE\ (SUC\ n) \leq PRE\ (SUC\ m)$

Here the tactic stops because the last rewrite allows to apply the target theorem.

Because of the exhaustive enumeration, this tactic can of course be extremely costly. Still, in the numerous practical cases where its execution time is reasonable, it is tremendously helpful since the user does not have to provide explicit information anymore: the only additional effort is to step back and look a step ahead in the intended proof to know which theorem shall be used afterwards. Many concrete situations happen to match this use case pattern. For instance, associativity-commutativity (AC) rewriting [28] is a particular case of it:

Example 8. Consider the goal $a + (b * c + -a) + a = b * c + a$ [13]. A standard proof of this goal is to first rearrange the innermost addition into $a+(-a+b*c)+a = b*c+a$ by carefully using the commutativity of addition and then using the theorem $\vdash \forall x\ y.\ x + (-x + y) = y$ to conclude. Instead, one can just use target rewriting with the AC of addition as the supporting theorem and $\vdash \forall x\ y.\ x + (-x + y) = y$ as the target theorem.

We called this tactic `TARGET_REWRITE_TAC`. It is actually able to take several supporting theorems as input (though, of course, more supporting theorems means more possibilities to explore, and thus a bigger execution time). In addition, it actually does not work with rewriting but with implicational rewriting, which allows to use implicational theorems as supporting theorems.

6 Related Work

In proof theory, applying reasoning *deep* in a goal is the precise focus of deep inference [7]. At first sight, it can hardly be said that implicational rewriting shares more than a conceptual relation to deep inference though, but some of the benefits can be seen as similar since in both cases the fact of reasoning deep allows to get rid of some "bureaucratic" manipulations. If considering sets of clauses instead of arbitrary formulas, implicational rewriting turns out to be very close to *superposition* [4]. Since formulas are normalized, superposition does not need to consider the polarity of the formula where it applies, therefore it actually corresponds only to the negative case of implicational rewriting. Studying this connection in detail to improve our implementation is part of future work.

As already mentioned, the implementation of IR can be seen as a particular form of rewriting where implication is used instead of equality. This is very similar to the "consequence conversions" in HOL4 [33], and more generally can be seen as a particular case of rewriting with preorders [17,32], where the used preorders are both $\Rightarrow$ and $\Leftarrow$ used in an interleaved way. Taking the context into account builds on top of several implementations serving similar ideas. Conceptually, one can find many connections with "window inference" [26].

Target rewriting is very close to the "smart matching" tactic of Matita [2]: this tactic uses as supporting theorems the whole database of already proven equational theorems and the target theorem is provided explicitly like in our approach. It uses a sophisticated implementation of superposition instead of our naive approach which just enumerates all possible rewrites, thus making it much more efficient. However it uses only equational theorems as supporting theorems, whereas target rewriting also accepts implicational theorems: this was particularly useful in, e.g., [20], where most theorems are prefixed by assumptions. In addition, smart matching only tries to match the top goal, whereas target rewriting also works deeply, with the same advantages as for implicational rewriting: no need to use book-keeping tactics, and therefore more compositionality. This is made possible precisely because we use implicational rewriting instead of matching Modus Ponens as is done for smart matching. However, this has of course a much bigger impact on the performance. The perfect solution would probably lie in between: using superposition to make target rewriting more efficient, or extending smart matching with ideas of target rewriting.

Conceptually, both smart matching and target rewriting are connected to *deduction modulo* [11] since they are essentially about making a distinction between the "important" steps of a proof and the ones that are just "glue" between the important steps: in our context, this glue is the use of supporting theorems; in the context of smart application, it is the use of the available equational knowledge base; and in deduction modulo, it is "calculation" steps as opposed to deduction steps. However, to the best of our knowledge, there is no tool similar to target rewriting or smart matching making use of deduction modulo.

Finally, as explained earlier, AC-rewriting can be seen as a special case of target rewriting. Many works have been devoted to this, e.g., in HOL90 [28], or more recently in Coq [6].

The advantage of these works is that they are of course more efficient, since they deal with a very special case. But they are much less general, and therefore not as useful as target rewriting.

7 Conclusion

We presented in this paper some tactics to reduce human interaction in interactive theorem provers. Their objective is not to provide the automation of intricate reasoning that the user could not achieve by him/herself, but rather to assist him/her in some quite simple but frequent reasoning tasks. The most important of our tactics is implicational rewriting, whose core idea was presented in detail. We argued how a big advantage of it is that it allows for a better composionality thus making it extremely useful in practice. We presented an implementation of implicational rewriting as well as some refinements improving further its usefulness. Finally we covered a few other tactics pursuing similar objectives of reducing the tediousness of human interaction and the fragility of proof scripts. The objective of this latter aspect is also to improve human interaction, but in a longer term perspective: when proof scripts are robust to change, they make the development of theories easier, and thus improve the user experience.

In practice, many proofs are surprisingly sequences of calls to `IMP_REWRITE_ TAC` interleaved with a few calls to `TARGET_REWRITE_TAC`, and only rarely other tactics. Of course, in cases where human creativity is really required, some subgoals or lemmas are set, but this is to be expected when reasoning in higher-order logic. Apart from these cases, one can observe that these tactics serve the purpose they were designed for: many reasoning tasks which are simple for a human become simple with the theorem prover. *In the end, the only reproach which can be made to this approach is that removing most explicitly set subgoals reduces the readability of the proof scripts.* We argue instead that the purpose of a proof script is not to mimic usual mathematical proofs: the information that can be found automatically should be found out by the machine. Tools like Proviola [31] can still be used in order to provide some valuable feedback to the user.

Acknowledgements. The author thanks L. Liu, M. Y. Mahmoud and G. Helali from Concordia University for their feedback after extensive use of this work, as well as N. Peltier from the Laboratory of Informatics of Grenoble, and the contributors of the hol-info mailing list for fruitful discussions.

References

1. Aravantinos, V.: Implicational Conversions for HOL Light and HOL4 (2013), `https://github.com/aravantv/impconv`, `https://github.com/aravantv/impconv/HOL4-impconv` (respectively)
2. Asperti, A., Tassi, E.: Smart Matching. In: Autexier, S., Calmet, J., Delahaye, D., Ion, P.D.F., Rideau, L., Rioboo, R., Sexton, A.P. (eds.) AISC/Calculemus/MKM 2010. LNCS (LNAI), vol. 6167, pp. 263–277. Springer, Heidelberg (2010)
3. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1999)
4. Bachmair, L., Ganzinger, H.: Rewrite-Based Equational Theorem Proving with Selection and Simplification. Journal of Logical Computation 4(3), 217–247 (1994)
5. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories. In: Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 825–885. IOS Press (2009)

6. Braibant, T., Pous, D.: Tactics for Reasoning Modulo AC in Coq. In: Jouannaud, J.-P., Shao, Z. (eds.) CPP 2011. LNCS, vol. 7086, pp. 167–182. Springer, Heidelberg (2011)
7. Brünnler, K., Tiu, A.F.: A Local System for Classical Logic. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 347–361. Springer, Heidelberg (2001)
8. Cooper, D.C.: Theorem Proving in Arithmetic Without Multiplication. Machine Intelligence 7, 91–99 (1972)
9. Brand, D., Darringer, J., Joyner, W.: Completeness of Conditional Reductions. Research Report RC-7404, IBM (1978)
10. Davis, M., Logemann, G., Loveland, D.W.: A Machine Program for Theorem-Proving. Communications of the ACM 5(7), 394–397 (1962)
11. Dowek, G., Hardin, T., Kirchner, C.: Theorem Proving Modulo. Journal of Automated Reasoning 31(1), 33–72 (2003)
12. Fitting, M.: First-Order Logic and Automated Theorem Proving. Texts and Monographs in Computer Science. Springer (1990)
13. Gonthier, G., Tassi, E.: A Language of Patterns for Subterm Selection. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 361–376. Springer, Heidelberg (2012)
14. Harrison, J.: The HOL Light System Reference (2013), `http://www.cl.cam.ac.uk/~jrh13/hol-light/reference.html`
15. Homeier, P.V.: HOL4 source code (2002), `http://ww.src/1/dep_rewrite.sml`
16. Huet, G.P.: A Mechanization of Type Theory. In: International Joint Conference on Artificial Intelligence, pp. 139–146. William Kaufmann (1973)
17. Inverardi, P.: Rewriting for preorder relations. In: Lindenstrauss, N., Dershowitz, N. (eds.) CTRS 1994. LNCS, vol. 968, pp. 223–234. Springer, Heidelberg (1995)
18. Kroening, D., Strichman, O.: Decision Procedures: An Algorithmic Point of View. Texts in Theoretical Computer Science. An EATCS Series. Springer (2008)
19. Liu, L., Hasan, O., Aravantinos, V., Tahar, S.: Formal Reasoning about Classified Markov Chains in HOL. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 295–310. Springer, Heidelberg (2013)
20. Mahmoud, M.Y., Aravantinos, V., Tahar, S.: Formalization of Infinite Dimension Linear Spaces with Application to Quantum Theory. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 413–427. Springer, Heidelberg (2013)
21. Mayr, R., Nipkow, T.: Higher-Order Rewrite Systems and Their Confluence. Theoretical Computer Science 192(1), 3–29 (1998)
22. Norrish, M.: Rewriting Conversions Implemented with Continuations. Journal of Automated Reasoning 43(3), 305–336 (2009)
23. Paulson, L.C.: A Higher-Order Implementation of Rewriting. Science of Computer Programming 3(2), 119–149 (1983)
24. Paulson, L.C., Blanchette, J.C.: Three Years of Experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers. In: IWIL@LPAR. EPiC Series, vol. 2, pp. 1–11 (2010)
25. Robinson, J.A., Voronkov, A. (eds.): Handbook of Automated Reasoning. Elsevier and MIT Press (2001)
26. Robinson, P.J., Staples, J.: Formalizing a Hierarchical Structure of Practical Mathematical Reasoning. Journal of Logical Computation 3(1), 47–61 (1993)
27. Siddique, U., Aravantinos, V., Tahar, S.: Formal Stability Analysis of Optical Resonators. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 368–382. Springer, Heidelberg (2013)
28. Slind, K.: AC Unification in HOL90. In: Joyce, J.J., Seger, C.-J.H. (eds.) HUG 1993. LNCS, vol. 780, pp. 436–449. Springer, Heidelberg (1994)

29. Slind, K., Norrish, M.: A Brief Overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 28–32. Springer, Heidelberg (2008)
30. Solovyev, A.: SSReflect/HOL Light manual. Flyspeck project (2012)
31. Tankink, C., Geuvers, H., McKinna, J., Wiedijk, F.: Proviola: A Tool for Proof Re-animation. In: Autexier, S., Calmet, J., Delahaye, D., Ion, P.D.F., Rideau, L., Rioboo, R., Sexton, A.P. (eds.) AISC/Calculemus/MKM 2010. LNCS (LNAI), vol. 6167, pp. 440–454. Springer, Heidelberg (2010)
32. Türk, T.: HOL4 source code (2006), `http://src/simp/src/congLib.sml`
33. Türk, T.: HOL4 source code (2008), `http://src/1/ConseqConv.sml`