Contents

.

1 Simplification of Herbrand Sequents

Simplification of Herbrand Sequents is needed in order to improve the readability of the sequent as well as to delete the information which is useless for interpreting the mathematical meaning encoded in it. The simplification goes through two different steps. The first one is simplification on term level. The second one is simplification on logical (formula) level.

- In the first step we try to rewrite each term in each atom formula to a term which is in normal form according to a given confluent and terminating system of rewriting rules. As a result we obtain the minimal sequent S_{min}, where S be the sequent to be simplified.

- The simplification in formula level takes S_{min} and tries to remove all formula occurrences which are irrelevant for the validity with respect to the background theory. The following steps are executed:

  1) transforms S_{min} to a formula and negate it. Let the resulting formula be F

  2) remove all implications from F. The resulting formula F' contains $\wedge, \vee$ and $\neg$ as logical symbols only.

  3) F' is transformed to a formula F'' in Negation Normal Form (NNF).

  4) to each atomic formulas in F'' is assigned a unique label (a natural number) and this label is encoded into the name of the corresponding atomic formula.

  5) F'' is transformed to a formula F''' in Conjunctive Normal Form (i.e. F''' is in Clause Form).

  6) F''' is given to the theorem prover (Otter[1] or Prover9) which returns a resolution refutation ρ.

  7) ρ is analyzed and all atomic formulas in F'' which are also in ρ are marked with a special marker (in our case $\star$).

  8) decode the names of all atomic formulas in F'', i.e. remove the label from the predicate name of each atomic formula. Keep the marker of all marked atomic formulas. Call the sequent $F'^{\star}$

  9) for each formula occurrence in $F'^{\star}$ check whether all atomic formulas are NOT marked. If this is the case, remove the whole formula

occurrence. Call the sequent $F^\star$. This is the simplified Herbrand sequent.

For Otter[1] and Prover9 there is already written an interface in CERES [2] which allows manipulation and visualization of the obtained result with the Prooftool[7].

The next two subsection describe the term and formula simplification in details.

1.1 Simplification on the term level

The simplification of the Herbrand sequent in term level is an algorithm which rewrites the sequent to a minimal one with respect to the ordering $\geq_{seq}$. That means that we firs have to rewrite all terms to a normal form. The rewriting of the terms is done according to a term-rewriting system. In order to guarantee the existence of unique normal form for each term, we assume that the term-rewriting system is confluent and terminating.

The term-rewriting algorithm is described as follows:

INPUT: a Herbrand Sequent S
OUTPUT: a minimal Herbrand Sequent with respect to $\geq_{seq}$
VARIABLES:
occ : a formula occurrence in the sequent
pos : a term position in a term
P : atomic formula
$l \rightarrow r$: a rule
σ : substitution
t : term
TRS : set of rules

Algorithm 1.1: *RewritingTermsInHerbrandSequent*(S)

```
for each occ ∈ S
  do { for each P ∈ occ
       do { repeat
            for each pos ∈ P
              do { for each (l → r) ∈ TRS
                   do { t ← P.pos
                        if unifiable(t, l)
                          then { σ ← mgu(t, l)
                                 P.pos ← σ(r) }
                   }
            until noFurtherReductionIsPossble
       }
  }
return (S
```

The implementation of the algorithm in C++ can be seen in the file opHerbrandSequentSimiplication which contains a all operation classes. It is a part of the *prooflib* library of the CERES system.

Theorem 1.1 (correctness)**.** The algorithm for term-rewriting a Herbrand Sequent returns the minimal sequent with respect to the ordering $\geq_{seq}$.

Proof:

Since we are using a terminating and confluent term-rewriting system, then each term has a normal form. This normal form is the minimal unique term with respect to the ordering $\geq_\tau$.

Once all terms in all atomic formulas in a formula occurrence are in normal form, then according to the definition of $=_E^f$, the whole formula corresponding to this formula occurrence is the minimal one with respect to $>_f = \{(P(t_1, \ldots, t_n), P(s_1, \ldots, s_n)) |\ s_i$ is the normal form of t_i, for $i = 1, \ldots, n, s, t \in T, P \in PS\}$.

Since each formula corresponding to each formula occurrence is the minimal one, then S_{min} is the smallest sequent according to. the ordering $>_{seq} = \{(S, S') |\ S'$ is the result of substitution a formula with the minimal one$\}$. $\square$

1.2 Simplification on the formula level

Simplifying a sequent in logical (formula) level consists in removing all formula occurrences in a Herbrand sequent which are irrelevant for the validity of the sequent as well as marking these atom formulas that are not essential for the validity of the sequent but which can not be deleted. A formula occurrence can be deleted only in the case that all atom formulas in it are marked by the algorithm. Formally, let $S = A_1, \ldots, A_n \vdash C_1, \ldots, C_m$ be a Herbrand sequent. Our goal is to find a sequent $S^{occ}_{min} = A_{i_1}, \ldots, A_{i_k} \vdash C_{j_1}, \ldots, C_{j_r}$, $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ and $\{j_1, \ldots, j_r\} \subseteq \{1, \ldots, m\}$, such that S^{occ}_{min} is minimal with respect to the number of formula occurrences and is still a valid sequent. Hence, according to the definition, it is also a Herbrand sequent. The general idea is that we negate the extracted Herbrand sequent S and transform it into a formula $\varphi = A_1 \wedge \cdots \wedge A_n \wedge (\neg C_1) \wedge \cdots \wedge (\neg C_m)$. Then, we transform φ into equivalent formula φ' in Negation Normal Form (NNF). For this purpose we apply the De-Morgan rules to each disjunct in φ pushing the negation as much as possible according to the following rules (we apply them to each conjunct in φ till no further reduction is possible):

$1)(F_1 \rightarrow F_2) \Rightarrow (\neg F_1 \vee F_2)$

$2)\neg(F_1 \wedge F_2) \Rightarrow (\neg F_1 \vee \neg F_2))$

$3)\neg(F_1 \vee F_2) \Rightarrow (\neg F_1 \wedge \neg F_2))$

$4)\neg\neg F \Rightarrow F$

The formulas F, F_1 and F_2 are quantifier-free and only conjunction, disjunction and negation signs can appear. The cases for quantifier formula are omitted because the extracted Herbrand sequent contains grounded quantifier-free formulas only.

The last transformation applies the distributivity laws. The obtained formula in clause form, i.e. $\varphi_{CNF} = H_1 \wedge \cdots \wedge H_p$, such that each H_i, $i \in \{1, \ldots, p\}$ is a clause, i.e. $H_i = B_{k_1} \vee \cdots \vee B_{k_q}$, where each B_j is a literal (atom formula or its negation). In fact this is exactly the conjunctive-normal form of φ.

Then, we transform φ_{CNF} to a clause set. Then we give this clause set and the axioms of the background theory to the theorem prover. The result of the theorem prover is a refutation tree. The atom formulas in the Herbrand sequent which occur in formulas which are used in a particular

refutation are marked. All non-marked formula can be removed from the Herbrand sequent, in the following two cases:

1)if the corresponding formula occurrence consists of only this atom formula

2)if all atom formulas in a corresponding formula occurrence are not marked. Then the whole formula occurrence can be dropped.

Otherwise we just keep the marked formula. The algorithm in pseudocode for the simplification in formula level looks as follows:

INPUT: a Herbrand Sequent S with terms in normal form

OUTPUT: minimal Herbrand Sequent

VARIABLES:

$S = A_1, \ldots, A_n \vdash C_1, \ldots, C_m$: a sequent with quantifier-free grounded formulas only

S' : the empty sequent

$\varphi, \varphi' F_1, F_2, F_1', F_2'$: quantifier-free ground formulas

$clauseSet$, $axiomClauseSet$: sets of clauses

$resProof$: a resolution proof

Algorithm 1.2: *SequentToClause*(S)

$\varphi \leftarrow A_1 \wedge \cdots \wedge A_n \wedge (\neg C_1) \wedge \cdots \wedge (\neg C_m)$
for each $(i \in \{1,n\}, j \in \{1,m\})$
do $\begin{cases} A_i \leftarrow \text{RemoveImplication}(A_i) \\ C_j \leftarrow \text{RemoveImplication}(C_j) \end{cases}$

for each $(i \in \{1,n\}, j \in \{1,m\})$
do $\begin{cases} A_i \leftarrow \text{TransformToNNF}(A_i) \\ C_j \leftarrow \text{TransformToNNF}(C_j) \end{cases}$

$\varphi' \leftarrow \text{LabelingAtomFormulas}(\varphi)$

$\varphi' \leftarrow \text{RenameAtomFormulasToLabels}(\varphi')$

$clauseSet \leftarrow \text{makeEqualities}(\varphi, \varphi')$

$\varphi_{CNF} \leftarrow \text{TransformToCNF}(\varphi')$

$clauseSet \leftarrow \text{union}(clauseSet, \text{TransformToClauseSet}(\varphi_{CNF}))$

$resProof \leftarrow \text{theoremProver}(clauseSet)$

$axiomClauseSet \leftarrow \text{getAxiomClauses}(resProof)$

$\varphi' \leftarrow \text{markAtomFormulas}(axiomClauseSet, \varphi')$

$\varphi'' \leftarrow \text{un-renameAtomFormulas}(\varphi')$

$S' \leftarrow \text{TransformFormulaToSequent}(\varphi'')$

for each $(occ \in S')$
do $\begin{cases} \textbf{if } (\text{allAtomFormulasInOccurrenceAreMarked}(occ)) \\ \quad \textbf{then } \text{delete}(occ, S') \end{cases}$

return (S')

Now we give an explanation about the function which are called in the

pseudo-code above. Function *RemoveImplication* applies the De Morgan's rules to transform a formula to a formula containing only conjunction, disjunction and negation.

Function *TransformToNNF* transforms a formula into a negation-normal form applying the rules described above.

Function *LabelingAtomFormulas* sets **unique** labels to all atom formulas. The labels are set linearly with respect to the formula seen as a linear text, not as a binary tree.

The function *RenameAtomFormulasToLabels* renames all atom formulas in a way such that the corresponding label is coded into the new predicate name. The reason for this renaming is because we want to keep an eye on each atom formula in the returned from the theorem prover refutation tree. Since it is quite possible the same atom formula to occur in formulas in different formula occurrences, it is impossible to understand where a formula in the refutation tree comes from if there are many such formulas in the Herbrand sequent. We illustrate this with the following simple example. Assume that this is a Herbrand sequent :

$(A \rightarrow B), A \vdash \neg A, B$, where A and B are atomic formulas. We can see that the formulas A and B occur in different occurrences. We can only notice that there is a redundancy of the formula A in the first and in the second formula occurrences in the antecedent part of the sequent. How can we decide which A could be removed and the sequent to be still a Herbrand sequent? According to the Herbrand sequent definition, only whole formula occurrences can be removed. So, we have to see whether the second formula occurrences A in the antecedent part can be removed. Indeed, it can be removed. To see this, we transform the sequent to a formula $(\neg A \vee B) \wedge A \wedge A \wedge \neg B$. The theorem prover produces a refutation from the clauses $(\neg A \vee B), A, \neg B$. But now the system does not know where the second A comes from. Exactly for this reason we label the atom formulas and encode the labels into their names. Then of course, we add the clauses which say that if a formula has two labels, then the renamed formulas are equivalent.

The function *makeEqualities* returns a set of clauses representing an equalities between all renamed formulas which have the same un-renamed origin formula.

The function *TransformToCNF* transforms a formula to a conjunctive-normal form. This is needed in order to obtain it as a set of clauses. Each conjunct is a clause. The atom formulas with negative polarity in each conjunct go to antecedent part of the sequent and those with positive polarity go in the consequent part of the sequent.

The function *union* performs a union of the two sets of clauses.

The function *theoremProver* calls the theorem prover and returns the refutation tree.

The function *getAxiomClauses* takes the axioms from the resolution refutation tree. In fact we take only axioms because all other nodes of the resolution refutation are subsequents of the axioms.

The function *markAtomFormulas* marks with a marker those atom formulas in the renamed Herbrand sequent which occur in the set of axioms obtained from the resolution refutation.

The function *un-renameAtomFormulas* renames the renamed atom formulas with their original names. It keeps the marker of all marked formulas.

The function *TransformFormulaToSequent* transforms the negation of the formula to a sequent. In this way we obtain the original Herbrand sequent but with marked atom formulas. This allows us to remove those formula occurrences of the Herbrand sequent whose atom formulas are all marked.

allAtomFormulasInOccurrenceAreMarked is a predicate which checks whether all atom formulas in a formula occurrence are marked.

The function *delete* deletes the whole formula occurrences from the sequent.

2 Examples

Consider the following two examples:

Example 2.1. *Let* $S = P(0), P(0), P(0) \to P(1) \vdash P(1), P(1) \wedge P(2)$

Construct the negation of S *and transform it to a formula:*

$$F = P(0) \wedge P(0) \wedge (P(0) \to P(1)) \wedge (\neg P(1)) \wedge \neg(P(1) \wedge P(2))$$

Labeling and renaming of the atom formulas:

$$F' = P_1(0) \wedge P_2(0) \wedge (P_3(0) \to P_4(1)) \wedge (\neg P_5(1)) \wedge \neg(P_6(1) \wedge P_7(2))$$

Transform F' *to a NNF and then to CNF:*

$$F'' = P_1(0) \wedge P_2(0) \wedge (\neg P_3(0) \vee P_4(1)) \wedge (\neg P_5(1)) \wedge (\neg P_6(1) \vee \neg P_7(2))$$

Transform F'' to a clause set:

$$\mathcal{C} = \{P_1(0) \vdash; P_2(0) \vdash; P_3(0) \vdash P_4(1); P_5(1) \vdash; P_6(1), P_7(2)) \vdash\}$$

Create the set of equivalences (";" is the separator instead of ","):

$$\mathcal{C}' = \{P_1(0) \vdash P_2(0);\ P_2(0) \vdash P_1(0);\ P_2(0) \vdash P_3(0);\ P_3(0) \vdash P_2(0);$$

$$P_4(1) \vdash P_5(1);\ P_5(1) \vdash P_4(1);\ P_5(1) \vdash P_6(1);\ P_6(1) \vdash P_5(1)\}$$

The set $\mathcal{C} \cup \mathcal{C}'$ is given to theorem prover and the following refutation is returned:
$\{P_2(0),\ \neg P_2(0) \vee P_3(0),\ P_3(0),\ \neg P_3(0) \vee P_4(1),$

$P_4(1),\ \neg P_4(1) \vee P_5(1),\ P_5(1), \neg P_5(1),\ \square\}$

Hence, we mark the following atom formulas in F':

$$F = P_1(0) \wedge P_2^{\bigstar}(0) \wedge (P_3^{\bigstar}(0) \rightarrow P_4^{\bigstar}(1)) \wedge (\neg P_5^{\bigstar}(1)) \wedge \neg(P_6(1) \wedge P_7(2))$$

The next step is to unlabel the formulas, negate it and transform it back to a sequent :

$$S = P(0), P^{\bigstar}(0), P^{\bigstar}(0) \rightarrow P^{\bigstar}(1) \vdash P^{\bigstar}(1), P(1) \wedge P(2)$$

The last step is to delete the formula occurrences which can be deleted :

$$P^{\bigstar}(0), P^{\bigstar}(0) \rightarrow P^{\bigstar}(1) \vdash P^{\bigstar}(1)$$

Example 2.2. *This example shows the simplification of a Herbrand sequent in formula level as well as in term level. The signature Σ consists of one constant symbol 0 interpreted as a $0 \in \mathbb{N}$, a unary function symbol s interpreted as a succesor function over $\mathbb{N}$, and two binary function symbols $+$ and $*$ interpreted as a sum and multiplication operation over $\mathbb{N}$ respectively. The sequent is :*

$S : A_1, A_2, A_3, A_4 \vdash C_1, C_2$, *where*

$A_1 : P((s(0) + s(0)) + s(0))$

$A_2 : P(s(0) + s(s(s(0) + s(0))))$
$A_3 : P(s(0) + s(s(s(0) + s(0)))) \rightarrow P(s(s(0)) * s(s(0) + s(0)))$
$A_4 : P(s(s(0) + s(0)) * s(s(0) + 0)) \rightarrow P(s(s(s(0)) * s(s(s(0)))))$
$C_1 : P(s(s(0)) * s(S(0)))$
$C_2 : P(s(s(s(s(0))) * s(s(0))))$

In this case the term-rewriting system consists of the following rules:

- $x + 0 \rightarrow x$
- $x * 0 \rightarrow 0$
- $x + s(y) \rightarrow s(x + y)$
- $x * s(y) \rightarrow x * y + x$

The sequent can be taught of as $P(3), P(4), P(4) \rightarrow P(5), P(5) \rightarrow P(7) \vdash P(4), P(7)$. *The idea is to show that* $P(3)$ *from the antecedent part and* $P(4)$ *from the consequent part are irrelevant for the validity of the sequent.*

This background theory can be turned into a Term-rewriting system. Furthermore, this term-rewriting system is confluent and terminating. Once we have all terms in normal form, we do not need the background theory anymore. We orient the equation from left to right and add them in the data structure which is a list of ***TermRewritingRules****. Then we call the operation* ***class OpSimplifyingSequentInTermLevel*** *which apply the rewrite rules till no further reduction is possible. The result is a sequent* $S_{trw} : A'_1, A'_2, A'_3, A'_4 \vdash C'_1, C'_2$, *where*

$A'_1 : P(s^3(0))$
$A'_2 : P(s^5(0))$
$A'_3 : P(s^5(0)) \rightarrow P(s^6(0))$
$A'_4 : P(s^6(0)) \rightarrow P(s^7(0))$
$C'_1 : P(s^4(0))$
$C'_2 : P(s^7(0))$

Now we transform the formulas in Conjunctive-normal forma and label the atom formulas. After that, we rename the sequent in such a way that we encode the labels of the atom formulas into the names of the same atom formulas calling the operation ***OpRenameFormaulasToLabelsInSequent****.*

The result is the sequent:

$$
\begin{array}{l}
P_1(s^3(0)),\\
P_2(s^5(0)),\\
\neg P_3(s^5(0)) \vee P_4(s^6(0)),\\
\neg P_5(s^6(0)) \vee P_6(s^7(0))\\
\vdash\\
P_7(s^4(0)),\\
P_8(s^7(0))
\end{array}
$$

Now we negate the sequent and transform in to a set of clauses calling the ***transformNegatedSequentToClauseSet*** *operation. The result is the following set of clauses:*

$$\{\vdash P_1(s^3(0)), \vdash P_2(s^5(0)), P_3(s^5(0)) \vdash P_4(s^6(0)), P_5(s^6(0)) \vdash P_6(s^7(0)),$$
$$P_7(s^4(0)) \vdash, P_8(s^7(0)) \vdash\}$$

Since the formula pairs $P_2(s^5(0))$ *and* $P_3(s^5(0))$, $P_4(s^6(0))$ *and* $P_5(s^6(0))$ *and* $P_6(s^7(0))$ *and* $P_8(s^7(0))$ *are renamed version of the formulas* $P(s^5(0))$, $P(s^6(0))$ *and* $P(s^7(0))$ *respectively, we construct the union of the set of clauses above with the following set of clauses representing the equivalence of the formulas in each pair:*

$$\{P_2(s^5(0)) \vdash P_3(s^5(0)), P_3(s^5(0)) \vdash P_2(s^5(0)), P_4(s^6(0)) \vdash P_5(s^6(0)),$$
$$P_5(s^6(0)) \vdash P_4(s^6(0)), P_6(s^7(0)) \vdash P_8(s^7(0)), P_8(s^7(0)) \vdash P_6(s^7(0))\}$$

Now we are ready to give the new set of clauses as an input to the theorem prover. The outcome is a refutation tree from which we collect all the axioms. The marked sequent is:

$$
\begin{array}{l}
P(s^3(0)), P^\star(s^5(0)), P^\star(s^5(0)) \rightarrow P^\star(s^6(0)), P^\star(s^6(0)) \rightarrow P^\star(s^7(0))\\
\vdash\\
P(s^4(0)), P^\star(s^7(0))
\end{array}
$$

The unmarked formulas $P(s^3(0))$ *and* $P(s^4(0))$ *correspond to formula occurrence that can be dropped. Hence, the simplified Herbrand sequent is:*

$$P^\star(s^5(0)), P^\star(s^5(0)) \rightarrow P^\star(s^6(0)), P^\star(s^6(0)) \rightarrow P^\star(s^7(0)) \vdash P^\star(s^7(0))$$

3 Specification of the term-rewriting system

The term-rewriting system is specified by the user and should be a confluent and terminating one. The syntax is as follows:

$Var := x|y|z$

$Const := s1|s2|s3|s4$

$Term := Var|Const|f(Term_1, \ldots, Term_n)$

$trsRule := Term -> Term$

The term-rewriting system is stored as a string in a file. The string is a sequence of rules separated by comma.
For example: $+(x, 0)\text{-}> 0, +(x, s(y))\text{-}> s(+(x, y)), *(x, s(y))\text{-}> +(*(x, y), x)$