An Improvement on Sub-Herbrand Universe Computation

Lifeng He[*,1,§], Yuyan Chao[2,§], Kenji Suzuki[3], Zhenghao Shi[3] and Hidenori Itoh[4]

[1]*Graduate School of Information Science, Aichi Prefectural University, Aichi, Japan*

[2]*Graduate School of Environment Management, Nagoya Sangyo University, Aichi, Japan*

[3]*Department of Radiology, The University of Chicago, Chicago, IL, USA*

[4]*Department of Information, Nagoya Institute of Technology, Nagoya, Japan*

Abstract: This paper presents an efficient algorithm for computing sub-Herbrand universes for arguments of functions and predicates in a given clause set. Unlike the previous algorithm, which processes all clauses in the given clause set once for computing each sub-Herbrand universe, the proposed algorithm computes all sub-Herbrand universes in the clause set by processing each clause in the clause set only once. We prove the correctness of our algorithm, and we provide experimental results on theorem proving benchmark problems to show the power of our approach.

INTRODUCTION

Herbrand's theorem [1] is the basis for most modern automatic proof procedures in automated first-order theorem proving. By Herbrand's theorem, for a given clause set S, a special universe, called *Herbrand universe*, can be created automatically. S is unsatisfiable if and only if there is an unsatisfiable set of ground instances of clauses of S, where a ground instance of a clause is derived by instantiating variables in the clause with elements of the Herbrand universe of S. Herbrand's theorem enables us to make theorem proving mechanical. However, theorem proving methods based directly on Herbrand's theorem, e.g., the multiplication method [2], are usually inefficient, because there may be too many ground instances that need to be considered.

Addressing to this problem, He *et al.* [3] proposed a method for computing a sub-universe of the Herbrand universe, denoted a *sub-Herbrand universe*, for each argument of predicates or functions in a given clause set S, and they proved that S is unsatisfiable if and only if there is a finite unsatisfiable set of ground instances of clauses of S derived by instantiating each variable, which appears as an argument of predicate symbols or function symbols, in S over its corresponding sub-Herbrand universes. Because such sub-universes are usually smaller (sometimes considerably so) than the Herbrand universe of S, the number of ground instances that need to be considered for reasoning can be reduced in many cases. Their experimental results demonstrated that this improvement is efficient for model generation theorem proving approach [4-6].

However, the algorithm proposed in [3] is inefficient. In order to compute a sub-Herbrand universe corresponding to an argument in a given clause set S, it has to processes all clauses in S once. For a large clause set, it takes more than hours to finish the computation.

This paper presents an efficient algorithm for computing sub-Herbrand universes in a clause set. Unlike the previous algorithm mentioned above, the proposed algorithm computes all sub-Herbrand universes in a given clause set S by processing each clause in S only once. The experimental results on theorem proving benchmark problems demonstrate that the proposed algorithm is much efficient than the previous one.

The rest of the paper is organized as follows: We review the previous algorithm proposed in [3] in the next section, then introduce our efficient algorithm in Section 3. Section 4 shows the correctness of our approach, and Section 5 reports the experimental results on benchmarks. Lastly, we give our conclusion in Section 6.

REVIEW OF THE PREVIOUS ALGORITHM

In this paper, the lower-case letters are used to represent predicate symbols, function symbols and constants, while the upper-case letters are used for atoms and variables. On the other hand, the Greek letters are used to represent arbitrary predicate symbols, function symbols, terms, substitutions, and other necessary information. A predicate (function) α with n arguments is called n-place predicate (function), and the ith ($1 \leq i \leq n$) argument of α is denoted to $\alpha\langle i \rangle$. Φ is denoted the empty set, $A \in I$ means that A is a member of I. Moreover, we view clauses as sets and assume that there is no same variable symbol in different clauses of a given clause set.

Let S be a set of clauses. Similar as in [3], for convenience, we use *SHU* to denote *sub-Herbrand Universe*, and *app* (τ, $\alpha\langle i \rangle$) to denote that a term τ appears as a value of an argument $\alpha\langle i \rangle$ in S. For example, suppose if there is $p(f(a), X)$ in S, then we have $app(a, f\langle 1 \rangle)$, $app(f(a), p\langle 1 \rangle)$, and $app(X, p\langle 2 \rangle)$.

*Address correspondence to this author at the Graduate School of Information Science, Aichi Prefectural University, Nagakute-cho, Aichi-gun, Aichi 480-1198, Japan; Tel: +81-561-64-1111, Ext. 3311; Fax: +81-561-64-1108; E-mail: helifeng@ist.aichi-pu.ac.jp

§Also with Shannxi University of Science and Technology, Xianyang, Shannxi, China.

1874-0618/07

Definition 1 (same domain argument). Arguments that hold the same *SHU* are called the *same domain arguments* (*SD arguments*).

Algorithm 1 (Algorithm for computing a preliminary *SHU*). Let *S* be a set of clauses, and $\alpha\langle i\rangle$, an argument in *S*. The *selected constant c* is a constant arbitrarily selected from *S* if there is any, otherwise, it is an artificial constant.

The preliminary *SHU* for $\alpha\langle i\rangle$ is a set *H* derived as follows:

1. Initially, set $H = \Phi$, $M = \{\alpha\langle i\rangle\}$, and $N = \Phi$.
2. If *M* is empty, then *H* is the preliminary *SHU* for $\alpha\langle i\rangle$, and all arguments in *N* are *SD* arguments. However, if *H* contains no constant, then $H = H \cup \{c\}$. On the other hand, if *M* is not empty, continue.
3. Move the first element $\beta\langle j\rangle$ of *M* to *N*. For each $app(\sigma, \beta\langle j\rangle)$ in each clause *C* of *S*:
   (1) if σ is a constant *c*. Let $H = H \cup \{c\}$;
   (2) if σ is functional term with function symbol *f*. Let $H = H \cup V\langle f\rangle$, where $V\langle f\rangle$ is the *possible value set* corresponding to *f*, whose definition will be given later;
   (3) if σ is a variable *X*. For each argument $\gamma\langle k\rangle$ such that there exists $app(X, \gamma\langle k\rangle)$ in *C*, add the argument $\gamma\langle k\rangle$ into *M* if $\gamma\langle k\rangle \notin M \cup N$.
4. Go to Step 2.

When the above algorithm terminated, all argument in *N* are *SD* arguments.

Algorithm 2 (Algorithm for deriving all preliminary *SHU*s). Let *S* be a clause set. All preliminary *SHU*s for the arguments of predicate symbols and function symbols in *S* can be established as follows:

1. Let *T* be the set of all arguments of predicate symbols and function symbols in *S*, $j = 0$.
2. If *T* is empty, terminate; $H_1, \ldots, H_j$, are the derived preliminary *SHU*s. All arguments in N_k $(1 \le k \le j)$ are *SD* arguments and have the same *SHU* H_k. Otherwise, if *T* is not empty, continue.
3. Let $\alpha\langle i\rangle$ be the first element of *T*, $j = j+1$. According to *Algorithm 1*, derive the preliminary *SHU* H_j for the argument $\alpha\langle i\rangle$ and the set N_j of the same *SHU* arguments of $\alpha\langle i\rangle$. Remove all elements of N_j from *T*, and go to Step 2.

All *SHU*s of the arguments of predicate symbols and function symbols in a set of clauses in the form of the Herbrand universe can be generated as follows:

Algorithm 3 (Algorithm for deriving *SHU*s in the form of the Herbrand universe).

Let *S* be a clause set, and *f1*, …, *fm*, all function symbols in *S*, and $H_1, \ldots, H_n$, the preliminary *SHU*s of arguments of predicate symbols and function symbols in *S* derived according to *Algorithm 2*. For each *i* such that $1 \le i \le n$, let C_i be the set of constants that appear in H_i, let $H_i^*(0) = C_i$, and for each *j* such that $1 \le j \le m$, $V^*(f_j, 0) = \Phi$.

Suppose that for $1 \le j \le m$, f_j is an h_j-place function symbol, and the *SHU* for the argument $f_j\langle t\rangle$ $(1 \le t \le h_j)$ is H_{u_t}, where $1 \le u_t \le n$. For $k = 0, 1, 2, \ldots$, let $V^*(f_j, k+1) = \{ f_j(\alpha_1, \cdots, \alpha_{h_j}) \mid \alpha_1 \in H_{u_1}^*(k), \ldots, \alpha_{h_j} \in H_{u_{h_j}}^*(k)\}$, and $H_i^*(k+1) = H_i^*(k) \cup \{V^*(f_j, k+1) \mid V\langle f_j\rangle \in H_i\}$.

Then, $V^*(f_j, \infty)$ is the set of the *possible values* of f_j, and $H_i^*(\infty)$ is the form of the Herbrand universe of H_i.

The above algorithm for computing *SHU*s in a give clause set *S* is not efficient. According to *Algorithm 1*, to computer an *SHU* corresponding to an argument $\alpha\langle i\rangle$, it processes all clauses in *S* once, and at that time, it matches $\alpha\langle i\rangle$ with every argument in *S*. For this reason, when the number of arguments in *S* and the number of *SHU*s in *S* are large, it will take too long time to finish the computation. For example, there are many problems in TPTP library (http://www.cs.miami.edu/~tptp/), a theorem proving benchmark problem library, such that the running times for their *SHU*s computation exceed 300 *sec*, which is the limitation time of general theorem proving contests (see http://www.cs.miami.edu/~tptp/CASC/).

THE PROPOSED ALGORITHM

We present an efficient algorithm that computes all *SHU*s in a given clause set *S* by processing each clause in *S* only once.

Definition 2 (Arguments corresponding to a variable). Let *C* be a clause in a set *S* of clauses. Every occurrence of a variable *X* in *C* is an argument of a predicate or a function in *C*. Such arguments are called the arguments corresponding to variable *X*.

For example, in clause $\neg p(X, f(Y)) \vee q(X) \vee r(Y)$, the arguments corresponding to variable *X* are $p\langle 1\rangle$ and $q\langle 1\rangle$, and those for variable *Y* are $f\langle 1\rangle$ and $r\langle 1\rangle$.

Lemma 1. All arguments corresponding to a variable in a clause are *SD* arguments.

Because all arguments corresponding to a variable in a clause are certainly substituted by the same term during reasoning, *Lemma 1* is obviously true.

Lemma 2. Let D_1 and D_2 be two sets of *SD* arguments. If D_1 and D_2 contain a common argument, then all arguments in D_1 and D_2 are *SD* arguments.

According to *Definition 1*, the proof of *Lemma 2* is trivial. Obviously, if D_1 and D_2 contain the same argument, then they can be combined.

Algorithm 4 (Algorithm for computing the sets of *SD* arguments in a clause). Let *C* be a clause. The sets of *SD* arguments in *C* can be derived as follows:

1. Let $X_1, \ldots, X_n$ be the variables in *C*. For each X_i ($1 \le i \le n$), let $D_1, \ldots, D_m$ be the candidate sets of *SD* arguments that have been established (initially none), and *D* the set of *SD* arguments corresponding to X_i. Suppose that $D_{j_1}, \ldots, D_{j_u}$ ($1 \le j_t \le m, 1 \le t \le u$) are the sets of *SD* arguments such that there is a common argument in *D* and D_{j_t}. Remove $D_{j_1}, \ldots, D_{j_u}$, and add $D \cup D_{j_1} \cup \ldots \cup D_{j_u}$ as a new candidate set of *SD* arguments. However, if there is no such argument, just add *D* as a new candidate set of *SD* arguments.
2. For each of constants and functions in *C*, it is an occurrence of an argument, say, α, in *C*. If α does not belong to any of the candidate sets of *SD* arguments that have been derived, then add the set $\{\alpha\}$ as a new candidate set of *SD* arguments in *C*.

After all variables, constants, and functions have been processed, the derived candidate sets of *SD* arguments are the sets of *SD* arguments in *C*.

Algorithm 5 (Algorithm for computing the sets of *SD* arguments in a clause set). Let *S* be a clause set, $C_1, \ldots, C_n$, the clauses in *S*. All sets of *SD* arguments in *S* can be derived by processing C_i ($1 \le i \le n$) one by one as follows:

Let $D_1, \ldots, D_m$ be the candidate sets of *SD* arguments being established (initially none), and $F_1, \ldots, F_t$, the sets of *SD* arguments in C_i. For each F_j ($1 \le j \le t$), let $D_{p_1}, \ldots, D_{p_u}$ ($1 \le p_k \le t, 1 \le k \le u$) be the sets of *SD* arguments such that there is a common argument in F_j and D_{p_k}. Remove $D_{p_1}, \ldots, D_{p_u}$, and add $F_j \cup D_{p_1} \cup \ldots \cup D_{p_u}$ as a new candidate set of *SD* arguments. If there is no such argument, simply add F_j as a new candidate set of *SD* arguments.

After all clauses are processed, the derived candidate sets of *SD* arguments are the sets of *SD* arguments in *S*.

Because there are only finite arguments of predicates and functions as well as finite clauses in a set of clauses, the above algorithm certainly terminates finitely. Moreover, because all sets of *SD* arguments that contain a common argument are combined whenever they are found, when the above algorithm terminates, each argument in the given clause set belongs and only belongs to a set of *SD* arguments.

Example 1. Let *S* be the following clause set:

$p_1(c)$.

$p_2(f(c))$.

$p_1(X) \vee p_2(X)$.

By the algorithm given in *Algorithm 4*, for clause $\neg p_1(c)$, we can derive a set of *SD* arguments: $D_1 = \{ p_1\langle 1\rangle \}$; for clause $\neg p_2(f(c))$, we can derive two sets of *SD* arguments: $D_2 = \{ p_2\langle 1\rangle \}$, and $D_3 = \{ f\langle 1\rangle \}$; and for clause $p_1(X) \vee p_2(X)$, we can derive a set of *SD* arguments: $D_4 = \{ p_1\langle 1\rangle, p_2\langle 1\rangle \}$.

By the algorithm given in *Algorithm 5*, after processing clause $\neg p_1(c)$, we have one candidate set of *SD* arguments $D_1 = \{ p_1\langle 1\rangle \}$; after processing clause $\neg p_2(f(c))$, we have three candidate set of *SD* arguments $D_1 = \{ p_1\langle 1\rangle \}$, $D_2 = \{ p_2\langle 1\rangle \}$ and $D_3 = \{ f\langle 1\rangle \}$. When processing the clause $p_1(X) \vee p_2(X)$, because there are $p_1\langle 1\rangle \in D_1$ and $p_1\langle 1\rangle \in D_4$, as well as $p_2\langle 1\rangle \in D_2$ and $p_2\langle 1\rangle \in D_4$, D_1, D_2 and D_4 are removed, and $D_5 = D_1 \cup D_2 \cup D_4 = \{ p_1\langle 1\rangle, p_2\langle 1\rangle \}$ is added.

As a result, we finally derive two sets of *SD* arguments in *S*, renamed as: $G_1 = \{ f\langle 1\rangle \}$, $G_2 = \{ p_1\langle 1\rangle, p_2\langle 1\rangle \}$.

Definition 3 (The constant set and function set corresponding to a set of *SD* arguments). Let *S* be a clause set, and *G* a set of *SD* arguments in *S*. The constant set *C* (function set *F*) corresponding to *G* is the set of constant *c* (function *f*) such that there is *app*(*c*, α) (*app*(*f*, α)) and $\alpha \in G$. However, if *C* is empty, let $C = \{a\}$, where *a* can be an arbitrary constant occurring in the Herbrand universe of *S*.

Example 2. Let *S* be the clause set, G_1 and G_2 the derived sets of *SD* arguments in *Example 1*.

By *Definition 3*, the constant set and function set corresponding to G_1 are $\{c\}$ and Φ, respectively, and those corresponding to G_2 are $\{c\}$ and $\{f\}$, respectively.

All *SHU*s (in the form of Herbrand Universe) of arguments of predicates and functions in a set of clauses can be generated as follows:

Algorithm 6 (Algorithm for computing *SHU*s in a clause set). Let *S* be a clause set. Let $G_1, \ldots, G_n$ be the sets of *SD* arguments in *S* derived by the algorithm given in *Algorithm 5*, C_i and F_i ($1 \le i \le n$), the constant set and function set corresponding to G_i, respectively.

For each *i* such that $1 \le i \le n$, let $H_i(0) = C_i$, and for each function *f* such that $f \in F_i$, let $V(f, 0) = \Phi$.

Suppose that *f* is an *h*-place function, and $f\langle t\rangle \in G_{u_t}$, where $1 \le t \le h$, $1 \le u_t \le n$. For $k = 0, 1, 2, \ldots$, let $V(f, k+1) = V(f, k) \cup \{f(\alpha_1, \ldots, \alpha_h) \mid \alpha_t \in H_{u_t}(k)\}$, and $H_i(k+1) = H_i(k) \cup \{V(f, k+1) \mid f \in F_i\}$.

Then $H_i(\infty)$ is the *SHU* (in the form of Herbrand Universe) for the arguments in G_i.

Example 3. Let *S* be the clause set given in *Example 1*. From *Example 1*, we have two sets of *SD* arguments: $G_1 =$

$\{ f\langle 1\rangle \}$, $G_2 = \{ p_1\langle 1\rangle , p_2\langle 1\rangle \}$. From *Example 2*, we have $C_1 = \{c\}$, $F_1 = \Phi$, $C_2 = \{c\}$, and $F_2 = \{f\}$.

By the algorithm given in *Algorithm 6*,

$V(f, 0) = \Phi$,

$H_1(0) = C_1 = \{c\}$,

$H_2(0) = C_2 = \{c\}$;

$V(f, 1) = \{f(c)\}$,

$H_1(1) = H_1(0) = \{c\}$,

$H_2(1) = H_2(0) \cup V(f, 0) = \{c, f(c)\}$;

$V(f, 2) = \{f(c)\}$,

$H_1(2) = H_1(1) = \{c\}$,

$H_2(2) = H_2(1) \cup V(f, 2) = \{c, f(c)\}$;

$\vdots$

$V(f, \infty) = \{f(c)\}$,

$H_1(\infty) = \{c\}$,

$H_2(\infty) = \{c, f(c)\}$.

That is, the *SHU* for $f\langle 1\rangle$ is $H_1(\infty)$, and that for $p_1\langle 1\rangle$ and $p_2\langle 1\rangle$ is $H_2(\infty)$.

CORRECTNESS

For convenience, similarly in [3], we use $D\langle\alpha\rangle$ to denote the set of *SD* arguments that contains argument α, $H[\, D\langle\alpha\rangle\,]$ the *SHU* for the arguments in $D\langle\alpha\rangle$, and $C[\, D\langle\alpha\rangle\,]$ and $F[\, D\langle\alpha\rangle\,]$, the constant set and the function set corresponding to $D\langle\alpha\rangle$, respectively.

Definition 4 (*SHU* ground instance). Let *S* be a set of clauses, and *C* a clause in *S*. An *SHU ground instance* of *C* is a clause obtained by replacing each variable *X* in *C* by a member of the *SHU* for the arguments corresponding to *X*.

Lemma 3. Any *SHU* ground instance of a clause *C* is a ground instance of *C*.

Proof. According to the algorithm given in *Algorithm 6* and *Definition 3*, only the constants and functions occurring in *S* are used for generating *SHU*s, therefore, any *SHU* is a subset of Herbrand universe of *S*, and then an S*HU* ground instance of a clause *C* is a ground instance of *C* (but the converse is not always true).

Definition 5 (Depth of a ground term). Let τ be a ground term. The depth of τ, denoted by $dep\langle\tau\rangle$, is defined as follows:

1. $dep\langle\tau\rangle = 1$ if τ is a constant;
2. $\mathrm{dep}\langle f(\beta_1,\ldots,\beta_n)\rangle = h+1$, where h is the maximum value among $\mathrm{dep}\langle\beta_1\rangle, \ldots, \mathrm{dep}\langle\beta_n\rangle$.

For example, $\mathrm{dep}\langle f(a,b)\rangle = 2$, $\mathrm{dep}\langle f(a,g(b,c)\rangle = 3$, and $\mathrm{dep}\langle f(a,g(b,h(c)))\rangle = 4$.

Algorithm 7 (Algorithm for driving an unsatisfiable set of ground instances). Let *S* be an unsatisfiable set of clauses. Then the empty clause can be derived from *S* by resolution. We can obtain an unsatisfiable set of ground instances of clauses of *S* by recording the clauses used in resolution as follows:

1. When deriving a factor $C\sigma$ of a clause *C*, instead of deleting all repeated literals from $C\sigma$, we underline each of them;
2. When deriving a resolvent, instead of deleting the two literals resolved up, we underline each of them.

The underlined literals will not be used in further resolution. However, they are instantiated by substitutions used in resolution. If we ignore underlines, a resolvent can be considered as a disjunction of instances of the clauses in *S*. A clause with all literals underlined corresponds to the empty clause. When such a clause, called an *extended empty clause*, is derived, for each variable *X* that remained in the clause (if any), let α be an argument corresponding to *X*. We substitute X with a constant in $C[\, D\langle\alpha\rangle\,]$. Let *E* be the resulting clause. If we ignore all underlines, *E* is a disjunction of ground instances the clauses in the given clause set. Let S_E be the set of such ground instances. Then S_E is an unsatisfiable set of ground instances of clauses of *S*.

Example 4. Let *S* be the following unsatisfiable set of clauses:

$$p(f(a, X_1), X_2) \tag{1}$$

$$p(a, X_3) \vee p(X_4, X_3) \tag{2}$$

$$p(X_5, X_6) \vee p(f(X_5, b), X_6) \tag{3}$$

By the algorithm given in *Algorithm 7*, the empty clause can be derived as follows:

i) from clause (2), by we can derive a factor, $p(a, X_3) \vee \underline{p(a, X_3)}$ (4)

ii) by resolve (4) and (3), we have, $\underline{p(a, X_3)} \vee \underline{p(a, X_3)} \vee \neg \underline{p(a, X_3)} \vee p(f(a,b), X_3)$ (5)

iii) by resolve (5) and (1), we have, $\underline{p(a, X_3)} \vee \underline{p(a, X_3)} \vee \neg \underline{p(a, X_3)} \vee \underline{p(f(a,b), X_3)} \vee \underline{\neg p(f(a,b), X_3)}$ (6)

Because all literals in clause (6) are underlined, clause (6) is an extended empty clause. By *Definition 3*, because there is not any constant *c* such that there is $app(c, p\langle 2\rangle)$ in *S*, we take $C[D\langle p\langle 2\rangle\rangle] = \{a\}$ (Of course, we could also take $C[D\langle p\langle 2\rangle\rangle] = \{b\}$). Substituting all variable X_3 in clause (6) with constant *a*, we can obtain clause *E*:

$$p(a,a) \vee p(a,a) \vee \neg p(a,a) \vee p(f(a,b),a) \vee \neg p(f(a,b),a) \quad (7)$$

Then the unsatisfiable set S_E of ground instances of clauses of *S* derived from clause (7) is:

$\neg p(f(a,b),a)$ …… a ground instance of clause (1)

$p(a,a) \vee p(a,a)$ …… a ground instance of clause (2)

$\neg p(a,a) \vee p(f(a,b),a)$ …… a ground instance of clause (3)

Lemma 4. Let *S* be a set of clauses, *T* a factor or a resolvent derived in resolution on *S*, and *X* a variable in *T*. Then all arguments corresponding to *X* in *T* are *SD* arguments.

Proof. We prove *Lemma 4* by induction on the following statement: *I*(*n*): Suppose that T_n is the clause derived in the *n*-th step in resolution. Then all arguments corresponding to a variable *X* in T_n are *SD* arguments.

Base case: Show *I*(0). T_0 is a clause in *S*. According to *Lemma 1*, all arguments corresponding to a variable *X* in T_0 are *SD* arguments.

Induction step: Suppose that *I*(0), …, *I*(*n*); to show that *I*(*n*+1). T_{n+1} is a factor of a clause *C* (a resolvent of two clauses C_1 and C_2), where *C* (each of C_1 and C_2) is a clause derived before *I*(*n*+1).

Because *X* is a variable in T_{n+1}, *X* is certainly a variable in *C* (*C*', where *C*' is either C_1 or C_2). For the appearances $app(X, \alpha_1\langle i_1\rangle), \ldots, app(X, \alpha_r\langle i_r\rangle)$ in T_{n+1} such that there are also $app(X, \alpha_1\langle i_1\rangle), \ldots, app(X, \alpha_r\langle i_r\rangle)$ in *C* (*C*'), by the induction assumption, $\alpha_1\langle i_1\rangle), \ldots, \alpha_r\langle i_r\rangle$ are *SD* arguments.

The remaining appearances of *X* in T_{n+1} are generated by substituting other variables, say, $Y_1, \ldots, Y_t$, in *C* (C_1 or C_2) with *X*. For each Y_k ($1 \le k \le t$), there is a sequence of $app(X, \beta_1\langle j_1\rangle)$, $app(Y_{s_1}, \beta_1\langle j_1\rangle)$, $app(Y_{s_1}, \beta_2\langle j_2\rangle)$, $app(Y_{s_2}, \beta_2\langle j_2\rangle)$, $app(Y_{s_2}, \beta_3\langle j_3\rangle)$, …, $app(Y_{s_u}, \beta_u\langle j_u\rangle)$, $app(Y_k, \beta_u\langle j_u\rangle)$, where $1 \le s_v \le t$, $s_v \ne k$, $1 \le v \le u$, and $\beta_l\langle j_l\rangle$ ($1 \le l \le u$) is an argument in *C* (C_1 and/or C_2).

Let *D*, $D_{s_1}, \ldots, D_{s_u}$, and D_k be the sets of *SD* arguments corresponding to the variables *X*, $Y_{s_1}, \ldots, Y_{s_u}$, and Y_k, respectively. Then *D* and D_{s_1} contain the common argument $\beta_1\langle j_1\rangle$, D_{s_1} and D_{s_2} contain the common argument $\beta_2\langle j_2\rangle$, …, D_{s_u} and D_k contain the common argument $\beta_u\langle j_u\rangle$. By *Lemma 2*, all of arguments in *D*, $D_{s_1}, \ldots, D_{s_u}$, and D_k are *SD* arguments.

Therefore, all arguments corresponding to variable *X* in T_{n+1} are *SD* arguments, and *I*(*n*+1) is true.

For example, let C_1 be clause $p(X, Y) \vee q(Y, Z)$, and C_2, $\neg p(U, U)$. Then, the resolvent of C_1 and C_2 is $\underline{p(X, X)} \vee q(X, Z) \vee \underline{\neg p(X, X)}$. For variable *Y* in C_2, which is substituted to *X* in resolution, there is a sequence $app(X, p\langle 1\rangle)$, $app(U, p\langle 1\rangle)$, $app(U, p\langle 2\rangle)$, and $app(Y, p\langle 2\rangle)$. By *Algorithm 4*, the set of arguments corresponding to variables *X*, *Y* and *U* are $D_X = \{p\langle 1\rangle\}$, $D_Y = \{p\langle 2\rangle, q\langle 1\rangle\}$, and $D_U = \{p\langle 1\rangle, p\langle 2\rangle\}$, respectively. By *Lemma 2*, all arguments in D_X, D_Y, and D_U are *SD* arguments. The set of all such arguments is just the set of the arguments corresponding to variable *X* in the resolvent.

Lemma 5. Let *S* be a clause set, f (*c*) a function (constant) in *S*. Then, if there is a$app(f, \alpha)$ ($app(c, \beta)$) in our proposed resolution, there is $f \in F[D\langle\alpha\rangle]$ ($c \in C[D\langle\beta\rangle]$).

Proof. For any a$pp(f, \alpha)$ in *S*, by *Definition 3*, $f \in F[D\langle\alpha\rangle]$, and for any $app(c, \beta)$) in *S*, by *Algorithm 6*, $c \in C[D\langle\beta\rangle]$.

If a variable *X* is substituted to a functional term with function *f* in resolution, then, by *Lemma 4*, all arguments corresponding to variables *X*, say, $\alpha_1, \ldots, \alpha_n$, are *SD* arguments. Moreover, for some argument α_i ($1 \le i \le n$), there are $app(Y, \alpha_i)$ and $app(f, \alpha_i)$ in *S*, where *Y* is either *X* itself in *S* or a variable substituted to *X* in the resolution. By *Definition 3*, $f \in F[D\langle\alpha_i\rangle]$. Because $D\langle\alpha_1\rangle = \cdots = D\langle\alpha_n\rangle$, we have $f \in F[D\langle\alpha_1\rangle], \ldots, f \in F[D\langle\alpha_n\rangle]$.

Now, consider the case where a variable *X* is substituted to a constant *c*. By **Lemma 4**, all arguments in the clause corresponding to variables *X*, say, $\beta_1, \ldots, \beta_m$, are *SD* arguments. If the substitution occurs in our proposed resolution, for some argument β_j ($1 \le j \le n$), there are $app(Y, \beta_j)$ in *S*, where *Y* is either *X* or a variable substituted to *X* in the resolution. By *Definition 3*, we have $c \in C[D\langle\beta_j\rangle]$. On the other hand, when a variable *X* in the extended empty clause derived by our proposed resolution is substituted to a constant *c*, by *Algorithm 68*, we also have $c \in C[D\langle\beta_k\rangle]$, where ($1 \le k \le m$). Because $D\langle\beta_1\rangle = \cdots = D\langle\beta_m\rangle$, for both cases, we have $c \in C[D\langle\beta_1\rangle]$, …, $c \in C[D\langle\beta_m\rangle]$. Therefore, *Lemma 5* is true.

Lemma 6. Let *S* be an unsatisfiable set of clauses, and S_E the unsatisfiable set of ground instances of clauses of *S* derived by the algorithm given in *Algorithm 7*. Then each clause of S_E is an *SHU* ground instance of *S*.

Proof. Because there is no variable in S_E, all terms in S_E are ground. We prove *Lemma 6* by showing the following statement: for each ground term τ, if there is $app(\tau, \gamma)$ in S_E, then $\tau \in H[D\langle\gamma\rangle]$. We do the proof by induction on $dep\langle\tau\rangle$.

Basically, $dep\langle\tau\rangle = 1$, τ is a constant. By *Lemma 5*, $c \in C[D\langle\gamma\rangle]$, by *Algorithm 6*, $\tau \in H[D\langle\gamma\rangle]$.

Assume that the above statement holds when $dep\langle\tau\rangle = i$, $1 \le i \le t$, show that it holds when $dep\langle\tau\rangle = t+1$.

Without loss of generality, suppose that $\tau = f(\delta_1, \ldots, \delta_u)$, where $dep\langle\delta_j\rangle \le t$, $1 \le j \le u$. By the induction hypothesis, $\delta_j \in H[D\langle j\rangle]$; By *Lemma 5*, $f \in F[D\langle\gamma\rangle]$; and by *Algorithm 6*, $f(\delta_1, \ldots, \delta_u) \in C[D\langle\gamma\rangle]$. That is, $\tau \in H[D\langle\gamma\rangle]$ holds for $dep\langle\tau\rangle = t+1$. Therefore, each clause in S_E is an *SHU* ground instance of some clause in *S*.

Theorem 1 (Correctness). A set *S* of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S^* of the *SHU* ground instances of clauses of *S*.

Proof. ($\Rightarrow$) Suppose that *S* is unsatisfiable. Then by the algorithm given in *Algorithm 7*, we can derive an unsatisfiable set S_E of ground instances of clauses of *S*. By *Lemma 6*, each clause of S_E is an *SHU* ground instance of some clause of *S*. Let $S^* = S_E$; then S^* is an unsatisfiable set of *SHU* ground instances of clauses of *S*. ($\Leftarrow$) Suppose that there is a finite unsatisfiable set S^* of *SHU* ground instances of clauses of *S*. By *Lemma 3*, each *SHU* ground instance clause of *S* is a ground instance clause of *S*, by Herbrand's theorem, *S* is unsatisfiable.

EXPERIMENTAL RESULTS

The reason that the previous algorithm is inefficient is that for calculating each *SHU* in a given clause set *S*, it processes all clauses in the given problem once. During processing, it matches the argument under considering to each argument in the problem. Suppose that there are *m SHU*s and *n* arguments in *S*, then the number of matching times will be $m \times n$. When *m* and *n* are large, it will take a long time to complete calculation of *SHU*s. On the other hand, the proposed algorithm is efficient since it calculates all *SHU*s in a problem by processing each clause in the problem only once. It calculates provisional *SHU*s for each clause independently, and then combines those provisional *SHU*s that are corresponding to *SD* arguments. Suppose that there are *k* clauses in *S*, then the average number of arguments in each clause will be n/k. Suppose further that there are averagely *h SHUs* in each clause, where $h \le m$, then the number of the argument match in the proposed algorithm will be $h \times n/k$. Since *h* is usually much smaller *m*, the number of matching times in the proposed algorithm is at least *k* times least than that in the previous algorithm.

There are 8013 problems in the TPTP library version 3.1.1. Among them, 4365 problems are non-range-restricted. The number of the problems from which 2 or more *SHUs* can be derived by our approach is 709. The maximum number of arguments in a problem is 1542050 (SYN826-1), and the average number of arguments in all problems is 26131. The maximum number of clauses in a problem is 2004 (SYN826-1), and the average number of clauses in all problems is 128.

We implemented the previous algorithm and the proposed algorithm in SCIS Prolog, and run them on all 709 problems on an Intel PentiumIII/980MHZ workstation, 512MB. The result of the two algorithms is the same for each problem. The numbers of transferred problems for the two algorithms in a limited time 2, 5, 10, 50, 100, 200, 300 seconds, respectively, are shown in Table **1**.

Table 1. Experimental Results: The Number of Problems Solved Within Various Limited Times

Running Time (*sec*)	≤ 2	≤ 5	≤ 10	≤ 50	≤ 100	≤ 200	≤ 300
The previous algorithm	404	579	609	634	642	646	654
The proposed algorithm	637	663	681	691	694	707	709

Table 2. Execution Time Versus the Number of Arguments

The Number of Arguments	**The Number of Problems**	**The Previous Algorithm (*sec*)**	**The Proposed Algorithm (*sec*)**
0 ~ 10000	642	3.66	1.43
10001 ~ 20000	8	21.45	2.07
20001 ~ 30000	5	455.64	2.43
30001 ~ 40000	0	—	—
40001 ~ 50000	9	4518.87	4.27
50001 ~ 60000	1	> 10000.00	6.56
60001 ~ 70000	10	> 10000.00	7.35
70001 ~ 80000	3	> 10000.00	8.42
80001 ~ 200000	5	> 10000.00	13.07
200001 ~ 500000	9	> 10000.00	31.18
500001 ~ 1000000	15	> 10000.00	147.46
1000001 ~ 1600000	2	> 10000.00	244.34

The execution time versus the number of arguments in a problem for the two algorithms is shown in Table **2**, where the time limitation is 10000 seconds. We can find that the previous algorithm could not give a result in the limited time when the number of arguments is larger than 50000, and the proposed algorithm can give results for all problems within 250 seconds.

CONCLUSION

In this paper, we proposed an efficient algorithm for calculating sub-Herbrand universes in a clause set. The experimental results on theorem proving benchmark problems demonstrated that the proposed algorithm is much efficient than the previous one.

However, we could not give the exactly complexity analysis for the two algorithms. It remains for future work.

ACKNOWLEDGEMENTS

This work was supported in part by The TOYOAKI Scholarship Foundation, and the Grants-in-Aid for Scientific Research, Japan Society for the Promotion of Science.

REFERENCES

[1] J. Herbrand, "Recherches sur la théorie de la demonstration", PhD thesis, University of Paris, 1930.

[2] P.C. Gilmore, "A proof method for quantification theory: Its justification and realizaéon", *IBM J. Res. Develop*., pp.28-35, 1960.

[3] Y. Chao, L. He, Z. Shi, T. Nakamura, K. Suzuki, and H. Itoh, "An Improvement of Herbrand Theorem and Its Application to Model Generation Theorem Proving", *J. Comp. Sci. Technol.*, Vol. 22, No. 4, pp.541-553, 2007.

[4] R. Manthey, and F. Bry, "SATCHMO: a theorem prover implemented in prolog", Proceedings of 9th Intl. Conf. on Automated Deduction, Argonne, Illinois, USA, 1988, pp. 415-434.

[5] L. He, "I-SATCHMO: an Improvement of SATCHMO", *J. Automated Reasoning*, 27, pp. 313-322, 2001.

[6] L. He, Y. Chao, T. Nakamura, and Itoh H, "R-SATCHMO: Refinements on I-SATCHMO", *J. Logic Comput.*, Vol. 14, pp.117-143, 2004.

Received: August 15, 2007 Revised: November 7, 2007 Accepted: November 12, 2007