PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

A New Algorithm for Subsumption Test

Byeong Man Kim*, Sang Ho Lee**, Seung Ryoul Maeng*, and Jung Wan Cho*

* Department of Computer Science & Center for Artificial Intelligence Research
Korea Advanced Institute of Science and Technology, Dae-Jeon, Korea

** Database Section
Electronics and Telecommunications Research Institute, Dae-Jeon, Korea

Abstract

To reduce the number of generated clauses in resolution-based deduction systems, subsumption has been around quite for a long time in the automated reasoning community. It is well-known that use of the subsumption sharply improves the effectiveness of theorem proving. However, subsumption tests can be very expensive because they should be applied repeatedly and are relatively slow. There have been several researches to overcome the expensiveness of subsumption. One of them is the s-link test based on the connection graph procedure. In the s-link test, it is essential to find a set of pairwise strongly compatible matching substitutions between literals in two clauses. This paper presents an improved algorithm of the s-link test with a new object, called *strongly compatible list*. By use of the strongly compatible lists and appropriate bit operations on them, the proposed algorithm reduces the possible combinations of matching substitutions between literals as well as improves the pairwise strongly compatible test itself. Two other subsumption algorithms and our algorithm are analyzed in terms of the estimated maximal number of string comparisons. Our analysis shows that the worst-case time complexity of our algorithm is much lower than the other algorithms.

1 Introduction

Logical Reasoning (or theorem proving) is the key to solving many puzzles, to solving problems in mathematics, to designing electronic circuits, to verifying programs, and to answering queries in deduction systems. Logical reasoning is a process of drawing conclusions that follows logically from the supplied facts. Since the first-order predicate logic is generally sufficient for logical reasoning and offers the advantage of being partially decidable, it is widely used in automated reasoning.

There have been a number of approaches to show that a formula is a logical consequence of a set of formulas. Notable among them is Robinson's resolution principle [Robinson 1965] which is very powerful and uses only one inference rule. Many refinements of the resolution principle based on graph have been proposed to increase the efficiency [Kowalski 1975, Sickel 1976, Andrew 1981, Bibel 1981, Kowalski 1979]. One of them is Kowalski's connection graph proof procedure [Kowalski 1975, Kowalski 1979] which has some distinct advantages over previous approaches based upon resolution.

1. Once an initial connection graph is constructed all information is present as to which literals are potentially resolvable so that no further search for unifiable complementary literals is needed.
2. Application of a deletion operation can result in further deletion operations, thus potentially leading to a snowball effect which reduces the graph rapidly. The probability of this effect rises with the number of deletion rules available.
3. The presence of the complete search space during connection graph proof procedure suggests the opportunity to use parallel evaluation strategies [Loganantharaj 1986,Loganantharaj 1987,Juang 1988] to improve the efficiency.

Various deletion strategies [Munch 1988,Gottlob and Leitsch 1985,Chang and Lee 1973] are suggested to reduce the number of clauses generated in theorem proving (automated reasoning). A very powerful deletion rule in resolution-based deduction systems is the subsumption [Eisinger 1981, Wos 1986]. The subsumption is used not only to discard a newly deduced clause when a copy already has been retained, but also to discard other types of unneeded information. The use of subsumption sharply improves the effectiveness of theorem proving, as illustrated by the benchmark problem, Sam's Lemma [Wos 1986].

However, the use of subsumption can be quite expensive because it must be repeated very often and is relatively slow [Wos 1986]. There have been two approaches for overcoming the expensiveness of subsumption. One is to reduce the number of necessary subsumption tests [Eisinger 1981], and the other is to improve the subsumption test itself [Gottlob and Leitsch 1985, Stillman 1973]. Eisinger [Eisinger 1981] proposes the s-link test which is based on the principal ideas of the connection graph proof procedure. His method provides an efficient preselection which singles out clauses D that do not possess the appropriate links to the clause C. Having preselected the candidates, we need to compose matching substitutions from literals in clause C to literals in clause D to find a matcher θ from C to D. In some cases many compositions are possible and hence the search for θ becomes quite expensive. Socher [Socher 1988] improves the search procedure by imposing restrictions on the possible matching substitutions.

In this paper we propose an improved s-link test with a new object, called *strongly compatible list*. By use of the strongly compatible lists and appropriate bit operations on them, the proposed algorithm reduces the possible combinations of matching substitutions between literals as well as improves the pairwise strongly compatible test itself. Two subsumption algorithms (Eisinger, Socher) and our algorithm are analyzed in terms of the estimated maximal number of string comparisons. Our analysis shows that the worst-case time complexity of our algorithm is much lower than the other algorithms.

In the next chapter, preliminary definitions and the s-link test are presented. A new subsumption algorithm based on strongly compatible lists and its related works and analysis are given in Chapter 3 and Chapter 4, respectively. In Chapter 5, our works are summarized.

2 Preliminaries

We assume that the readers are familiar with materials in [Chang and Lee 1973]. A variable starts with an upper case letter and a constant starts with a lower case letter.

Definition 2.1 A *substitution* σ is a mapping from variables to terms.

We represent a substitution σ with $s_i\sigma = t_i$ for each i $(1 \leq i \leq n)$ by the set of pairs $\{t_1/s_1, \cdots, t_n/s_n\}$, and represent the composition of substitution of σ and τ by $\sigma \bullet \tau$. For convenience, we denote $\sigma_1 \bullet \cdots \bullet \sigma_n$ by $\bullet_{i=1}^{n}\sigma_i$.

Definition 2.2 Two substitutions σ and τ are *strongly compatible*, if $\sigma \bullet \tau = \tau \bullet \sigma$.

Definition 2.3 Substitutions $\sigma_1, \cdots, \sigma_n$ are *pairwisestrongly compatible*, if any two substitutions $\sigma_i, \sigma_j \in \{\sigma_1, \cdots, \sigma_n\}$ are strongly compatible.

Definition 2.4 A *matching substitution* from a term (or a literal) s to a term (or a literal, respectively) t is a substitution μ such that $s\mu = t$.

Definition 2.5 $uni(C, l_i, D)$ is a set of all matching substitutions mapping a literal l_i in clause C onto some literal in clause D.

For example, given $C = \{p(X,Y), q(Y,c)\}$ and $D = \{p(a,b), p(b,a), q(a,c)\}$, we have $uni(C, p(X,Y), D) = \{\{a/X, b/Y\}, \{b/X, a/Y\}\}$ and $uni(C,\ q(Y,c),\ D) = \{\{a/Y\}\}$.

Definition 2.6 If there is a τ with $\theta = \sigma \bullet \tau$ for any other unifier θ for s and t, σ is a *most general unifier* (mgu) for s and t.

To reduce the search space in theorem proving, redundant clauses must be removed. The *redundant* clause means a clause whose removal does not affect the unsatisfiability. The redundant clause includes a tautology or a subsumed clause. The subsumption can be defined in two ways.

Definition 2.7 A clause C_1 *subsumes* another clause C_2 if C_1 logically implies C_2.

Definition 2.8 A clause C_1 *θ-subsumes* another clause C_2 if $|C_1| \leq |C_2|$ and there is a substitution θ such that $C_1\theta \subseteq C_2$.

It has been shown [Gottlob and Leitsch 1985, Loveland 1978] that these two definitions are not equivalent. If we use the first definition, then most of the resolution-based proof procedures are not complete because a clause always subsume its factors. In this paper we are concerned only with the θ-subsumption.

In order to perform a subsumption test on given two clauses, we must find a matcher θ such that $C\theta \subseteq D$. It is well known that finding such θ is NP-complete [Gottlob and Leitsch 1985] and the search for θ may become expensive. There have been some efforts to reduce the cost of finding a matcher θ [Gottlob and Leitsch 1985, Socher 1988, Chang and Lee 1973, Eisinger 1981, Stillman 1973]. One of them is the s-link test based on the connection graph procedure. The subsumption test based on the s-link is provided by the following theorem [Eisinger 1981]:

Theorem 2.1 Let $C = \{l_1, \ldots, l_n\}$ and D be clauses. Then C θ-subsumes D if and only if $|C| \leq |D|$ and there is an n-tuple $(\sigma_1, \ldots, \sigma_n) \in \times_{i=1}^{n} uni(C, l_i, D)$ such that all σ_i $(1 \leq i \leq n)$ are pairwise strongly compatible.

Example 2.1 *(of Theorem 2.1 [Socher 1988])* Given a set $\{C,\ D_1,\ D_2,\ D_3\}$ of clauses with $C = \{p(X,Y), q(Y,c)\}$, $D_1 = \{p(a,c), r(b,c)\}$, $D_2 = \{p(U,V), q(V,W)\}$ and $D_3 = \{p(a,b),\ p(b,a),\ q(a,c)\}$ one want to find out, which clauses are subsumed by C. D_1 can be excluded because the literal $q(Y,c)$ in C is not unifiable with any literal in D_1, that is, there is no s-link from $q(Y,c)$ to a literal in D_1. D_2 cannot be a candidate because $uni(C,\ q(Y,c),\ D_2) = \{\}$. For D_3 we obtain the two pairs $(\sigma_1,\ \tau)$ and $(\sigma_2,\ \tau)$, where $\sigma_1 = \{a/X,\ b/Y\}$, $\sigma_2 = \{b/X,\ a/Y\}$ and $\tau = \{a/Y\}$. From these two pairs only $(\sigma_2,\ \tau)$ is strongly compatible and thus C subsumes D_3. □

As shown in Example 2.1, in order to find clauses that are subsumed by a clause $C = \{l_1,\ \ldots,\ l_m\}$, first we have to preselect clauses that are connected to every literals in C by s-links of a connection graph. If D is such clause then each literal in C is unifiable with some literals in D. For such candidate D, we need to perform a pairwise strongly compatible test on all elements of $\times_{i=1}^{m} uni(C, l_i, D)$.

3 A New Subsumption Algorithm Based on Strongly Compatible Lists

The s-link test [Eisinger 1981] for long clauses with more than one matching substitution for each literal may require an expensive search of all elements of the Cartesian product.

We define *the strongly compatible list* of matching substitutions in order to improve the s-link test. With the strongly compatible lists, we can single out useless matching substitutions and improve the pairwise strongly compatible test itself.

The following three bit operations are used in this paper.

$w_1 + w_2$	:	bitwise disjunction of w_1 and w_2
$w_1 * w_2$	:	bitwise conjunction of w_1 and w_2
$\overline{w}$	:	bitwise complementation

where w_i is a bit sequence. For convenience, we denote $w_1 + \cdots + w_n$ by $+_{i=1}^{n} w_i$. Similarly, we denote $w_1 * \cdots * w_n$ by $*_{i=1}^{n} w_i$.

To test whether the given two matching substitutions are strongly compatible, we need the following definition.

Definition 3.1 Let $\{v_1, \cdots, v_n\}$ be an ordered set of variables in clause C, and a matching substitution σ between literals in clauses C and D be $\{t_1/s_1, \cdots, t_m/s_m\}$. $\delta(\sigma)$ is an n-length list such that the ith element is t_j if $v_i = s_j$, ϕ otherwise. $\delta(\sigma) = (t_1, \cdots, t_n)$ indicates that substitution σ does not substitute for variable v_i if t_i is ϕ, otherwise it substitutes t_i for v_i.

Example 3.1 Let $C = \{l_1, l_2\}$ and $D = \{k_1, k_2, k_3\}$ with $l_1 = p(X, Y)$, $l_2 = p(Y, Z)$, $k_1 = p(a, b)$, $k_2 = p(a, c)$, $k_3 = p(d, b)$ and $\{X, Y, Z\}$ be an ordered set of variables in C. We want to find all the matching substitutions between literals in C and literals in D. Then, we can obtain that $\sigma_1 = \{a/X, b/Y\}$, $\sigma_2 = \{a/X, c/Y\}$, $\sigma_3 = \{d/X, b/Y\}$, $\sigma_4 = \{a/Y, b/Z\}$, $\sigma_5 = \{a/Y, c/Z\}$, and $\sigma_6 = \{d/Y, b/Z\}$ for each (l_i, k_j) where $1 \le i \le 2$ and $1 \le j \le 3$. By Definition 3.1 we obtain that $\delta(\sigma_1) = (a, b, \phi)$, $\delta(\sigma_2) = (a, c, \phi)$, $\delta(\sigma_3) = (d, b, \phi)$, $\delta(\sigma_4) = (\phi, a, b)$, $\delta(\sigma_5) = (\phi, a, c)$, and $\delta(\sigma_6) = (\phi, d, b)$. □

If two matching substitution σ_1 and σ_2 are strongly compatible, they should not substitute for same variables differently. That is, if σ_1 substitutes a term t for a variable v then σ_2 has to substitute the term t for the variable v or does not have to substitute for the variable v. This can be formally described in Lemma 3.1.

Lemma 3.1 Let $\{v_1, \cdots, v_n\}$ be an ordered set of variables in clause C, and σ_1 and σ_2 be matching substitutions from literals in clause C to literals in clause D. σ_1 and σ_2 are strongly compatible if and only if $\pi_i(\delta(\sigma_1)) = \phi \vee \pi_i(\delta(\sigma_2)) = \phi \vee \pi_i(\delta(\sigma_1)) = \pi_i(\delta(\sigma_2))$ for each i $(1 \le i \le n)$, where $\pi_i(X)$ is a selection function which returns the ith element of list X.

(Proof) $(\leftarrow)$ Each case is considered separately.

(i) in the case $\pi_i(\delta(\sigma_1)) = \phi$
Since $v_i\sigma_1 = v_i$ and clauses C and D are variable-disjoint, $v_i(\sigma_1 \bullet \sigma_2) = (v_i\sigma_1)\sigma_2 = v_i\sigma_2 = (v_i\sigma_2)\sigma_1 = v_i(\sigma_2 \bullet \sigma_1)$.

(ii) in the case $\pi_i(\delta(\sigma_2)) = \phi$
Since $v_i\sigma_2 = v_i$ and clauses C and D variable-disjoint, $v_i(\sigma_2 \bullet \sigma_1) = (v_i\sigma_2)\sigma_1 = v_i\sigma_1 = (v_i\sigma_1)\sigma_2 = v_i(\sigma_1 \bullet \sigma_2)$.

(iii) in the case $\pi_i(\delta(\sigma_1)) = \pi_i(\delta(\sigma_2))$
Since $v_i\sigma_1 = v_i\sigma_2$ and clauses C and D are variable-disjoint, $v_i(\sigma_1 \bullet \sigma_2) = (v_i\sigma_1)\sigma_2 = (v_i\sigma_2)\sigma_2 = v_i\sigma_2 = v_i\sigma_1 = (v_i\sigma_1)\sigma_1 = (v_i\sigma_2)\sigma_1 = v_i(\sigma_2 \bullet \sigma_1)$.

From (i), (ii), and (iii), if $\pi_i(\delta(\sigma_1)) = \phi$ or $\pi_i(\delta(\sigma_2)) = \phi$ or $\pi_i(\delta(\sigma_1)) = \pi_i(\delta(\sigma_2))$ for each i $(1 \le i \le n)$, then σ_1 and σ_2 are strongly compatible.

$(\rightarrow)$ Assume that $\pi_i(\delta(\sigma_1)) \ne \phi$, $\pi_i(\delta(\sigma_2)) \ne \phi$, and $\pi_i(\delta(\sigma_1)) \ne \pi_i(\delta(\sigma_2))$. By Definition 3.1, σ_1 and σ_2 contain s_1/v_i and s_2/v_i, respectively, where $s_1 \ne s_2$. Hence, $v_i(\sigma_1 \bullet \sigma_2) \ne v_i(\sigma_2 \bullet \sigma_1)$ (i.e. $\sigma_1 \bullet \sigma_2 \ne \sigma_2 \bullet \sigma_1$). This is contradictory to that σ_1 and σ_2 are strongly compatible. The proof is completed. □

Lemma 3.1 suggests a new method for testing whether the given two matching substitutions σ_1 and σ_2 are strongly compatible. That is, without calculating $\sigma_1 \bullet \sigma_2$ and $\sigma_2 \bullet \sigma_1$, we can determine whether σ_1 and σ_2 are strongly compatible by only comparing $\delta(\sigma_1)$ with $\delta(\sigma_2)$. For example, we can know that σ_1 and σ_4 in Example 3.1 are not strongly compatible because $\pi_2(\delta(\sigma_1)) \ne \phi \wedge \pi_2(\delta(\sigma_4)) \ne \phi \wedge \pi_2(\delta(\sigma_1)) \ne \pi_2(\delta(\sigma_4))$.

Definition 3.2 Let $\{v_1, \cdots, v_m\}$ be an ordered set of variables in clause C, and let $\{\sigma_1, \cdots, \sigma_n\}$ be an ordered set of matching substitutions from literals in clause C to literals in clause D. $P_i(X)$, $1 \le i \le m$, is an n-bit sequence such that its j'th bit is 1 if the i'th element of $\delta(\sigma_j)$ is X or ϕ, otherwise 0 for each j $(1 \le j \le n)$. Especially, when X is ϕ all bits of $P_i(X)$ are 1.

Example 3.2 From Example 3.1, we have $P_1(a) = 110111$, $P_1(d) = 001111$, $P_1(\phi) = 111111$, $P_2(a) = 000110$, $P_2(b) = 101000$, $P_2(c) = 010000$, $P_2(d) = 000001$, $P_3(\phi) = 111111$, $P_3(b) = 111101$, and $P_3(c) = 111010$. In this case, $P_1(a) = 110111$ indicates that variable v_1 having value a is compatible with substitutions 1,2,4,5,6 but not with the substitution 3. □

Let $\{\sigma_1, \cdots, \sigma_n\}$ be an ordered set of matching substitutions from literals in clause C to literals in clause D and m be the number of variables in C. Matching substitutions which are strongly compatible with σ_i, $1 \le i \le n$, can be represented by an n-bit sequence which is calculated by the following function $\beta(\sigma_i)$:

$$\beta(\sigma_i) = *_{j=1}^{m} P_j(\pi_j(\delta(\sigma_i))).$$

We call $\beta(\sigma_i)$ the *strongly compatible list* for σ_i.

Lemma 3.2 Let $\{v_1, \cdots, v_m\}$ be an ordered set of variables in clause C, and $\{\sigma_1, \cdots, \sigma_n\}$ be an ordered set of matching substitutions between literals in clauses C and literals in clause D. $\sigma \in \{\sigma_1, \cdots, \sigma_n\}$ and σ_k, $1 \le k \le n$, are strongly compatible if and only if the k'th bit of $\beta(\sigma)$ is 1.

(Proof) We must show that if the k'th bit of $*_{i=1}^{m} P_i(\pi_i(\delta(\sigma)))$ is 1 then σ and σ_k are strongly compatible, and also show that if σ is strongly compatible with σ_k then the k'th bit of $*_{i=1}^{m} P_i(\pi_i(\delta(\sigma)))$ is 1.

$(\leftarrow)$ From the fact that the k'th bit of $*_{i=1}^{m} P_i(\pi_i(\delta(\sigma)))$ is 1, the k'th bit of $P_i(\pi_i(\delta(\sigma)))$ is 1 for each i $(1 \le i \le m)$. By Definition 3.2, $\pi_i(\delta(\sigma)) = \phi$ or $\pi_i(\delta(\sigma)) = \pi_i(\delta(\sigma_k))$ or $\pi_i(\delta(\sigma_k)) = \phi$ for each i $(1 \le i \le m)$. Therefore, by Lemma 3.1, σ and σ_k are strongly compatible.

$(\rightarrow)$ From Lemma 3.1, $\pi_i(\delta(\sigma)) = \phi$ or $\pi_i(\delta(\sigma)) = \pi_i(\delta(\sigma_k))$ or $\pi_i(\delta(\sigma_k)) = \phi$ for each i $(1 \le i \le m)$. By Definition 3.2, the k'th element of $P_i(\pi_i(\delta(\sigma)))$ is 1 for each i $(1 \le i \le m)$. Therefore, the k'th bit of $*_{i=1}^{m} P_i(\pi_i(\delta(\sigma)))$ is 1. □

Example 3.3 From Example 3.2, we can obtain $\beta(\sigma_i)$ for each i $(1 \le i \le 6)$ as follows:

$\beta(\sigma_1) = P_1(a) * P_2(b) * P_3(\phi) = 110111 * 101000 * 111111 = 100000$

$\beta(\sigma_2) = P_1(a) * P_2(c) * P_3(\phi) = 110111 * 010000 * 111111 = 010000$

$\beta(\sigma_3) = P_1(d) * P_2(b) * P_3(\phi) = 001111 * 101000 * 111111 = 001000$

$\beta(\sigma_4) = P_1(\phi) * P_2(a) * P_3(b) = 111111 * 000110 * 111101 = 000100$

$\beta(\sigma_5) = P_1(\phi) * P_2(a) * P_3(c) = 111111 * 000110$

$* \ 111010 = 000010$
$\beta(\sigma_6) = P_1(\phi) * P_2(d) * P_3(b) = 111111 * 000001 * 111101 = 000001.$
From this, we know that each $\sigma_i (1 \le i \le 6)$ is strongly compatible with only itself. □

Some matching substitutions do not contribute to construct a matcher θ such that $C\theta \subseteq D$. If such matching substitutions can be identified and removed before the actual pairwise strongly compatible tests, we can reduce the effort to find a matcher θ. One class of such matching substitutions can be defined as follows:

Definition 3.3 Let $C = \{l_1, \cdots, l_m\}$ and D be clauses and σ a matching substitution mapping a literal in C onto a literal in D. If there is an $l_k \in \{l_1, \cdots, l_m\}$ such that any matching substitution in $uni(C, l_k, D)$ is not strongly compatible with σ then σ is *useless*.

Intuitively we know that a matching substitution σ is useless if the number of 1s in $\beta(\sigma)$ is less than m, the number of literals of C. But the number of 1s in $\beta(\sigma)$ is not always less than m though σ is useless. Let $C = \{l_1, \cdots, l_m\}$ and D be clauses and $\{\sigma_1, \cdots, \sigma_n\}$ be a set of matching substitutions from literals in C to literals in D. We can represent $uni(C, l_k, D)$ by an n-bit sequence M_{l_k} such that its i'th bit is 1 if $\sigma_i \in uni(C, l_k, D)$, otherwise 0 for each k and i $(1 \le k \le m, 1 \le i \le n)$. Given these n-bit sequences, we can easily test whether a matching substitution σ is useless, that is, if there is an l_k such that $M_{l_k} * \beta(\sigma) = 0$ then σ is useless.

Example 3.4 From Example 3.1, we have $uni(C, l_1, D) = \{\sigma_1, \sigma_2, \sigma_3\}$ and $uni(C, l_2, D) = \{\sigma_4, \sigma_5, \sigma_6\}$ and thus $M_{l_1} = 111000$ and $M_{l_2} = 000111$. We have $\beta(\sigma_k)$ for each k $(1 \le k \le 6)$ as shown in Example 3.3. Since $\beta(\sigma_i) * M_{l_2} = 0$ for each i $(1 \le i \le 3)$ and $\beta(\sigma_j) * M_{l_1} = 0$ for each j $(4 \le j \le 6)$, all σ_k $(1 \le k \le 6)$ are useless. □

Theorem 3.1 Let $C = \{l_1, \cdots, l_m\}$ and D be clauses. If $\sigma \in uni(C, l_k, D)$ $(1 \le k \le m)$ is a useless matching substitution then there is no θ such that $C\theta \subseteq D$ and $\theta = \sigma_1 \bullet \cdots \sigma_{k-1} \bullet \sigma \bullet \sigma_{k+1} \cdots \bullet \sigma_m$ where $\sigma_i \in uni(C, l_i, D)$ for each i $(1 \le i \le m, i \ne k)$.
(Proof) Let σ be a member of $uni(C, l_k, D)$ and a useless matching substitution. Suppose that there is a θ such that $C\theta \subseteq D$ and $\theta = \sigma_1 \cdots \bullet \sigma_{k-1} \bullet \sigma \bullet \sigma_{k+1} \cdots \bullet \sigma_m$ where $\sigma_i \in uni(C, l_i, D)$. By Theorem 2.1, $\sigma_1, \cdots, \sigma_{k-1}, \sigma, \sigma_{k+1}, \cdots, \sigma_m$ must be pairwise strongly compatible and thus σ is strongly compatible with $\sigma_i \in uni(C, l_i, D)$ for each i $(1 \le i \le m, i \ne k)$. $\sigma \in uni(C, l_k, D)$ is strongly compatible with itself. Therefore each $uni(C, l_i, D)$ for each i $(1 \le i \le m)$ has at least one matching substitution which is strongly compatible with σ. This is contradictory to that σ is a useless matching substitution. Hence, there is no such θ. □

By Theorem 3.1, it is not necessary to perform pairwise strongly compatible test on useless matching substitutions. If $\sigma \in uni(C, l_k, D)$ is a useless matching substitution then we can remove σ from $uni(C, l_k, D)$ without changing the result of subsumption tests. In Example 3.4, we know that clause C does not subsume clause D without pairwise strongly compatible tests, since all σ_i are useless.

Given two clauses C and D, there may be more than one matcher θ such that $C\theta \subseteq D$. To test that C subsumes D, we only find a matcher, that is, we have no need to find all matchers. By this property we can remove more matching substitutions.

Definition 3.4 Let $C = \{l_1, \cdots, l_m\}$ and D be clauses and $\{\sigma_1, \cdots, \sigma_n\}$ be an ordered set of matching substitutions from literals in C to literals in D. If $\sigma_i, \sigma_j \in uni(C, l_r, D)$ for some r $(1 \le r \le m)$ and σ_j is strongly compatible with each σ_k which is strongly compatible with σ_i then σ_j *includes* σ_i, denoted by $\sigma_i \le_\sigma \sigma_j$, where $k \ne i$ and $k \ne j$.

Let $\{\sigma_1, \cdots, \sigma_n\}$ be an ordered set of matching substitutions and $\gamma(\sigma_i)$ be $\beta(\sigma_i) * \overline{\mu_i^n}$, where $\mu_i{}^n$ is the n-bit sequence such that the value of its i'th bit is 1 and all remaining bits are 0. Then, we can easily test the $\le_\sigma$-relation by bit operations, i.e. if $\gamma(\sigma_i) * \overline{\gamma(\sigma_j)} = 0$ then $\sigma_i \le_\sigma \sigma_j$.

Example 3.5 Let $C = \{p(X), q(Y)\}$, $D = \{p(a), p(b), q(a), q(b)\}$ and $\{X, Y\}$ be an ordered set of variables in C. Then we have $\sigma_1 = \{a/X\}$, $\sigma_2 = \{b/X\}$, $\sigma_3 = \{a/Y\}$, and $\sigma_4 = \{b/Y\}$. By Definition 3.1, we have $\delta(\sigma_1) = (a, \phi)$, $\delta(\sigma_2) = (b, \phi)$, $\delta(\sigma_3) = (\phi, a)$, $\delta(\sigma_4) = (\phi, b)$. We can calculate the following strongly compatible lists of matching substitutions:

$$\beta(\sigma_1) = P_1(a) * P_2(\phi) = 1011 * 1111 = 1011$$
$$\beta(\sigma_2) = P_1(b) * P_2(\phi) = 0111 * 1111 = 0111$$
$$\beta(\sigma_3) = P_1(\phi) * P_2(a) = 1111 * 1110 = 1110$$
$$\beta(\sigma_4) = P_1(\phi) * P_2(b) = 1111 * 1101 = 1101.$$

From this strongly compatible lists, we can obtain $\gamma(\sigma_1)$ and $\gamma(\sigma_2)$ as follows:

$$\gamma(\sigma_1) = \beta(\sigma_1) * \overline{\mu_1^4} = 1011 * 0111 = 0011$$
$$\gamma(\sigma_2) = \beta(\sigma_2) * \overline{\mu_2^4} = 0111 * 1011 = 0011.$$

Since $\gamma(\sigma_1) * \overline{\gamma(\sigma_2)} = 0$, we obtain the relation $\sigma_1 \le_\sigma \sigma_2$. Similarly, we obtain the relation $\sigma_3 \le_\sigma \sigma_4$. □

Theorem 3.2 Let $C = \{l_1, \cdots, l_m\}$ and D be clauses, $\sigma \in uni(C, l_k, D)$ and $\sigma' \in uni(C, l_k, D)$ for some k $(1 \le k \le m)$, $\sigma \le_\sigma \sigma'$ and $\sigma_i \in uni(C, l_i, D)$ for each i $(1 \le i \le m, i \ne k)$. If there is a θ such that $C\theta \subseteq D$ and $\theta = \sigma_1 \bullet \cdots \bullet \sigma_{k-1} \bullet \sigma \bullet \sigma_{k+1} \cdots \bullet \sigma_m$, then there is a θ' such that $C\theta' \subseteq D$ and $\theta' = \sigma_1 \bullet \cdots \bullet \sigma_{k-1} \bullet \sigma' \bullet \sigma_{k+1} \cdots \bullet \sigma_m$.
(Proof) Let us suppose that there is a θ such that $C\theta \subseteq D$ and $\theta = \sigma_1 \bullet \cdots \bullet \sigma_{k-1} \bullet \sigma \bullet \sigma_{k+1} \cdots \bullet \sigma_m$. Then, $\sigma_1, \cdots \sigma_{k-1}, \sigma, \sigma_{k+1} \cdots \sigma_m$ are pairwise strongly compatible by Theorem 2.1. Since σ' includes σ, σ' is strongly compatible with $\sigma_1, \cdots, \sigma_{k-1}, \sigma_{k+1}, \cdots, \sigma_m$. Thus $\sigma_1, \cdots, \sigma_{k-1}, \sigma', \sigma_{k+1}, \cdots, \sigma_m$ are pairwise strongly compatible. Therefore, there is a θ' such that $C\theta' \subseteq D$ and $\theta' = \sigma_1 \bullet \cdots \bullet \sigma_{k-1} \bullet \sigma' \bullet \sigma_{k+1} \cdots \bullet \sigma_m$ by Theorem 2.1. □

By Theorem 3.2, we do not need perform a strongly compatible test on the combinations of matching substitutions which contain a matching substitution σ_1 such that $\sigma_1 \in uni(C, l_i, D)$, $\sigma_2 \in uni(C, l_i, D)$, and $\sigma_1 \le_\sigma \sigma_2$. In Example 3.5, we can remove σ_1 and σ_3 because σ_2 and σ_4 include σ_1 and σ_3, respectively.

As Theorem 3.1 (useless theorem) and Theorem 3.2 (included theorem) suggest, we can remove the useless or included matching substitution before we take a pairwise strongly compatible test. We call a matching substitution which is either useless or included *unnecessary*.

One phenomenon we want to point out is that a matching substitution becomes unnecessary due to the propagation of deletion, so needs to be deleted. Therefore we should keep deleting unnecessary matching substitutions until there is no more such matching substitution. For examples, let σ_1, σ_2 and σ_3 be matching substitutions from literals in C to literals in D, and let the number of literals in C be 3. Suppose that σ_1 is strongly compatible with σ_2 and σ_3, and σ_2 is not strongly compatible with σ_3. Then σ_1 is not a useless matching substitution. However, the removal of useless matching substitutions, σ_2 and σ_3, causes σ_1 to be a useless matching substitution and thus it can be removed.

Let $C = \{l_1, \cdots, l_n\}$, and D clause. Then, in the worst case $O(n^2)$ strongly compatible tests will be needed for each combination $(\sigma_1, \cdots, \sigma_n) \in \times_{i=1}^{n}$ $uni(C, l_i, D)$ in order to check C subsumes D. However, given $\beta(\sigma_i)$ we can enhance the performance of a subsumption test by the following theorem.

Theorem 3.3 Let $C = \{l_1, \cdots, l_m\}$ and D be clauses, $\{\sigma_1, \cdots, \sigma_n\}$ be a set of matching substitutions from literals in C to literals in D, and $\{\sigma_{x_1}, \cdots, \sigma_{x_m}\}$ be a subset of $\{\sigma_1, \cdots, \sigma_n\}$. There is a $\theta = \sigma_{x_1}\bullet \cdots \bullet\sigma_{x_m}$ such that $C\theta \subseteq D$ and $\sigma_{x_k} \in uni(C, l_k, D)$ for each k $(1 \leq k \leq m)$ if only if $*_{k=1}^{m}\ \beta(\sigma_{x_k}) * +_{k=1}^{m}\ {\mu_{x_k}}^{n} = +_{k=1}^{m}\ {\mu_{x_k}}^{n}$.

(Proof) ($\leftarrow$) Since $*_{k=1}^{m}\ \beta(\sigma_{x_k}) * +_{k=1}^{m}\ {\mu_{x_k}}^{n} = +_{k=1}^{m}\ {\mu_{x_k}}^{n}$, by Lemma 3.2, $\sigma_{x_1}, \cdots, \sigma_{x_m}$ are strongly compatible with each of $\{\sigma_{x_1}, \cdots, \sigma_{x_m}\}$. Therefore $\sigma_{x_1}, \cdots, \sigma_{x_m}$ are pairwise strongly compatible. Thus, by Theorem 2.1, there is a $\theta = \sigma_{x_1}\bullet \cdots \bullet\sigma_{x_m}$ such that $C\theta \subseteq D$ and σ_{x_k} $\in uni(C, l_k, D)$ for each k $(1 \leq k \leq m)$
($\rightarrow$) By Theorem 2.1, $\sigma_{x_1}, \cdots, \sigma_{x_m}$ are pairwise strongly compatible. Therefore, by Lemma 3.2, the x_i bit of $\beta(\sigma_{x_k})$ for each i, k $(1 \leq i, k \leq m)$ is 1. Thus $*_{k=1}^{m}$ $\beta(\sigma_{x_k}) * +_{k=1}^{m}\ {\mu_{x_k}}^{n} = +_{k=1}^{m}\ {\mu_{x_k}}^{n}$. □

Now we can formulate a new algorithm that returns a pairwise strongly compatible set $\{\sigma_1, \cdots, \sigma_m\}$ such that $(\sigma_1, \cdots, \sigma_m) \in \times_{i=1}^{m}$ $uni(C, l_i, D)$ if exists, otherwise return $\{\}$. The detail algorithm, *Pairwise Strongly Compatible Test* (PSCT), is described in Figure 1 and it can be summarized as follows:

1. Calculate the strongly compatible list for each matching substitution.
2. Remove unnecessary matching substitutions until there is no such matching substitution.
3. Find out an m-tuple $(\sigma_1, \cdots, \sigma_m)$ such that $*_{k=1}^{m}$ $\beta(\sigma_k) * +_{k=1}^{m}\ {\mu_k}^{n} = +_{k=1}^{m}\ {\mu_k}^{n}$.

Example 3.6 Given $C = \{p(X,Y), r(Y,Z), s(X,Z)\}$ and $D = \{p(b,a), p(a,b), r(a,d), r(b,c), s(a,d), s(a,c)\}$ we want to find out a substitution θ such that $C\theta \subseteq D$. Let $\{X, Y, Z\}$ be an ordered set of variables in C. Then, we can obtain that

$M_{p(X,Y)} = 110000$, $M_{r(Y,Z)} = 001100$, $M_{s(X,Z)} = 000011$,
$\beta(\sigma_1) = 101000$, $\beta(\sigma_2) = 010111$, $\beta(\sigma_3) = 101010$,
$\beta(\sigma_4) = 010101$, $\beta(\sigma_5) = 011010$, $\beta(\sigma_6) = 010101$.

Since $\beta(\sigma_1) * M_{s(X,Z)} = 0$, σ_1 is removed and thus the strongly compatible lists are adjusted as follows:

$\beta(\sigma_2) = 010111$, $\beta(\sigma_3) = 001010$, $\beta(\sigma_4) = 010101$,
$\beta(\sigma_5) = 011010$, $\beta(\sigma_6) = 010101$.

Since $\beta(\sigma_3) * M_{p(X,Y)} = 0$, σ_3 is useless. By further removing the useless matching substitution σ_3, we can obtain that

$\beta(\sigma_2) = 010111$, $\beta(\sigma_4) = 010101$, $\beta(\sigma_5) = 010010$,
$\beta(\sigma_6) = 010101$.

Since $\beta(\sigma_5) * M_{r(Y,Z)} = 0$, σ_5 is useless and thus removed. Consequently we can obtain following strongly compatible lists:

$\beta(\sigma_2) = 010101$, $\beta(\sigma_4) = 010101$, $\beta(\sigma_6) = 010101$.

Since $\beta(\sigma_2) * \beta(\sigma_4) * \beta(\sigma_6) * 010101 = 010101$, σ_2, σ_4 and σ_6 are pairwise strongly compatible. Thus, there is a substitution $\theta = \sigma_2\bullet\sigma_4\bullet\sigma_6 = \{a/X, b/Y, c/Z\}$. □

```
Our Algorithm PSCT
Input: clauses $C = \{l_1, \cdots, l_m\}$ and $D$
Output: a pairwise strongly compatible set $\{\sigma_1, \cdots, \sigma_m\}$
such that $(\sigma_1, \cdots, \sigma_m) \in \times_{i=1}^{m}$ $uni(C, l_i, D)$

1. Calculate $\beta(\sigma)$ for all $\sigma \in \bigcup_{i=1}^{m}$ $uni(C, l_i, D)$.
2. Let I be an n-bit sequence such that all its bits are 0.
   (a) for each $\sigma_k \in \bigcup_{i=1}^{m}$ $uni(C, l_i, D)$, if $\sigma_k$ is useless then
       i. remove $\sigma_k$.
       ii. I = I + ${\mu_k}^{n}$.
   (b) for each $\sigma_k \in \bigcup_{i=1}^{m}$ $uni(C, l_i, D)$, if there is a $\sigma_l$ such that $\sigma_k \leq_\sigma \sigma_l$ then
       i. remove $\sigma_k$.
       ii. I = I + ${\mu_k}^{n}$.
   (c) for each $\sigma_k \in \bigcup_{i=1}^{m}$ $uni(C, l_j, D)$, $\beta(\sigma_k) = \beta(\sigma_k) * \overline{I}$.
3. If $uni(C, l_i, D) = \{\}$ for some $i$, then return $\{\}$.
4. Repeat step 2~3 until there is no unnecessary matching substitution.
5. For each $m$-tuple $(\sigma_{i_1}, \cdots, \sigma_{i_m})$ where $\sigma_{i_k} \in uni(C, l_k, D)$,
   if $*_{k=1}^{m}\ \beta(\sigma_{i_k}) * +_{k=1}^{m}\ {\mu_{i_k}}^{n} = +_{k=1}^{m}\ {\mu_{i_k}}^{n}$, then return $\{\sigma_{i_1}, \cdots, \sigma_{i_m}\}$.
6. return $\{\}$.
```

Figure 1: Algorithm PSCT

4 Related Works and Analysis

This section compares our algorithm with the two existing s-link tests, namely Eisinger's algorithm and Socher's algorithm. The analysis is based on the number of string comparisons to determine whether a clause $C = \{l_1, \cdots, l_m\}$ subsumes a clause D. To measure the complexity of three algorithms, we use the following symbols:

r: the maximal arity of predicate symbols occurring in literals in clauses C and D.

N_C: the number of distinct variables in a literal in clauses C and D.

N_D: the number of distinct terms which are substituted for a variable in clause C.

N_S: the number of strongly compatible tests needed to see whether m matching substitutions between literals are pairwise strongly compatible.

N_P: the number of pairwise strongly compatible tests needed to find a matcher θ such that $C\theta \subseteq$ D.

To simplify the analysis, we assume that the number of matching substitutions in each $uni(C, l_i, D)$ $(1 \leq i \leq m)$ is equal and let it be k.

In Eisinger's algorithm, subsumption tests for long clauses with more than one matching substitution for each literal may require an expensive search of all elements of the Cartesian product. Since compositions of substitutions are needed to see whether two given substitutions are strongly compatible and $O(N_C^2)$ string comparisons are needed for each strongly compatible test, $O(N_S N_C^2)$ string comparisons are needed for each pairwise strongly compatible test. Thus $O(N_P N_S N_C^2)$ string comparisons are needed for the subsumption test. Since $k^2 \leq N_P \leq k^m$, $1 \leq N_S \leq \frac{m(m-1)}{2}$, and $1 \leq N_C \leq r$, in the worst case $N_C = r$, $N_S = m(m-1)/2$ and $N_P = k^m$, so the worst-case time complexity of Eisinger's is $O(k^m m^2 r^2)$.

Socher proposes an improvement of the s-link test for subsumption of two clauses [Socher 1988]. He improves the search for θ such that $C\theta \subseteq D$ by imposing a restriction on the possible matching substitutions. It is based on the idea of giving the variables and literals of a clause a characteristic property, which in fact denotes information about the occurrences of variables in various argument positions of a literal. An order for these characteristic is defined and it is shown that the order is compatible with the matching substitution σ from C to D. Thus all matching substitutions that do not respect the order can be singled out. However, he does not improve the pairwise strongly compatible test itself and thus does not reduce the worst-case time complexity of the s-link test.

In some cases Socher's algorithm can not single out a matching substitution which is either useless or included matching substitution. For example, let $C = \{p(X,Y), q(Y,X)\}$ and $D = \{p(a,c),\ p(b,d),\ q(c,b),\ p(d,a)\}$ be given. No matching substitution is singled out because the characteristic matrices of literals p and q in C are equal to those ones in D. However, all matching substitutions are useless in our approach, so no pairwise strongly compatible test is performed.

By using strongly compatible lists and bit operations, we improve the pairwise strongly compatible test and thus reduce the worst-case time complexity of the s-link test. $O(km^2 N_C N_D)$ string comparisons are needed to calculate all strongly compatible lists. $O(m)$ bit-conjunctions are needed for a pairwise strongly compatible test when m strongly compatible lists are given. Thus, $O(km^2 N_C N_D)$ string comparisons and $O(mN_P)$ bit-conjunctions are needed for a subsumption test. Since $k^2 \leq N_P \leq k^m$, $1 \leq N_C \leq r$, and $1 \leq N_D \leq km$, in the worst case $N_P = k^m$, $N_C = r$ and $N_D = km$, so the worst-case time complexity is $O(k^2rm^3$ string comparisons $+\ mk^m$ bit-conjunctions).

Let n be the ratio of the time complexity of a string comparison to the time complexity of a bit-conjunction. Then, in the case that k^2rm^3 is greater than $\frac{mk^m}{n}$, the worst-case time complexity of our algorithm is $O(k^2rm^3)$ and we can reduce the worst-case time complexity of Eisinger's algorithm by $O(\frac{k^{m-2}r}{m})$. In the other case, the worst-time complexity of our algorithm is $O(\frac{mk^m}{n})$ and we can reduce the worst-case time complexity of Eisinger's algorithm by $O(mr^2n)$.

5 Conclusions

Subsumption tests for long clauses with more than one matching substitution for each literal may require an excessive search for all elements in the Cartesian product. We have presented a new subsumption algorithm, called PSCT algorithm, which has a lower worst-case time complexity than the existing methods. The efficiency of our algorithm is based on the following facts.

1. Construction of strongly compatible lists allows us to identify unnecessary matching substitutions at the early stage of the subsumption test. Such matching substitutions are removed and are not involved at the actual pairwise strongly compatible test to come. This filtering process reduces the number of possible combinations of matching substitutions clearly.

2. As for the pairwise strongly compatible test itself, the test is carried out efficiently due to the appropriate bit operations on the strongly compatible lists which are already constructed.

The approaches [Socher 1988, Eisinger 1981] that actually compose the matching substitutions to check pairwise compatibility are considered to be slow and expensive. In most cases our approach outperforms others [Socher 1988, Eisinger 1981] even though it may involve the cost overhead for computation of the strongly compatible lists of matching substitutions. Furthermore, it should be noted that our subsumption algorithm can be used in general theorem proving approach even though it is described in the context of the connection graph proof procedure in this paper.

References

[Andrew 1981] P. B. Andrews, Theorem Proving via General Matings, *Journal of ACM* **28** (2) (1981) 193-214.

[Bibel 1981] W. Bibel, On Matrices with Connections, *Journal of ACM* **28** (4) (1981) 633-645.

[Chang and Lee 1973] C. L. Chang and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, New York, 1973).

[Eisinger 1981] N. Eisinger, Subsumption and Connection Graph, in: *Proceedings of the IJCAI-81* (1981) 480-486.

[Gottlob and Leitsch 1985] G. Gottlob and A. Leitsch, On the Efficiency of Subsumption Algorithms, *Journal of ACM* **32** (2) (1985) 280-295.

[Juang et al. 1988] J. Y. Juang, T. L. Huang, and E. Freeman, Parallelism in Connection Graph-based Logic Inference, in: *Proceedings of the 1988 International Conference on Parallel Processing* **2** (1988) 1-8.

[Kowalski 1975] R. Kowalski, A Proof Procedure using Connection Graphs, *Journal of ACM* **22** (4) (1975) 572-595.

[Kowalski 1979] R. Kowalski, *Logic for Problem Solving* (North Holland, Oxford, 1979).

[Loganantharaj 1986] R. Loganantharaj, Parallel Theorem Proving with Connection Graphs, in: *Proceedings of 8th International Conference on Automated Deduction* (1986) 337-352.

[Loganantharaj 1987] R. Loganantharaj, Parallel Link Resolution of Connection Graph Refutation and its Implementation, in: *Proceedings of International Conference on Parallel Processing* (1987) 154-157.

[Loveland 1978] D. Loveland, *Automated Theorem Proving: A Logical Basis* (North-Holland, Amsterdam, 1978).

[Munch 1988] K. H. Munch, A New Reduction Rule for the Connection Graph Proof Procedure, *Journal of Automated Reasoning* **4** (1988) 425-444.

[Robinson 1965] J. A. Robinson, A Machine-Oriented Logic Based on the Resolution Principle, *Journal of ACM* **12** (1) (1965) 23-41.

[Sickel 1976] S. Sickel, A Search Technique for Clause Interconnectivity Graphs, *IEEE Transaction on Computers* **25** (8) (1976) 823-835.

[Socher 1988] R. Socher, A Subsumption Algorithm Based on Characteristic Matrices, in: *Proceedings of 9th International Conference on Automated Deduction* (1988) 573-581.

[Stillman 1973] R. B. Stillman, The Concept of Weak Substitution in Theorem-Proving, *Journal of ACM* **20** (4) (1973) 648-667.

[Wos 1986] L. Wos, Automated Reasoning: Basic Research Problems, Argonne National Laboratory, Technical Memorandum No.67, March 1986.