

Short Paper

Completeness of the LELS Inference Rule in Automated Theorem Proving

CHUEN-HSUEN JEFF HO
Department of Computer Science
Chung-Cheng Institute of Technology
Tao-Yuan, Taiwan, 335, R.O.C.
E-mail: jeffho@cs.ccit.edu.tw

The Extended Linking Strategy (ELS) is a hyper-style strategy whose underlying principle is to control and perform (extend) a series of standard resolution on clauses. However it may be treated as a unique inference rule that serially links several resolution steps into one. We define links and clause chains, and introduce the ideas of ELS with left merging (LELS). We also presented the soundness and completeness proofs for ground LELS and use a fundamental theorem of logic (Herbrand's theorem) and facts about the unification algorithm to show that LELS is, in fact, complete for first-order predicate calculus. Some experimental results are included. Employment of LELS not only prevents an automated proving program from producing too many new clauses, but also enables it to draw in fewer steps conclusions that typically require many steps when unlinked inference rules are used.

Keywords: extended linking, left merging, clause chain, cycling, refutation completeness, induction proof

1. INTRODUCTION

The Extended Linking Strategy (ELS) for resolution theorem proving [1], which will be discussed in the following sections, is related to existing, well-known hyper- and linking-strategies, the goal of which is to produce inference schemes that take larger and, hopefully, more "relevant/interesting/etc" inference steps. ELS was developed to deal with two existing problems in the Automated Theorem Proving field. The first problem we deal with is that, given an initial set of clauses and a set of inference rules for generating new knowledge from these clauses, automated reasoning systems may generate a potentially infinite number of possible inferences. In general, only a small subset of the possible inferences of an automated reasoning system is useful.

The other problem is the restrictions in the existing reasoning rules. For example, unit clauses (or satellites) must be included in the initial clause set in order for UR-resolution [2] (or Hyper-resolution [3]) to be applied. Without unit clauses, some rules cannot be applied to the system at all. The user has relatively little control over which clauses the program will focus on.

Received January 23, 1996; accepted May 12, 1998.
Communicated by Jieh Hsiang.

1.1 What is ELS?

The idea of ELS can be stated as follows: a clause chain consists of two or more clauses linked together to combine a series of resolutions into a single step to produce a single resolvent. ELS is similar to *Hyper-resolution* [3], *UR-resolution* [2], *linked UR-resolution* [4] and the like. Hyper-resolution resolves, in a single step, a nucleus clause with one of more satellite(unit) clauses. This is an example of branching. Branching means that a given clause may be resolved with any number of other clauses during a single resolution step.

ELS uses chaining rather than branching. ELS starts linking from a clause C_0 to clause C_1 by connecting a literal of C_0 and its counterpart in C_1 . It then picks one of the remaining literals in C_1 and links to C_2 , etc. until it reaches the last clause C_d . The unification follows the *direction* of the chaining. All intermediate clauses are linked to exactly two other clauses (the immediate predecessor and immediate successor).

Following is a set of clauses S^* which will be used for illustration purposes throughout the definition sections. Let $S^* = \{\neg P(a), P(y)\neg M(y)R(b), M(z)R(c)Q(x), \neg R(y)T(y)Q(b), \neg T(x)P(y)\}$. Note that M, P, Q, R, T are predicates, a, b, c are constants, and x, y, z are variables. A link connecting a literal of one clause with a unifiable literal of the opposite sign in another clause is called an R -link and is denoted by \xrightarrow{R} . Note that we use a directed arrow to emphasize the directed nature of chains in ELS even though resolution is not directional itself. For example, clauses 1 and 2 can be linked together from $\neg P(a)$ to $P(y)$ with a unifier a/y , $\neg P(a) \xrightarrow{R} P(a)\neg M(a)R(b)$.

A clause chain of length d is a sequence of clauses C_0, \dots, C_d such that

- a. the C_i 's are all distinct instances with separate variables;
- b. there is exactly one R -link from C_i to C_{i+1} for each i ;
- c. each R -link inherits the unifiers of variables from previous ones;
- d. no two R -links are incident with a common literal.

For example, let C_0 be the clause $\neg P(a)$ in S^* . Then, a clause chain of length 4 without merging can be constructed from it as follows:

$$\begin{array}{l}
 \neg P(a) \xrightarrow{R} P(y)R(b)\neg M(y) \quad \text{unifier } a/y \\
 \xrightarrow{R} M(z)Q(x)R(c) \quad \text{unifier } a/z \\
 \xrightarrow{R} \neg R(y)Q(b)T(y) \quad \text{unifier } c/y \\
 \xrightarrow{R} \neg T(x)P(y) \quad \text{unifier } c/x, a/y.
 \end{array} \tag{1}$$

The clause in a chain with no incoming R -link is the START clause. The clause with no outgoing R -link is the END clause. The remaining clauses are intermediate clauses. For example, in clause chain (1), $\neg P(a)$ and $\neg T(x)P(y)$ are the START and END clauses, respectively. An R -path is a series of R -links in a clause chain. The length of the R -path is the number of R -links in the path. An *avail*-literal of a chain is a literal which has no connection with the R -path. For example, the *avail*-literals of clause chain (1) is $\{R(b)Q(b)\}$. The *direction* of the linking process is from C_0 to C_d ; i.e., it follows the direction of the R -links. In a clause chain, $\& = \{C_0, \dots, C_k\}$ a *cycle* is formed between C_i and C_j , where $i \leq k, j \leq k, i \neq j$, if there exists an R -link from an *avail*-literal of C_j back to an *avail*-literal of C_i .

ELS clause chains will not be allowed to be cyclic. The prohibition of clause cycling is necessary to assure that the linked clauses correspond to a sound deduction and to enforce the linking restrictions in order to avoid tautologous results.

1.2 Chaining and Left-Merging

Merging of identical and unifiable literals is allowed during the chaining process. Left merging is similar to the *merge back* operation used in OL-resolution[5]. Left merging keeps the very first occurrence of a literal and deletes the identities appearing at other clauses of a growing clause chain. For example, $Q(b)$ of C_3 in clause chain (1) is left-merged by $Q(x)$ of C_2 with unifier b/x . Following is the formal definition of Left-Merging.

Definition: A literal L of a clause C_i in a chain $\&$ is left-merged with a literal L' of C_j , $j < i$ in $\&$ if L is not in the R -path and L, L' are unifiable with a unifier σ .

Definition: ELS with Left-merging is denoted as LELS. A clause chain established by applying LELS is an LELS chain.

Definition: Let $\&$ be a chain, and let C_i be a clause in $\&$. $RLST(C_i)$ is the sequence $\{L_0, \dots, L_{i-1}\}$ of literals from C_j , $j < i$, $C_j \in \&$ with outgoing R -links. For example, $RLST(C_4)$ in (1) is $\{\neg P(a), \neg M(a), R(c), T(c)\}$.

Definition: A literal L of a clause C_i in a chain $\&$ is backward-deleted by $\neg L$ of a clause C_j , $j < i$ if $\neg L$ is an element of $RLST(C_j)$. For example, in clause chain (1), the literal $P(y)$ of C_4 is backward-deleted by $\neg P(a)$ of C_0 with unifier a/y because $\neg P(a) \in RLST(C_4)$.

Definition: A literal is called a *free*-literal during a chaining process (after unification and substitution) if it is an *avail*-literal and is neither a left-merged nor a backward-deleted literal. For example, the *free*-literals of clause chain (1) are $R(b)$ and $Q(b)$.

The termination conditions of an LELS chain play a very important role in proving theorems. A chain without termination cannot have a resolvent. It also destroys the completeness properties in proofs. When a chain terminates, its Clause Chain Resolvent (CCR) can be generated by resolving along the R -path of the chain. \triangleright CCR means to derive the resolvent of a chain. Resolving a clause chain is collecting all the *free*-literals in the chain; ie., the CCR is the disjunction of these *free*-literals.

Definition: Let $\&$ be a chain, and let C_i be a clause in $\&$. $FREE(C_i)$ is a set of *free*-literals collected from every $C_k \in \&$, $k \leq i$. For example, in clause chain (1), $FREE(C_4) = \{R(b), Q(b)\}$ and $FREE(C_3) = \{R(b), Q(b)\}$.

A clause chain $\& = \{C_0, \dots, C_k\}$ with chain length limit d ends if C_k is

1. a unit clause; or
2. a non-unit clause with $FREE(C_k) - FREE(C_{k-1}) = \{\}$; or
3. a non-unit clause with $FREE(C_k) - FREE(C_{k-1}) \neq \{\}$ and
 - 3a. a cycle is formed; or
 - 3b. $FREE(C_k) \cap RLST(C_k) \neq \{\}$; or

- 3c. no more target clauses are available in the search space; or
4. the END clause because $k = d$.

Conditions 1 and 2 are obvious because there are no more literals left for C_k to continue the linking. For example, in clause chain (1), $\text{FREE}(C_4) - \text{FREE}(C_3) = \{\}$. C_4 has no *free*-literals left for further linking. Condition 3a is set for avoiding tautologous results. Condition 3b is used to prevent possible infinite chaining with repeated clauses. This is a special restriction used for LELS. Condition 3c is applied when no available clause in the search space can link with C_k either because it is excluded by conditions 3a and 3b or it has no connection with C_k . Condition 4 is an optional condition, controlled by the program user.

2. GROUND LELS

Propositional (or ground) LELS is applied only to ground clauses. It leads naturally to a general principle of LELS for first-order clauses to be discussed later. It is, thus, natural to introduce first-order LELS by first expressing the rule at the simpler ground level.

Definition: A ground LELS chain is an LELS chain containing only ground clauses.

Definition: Let S be a ground set and $C \in S$. An LELS-derivation from S is a sequence of clause chains $\&_1, \dots, \&_n$ with R_1, \dots, R_n the CCRs satisfying the following conditions:

1. $\&_1$ is a chain starting with C .
2. $\&_{i+1}$ is a chain starting with R_i , $i \geq 1$.
3. Each $\&_i$ satisfies one of the chain termination conditions.

Definition: A clause chain $\& = \{C_0, \dots, C_k\}$ is a *contradiction* if it has no *free*-literals, $\text{FREE}(C_k) = \{\}$.

Definition: An LELS-refutation of S is an LELS-derivation of a contradictory chain. Equivalently, it is a derivation whose last chain resolvent is the empty clause (\square). We denote it by $S \succ_{\text{lels}} \square$.

Most of the theorems in this paper are proved by using the *Splitting Rule* of Davis and Putnam[6]: Let S be a ground set, and let P be an atom occurring in S . S can be put into the form $P \vee A_1, \dots, P \vee A_m, \neg P \vee B_1, \dots, \neg P \vee B_n, C_1, \dots, C_k$, where A_i, B_i and C_i are free of P and $\neg P$. Let $S_1 = \{A_1, \dots, A_m, C_1, \dots, C_k\}$ and $S_2 = \{B_1, \dots, B_n, C_1, \dots, C_k\}$. S is unsatisfiable if and only if both S_1 and S_2 are unsatisfiable. Let $k(S)$ denote the number of occurrences of literals in S minus the number of clauses in S . Let S' be formed from S by deleting a literal from a non-unit clause. Then, $k(S') < k(S)$. Similarly, if S' is formed from S by deleting a clause from S , then $k(S') \leq k(S)$, and the comparison is strict if the deleted clause is non-unit. Finally, let $S_1(S_2)$ be one of the sets formed by splitting for a minimally unsatisfiable set S of clauses. If $k(S) > 0$, then $k(S_i) < k(S)$ for $i = 1, 2$.

Lemma 1: Let C_0, \dots, C_k be ground clauses, and let $\& = \{C_0, \dots, C_k\}$ be a clause chain. Let I be an interpretation. If I satisfies each C_i , then I satisfies the CCR of $\&$.

Proof: Because computing the CCR from the clauses of the chain involves backward deletion, the proof is not a trivial consequence of the soundness of ordinary resolution. The proof is by induction on the length k of $\&$.

Base Case: $k = 1$. In this case, the chain is just $\{C_0, C_1\}$, and the CCR is just an ordinary binary resolvent. The result follows from resolution theory.

Induction Case: $k > 1$. Suppose the result is true for chains of length less than k . Then, the chain resolvent CCR_{k-1} of $\{C_0, \dots, C_{k-1}\}$ is satisfied by I . Now, in computing the CCR for $\&$, one literal from CCR_{k-1} (in fact from C_{k-1}) is chosen to resolve with a literal from C_k . Let the chosen literal be P , and let C_k be $\neg P \vee L_1 \vee \dots \vee L_n$. By hypothesis, I satisfies C_k .

- a. If I makes P false, then some other literals of CCR_{k-1} must be true, and these literals also occur in CCR_k . Clearly, CCR_k is satisfied by I .
- b. Suppose now that I makes P true. Then, one of the literals L_m must be true because I satisfies C_k . If any such L_m remains in CCR_k , then again I satisfies CCR_k . The only way this can fail is if each such L_m is backward-deleted. For L_m to be backward-deleted, there must have been clauses C_i and C_{i+1} earlier in the chain such that $C_i = \neg L_m \vee C'_i$ and $C_{i+1} = L_m \vee C'_{i+1}$, and the R -link from C_i originated from $\neg L_m$. Moreover, since L_m is true in I in this subcase, some other literal of C_i must be true, say M .
 - b_1 . Some true M from C_i is not itself backward-deleted. If such an M is not backward-deleted, then either it occurs in CCR_k or it is left-merged or resolved because every new R -link must emanate from a *free*-literal of the most recent clause in the chain. If M occurs in CCR_k , then I satisfies CCR_k . If, in fact, M had been resolved, then M would occur on $\text{RLST}(C_k)$ to the right of $\neg L_i$ from CCR_k . Thus, L_i would, in fact, occur in CCR_k , and I would satisfy CCR_k .
 - b_2 . If all the true literals of C_i were themselves backward-deleted, then by the induction hypothesis, CCR_i is satisfied by I . Recall, $C_i = \neg L_m \vee M_j \vee \dots \vee C''_i$, where M_j are all the true literals of C_i . (Recall, that L_m was assumed to be true in I .) Therefore, some true literal from clause C_v , $v < i$ must occur in CCR_i . This literal can now play the role of M in b_1 . □

The intention behind the application of LELS is to derive a contradictory chain from a ground clause set, thus proving the inconsistency of the set; this then proves the unsatisfiability of the set, given that LELS is sound. Soundness can be stated as follows:

Theorem 1: *LELS is sound.*

Proof: Trivial because the CCR of each LELS chain is formed from the clauses by using R -links to resolve literals with opposite signs, i.e., a sequence of ordinary resolutions. □

Example 1: Following is a minimal unsatisfiable ground set:

- | | | | |
|---------------|--------------------|--------------------|--------------------|
| 1. $L \neg M$ | 2. $\neg L \neg P$ | 3. $L \neg Q$ | 4. $\neg L \neg R$ |
| 5. $N M Q$ | 6. $N P R$ | 7. $\neg N \neg T$ | 8. T |

Solution by LELS: If 5. $\{NMQ\}$ is selected as the START clause, then the LELS-derivation of the set can be derived as the follows:

$$NMQ \xrightarrow{R} \neg QL \xrightarrow{R} \neg L \neg P \xrightarrow{R} PNR \xrightarrow{R} \neg R \neg L.$$

The chain terminated on clause 4, $\{\neg R \neg L\}$, because $\neg L$ is backward-deleted (termination condition 2). N is a left-merged literal. By collecting literals N and M , RLST(C_4) = $\{Q, L, \neg P, R\}$, FREE(C_4) = $\{N, M\}$. The newly derived clause $\{NM\}$ is assigned to be clause 9. From it, we can construct another LELS chain:

$$NM \xrightarrow{R} \neg ML \xrightarrow{R} \neg L \neg P \xrightarrow{R} PNR \xrightarrow{R} \neg R \neg L.$$

RLST(C_4) = $\{M, L, \neg P, R\}$ and FREE(C_4) = $\{N\}$. $\{N\}$ is resolved as clause 10 and starts the contradictory chain:

$$N \xrightarrow{R} \neg N \neg T \xrightarrow{R} T.$$

3. THE LELS COMPLETENESS PROPERTIES

We can prove the completeness of LELS resolution in two steps. Because the cycle termination property requires special consideration, we first prove the completeness of a modified LELS in which the cycle termination condition is not used.

3.1 Ground Cases

Definition: Simple LELS (SLELS) resolution is LELS in which cycle terminations are ignored.

Thus, chains are terminated in SLELS only when there are no *free*-literals in the last clause from which to generate another *R*-link.

Theorem 2: Let S be a minimally unsatisfiable set of clauses. Let D be a clause of S , and let P be a literal of D . There exists an SLELS refutation of S starting from P in D .

Proof: The proof is by induction on $k(S)$.

Base case: $k(S) = 0$. Then, S consists of two conflicting unit clauses, and the proof is trivial.

Induction case: $k(S) > 0$. Assume that the result holds for any minimally unsatisfiable set S' with $k(S') < k(S)$. Let the clauses of S be $P \vee A_1, \dots, P \vee A_n, \neg P \vee B_1, \dots, \neg P \vee B_m, C_1, \dots, C_k$, as usual, and suppose D is the clause $P \vee A_1$. There are two subcases.

- a. A_1 is empty. Then, the unit clause P occurs in S , and there are no other clauses containing P . Consider the set $S_2 = \{B_1, \dots, B_m, C_1, \dots, C_k\}$. S_2 is, in fact, also minimally unsatisfiable, and of course $k(S_2) < k(S)$. By induction, there is an SLELS refutation R of S_2 starting from some literal in B_1 . Let the chains of R be $\&_1, \dots, \&_s$. We will consider these chains one by one. Let the first chain be $\&_1 : B_1$

$\xrightarrow{R} \dots \xrightarrow{R} D_i$, where D_i is the END clause. Now, we append $\neg P$ to all the B_i clauses used in this chain and append a new R -link to the beginning of $\&_1$ as follows: $P \xrightarrow{R} \neg P \vee B_1 \xrightarrow{R} \dots \xrightarrow{R} D'_i$, where each D'_i is either D_i or $D_i \vee \neg P$. By the definition of LELS, all $\neg P$ in each B_i as well in each $D_i \vee \neg P$ clause are immediately backward-deleted by P because $\neg P$ is in $\text{RLST}(D_i)$. It is clear that the chain resolvent formed at the point where P is added is exactly the same as the resolvent up to the corresponding point in $\&_1$. We now repeat the argument for the remaining chains.

- b. A_1 is not empty. Let Q be a literal of A_1 . Then, D is the clause $P \vee Q \vee A'_1$. Let S_1 and S_2 be the split sets obtained by splitting on Q , and let S_{1m} and S_{2m} be corresponding minimally unsatisfiable subsets. Again, note that $P \vee A'_1$ must be in S_{1m} . $k(S_{1m}) < k(S)$, so by induction, there is an SLELS refutation R of S_{1m} starting from P in $P \vee A_1$. Now, add Q back to all the clauses from which it was deleted. Consider the first chain of R , $\&_1$. Again, following our remarks about literals in $\text{FREE}(D_i)$, it is clear that Q remains in $\&_1$. Moreover, if any other clause in that chain gets the literal Q added back, it will be left-merged. Of course all other mergings will be exactly the same as before. Thus, all occurrences of Q are merged, and no other literal in $\&_1$ that was not merged before will be newly merged. This means that for each clause D_i in this chain, exactly the same literals as before are available for linking; and therefore, each former R -link can still be made. Finally, the last clause has no literals available for linking, even if a Q was added back to it. The result is a modified chain which has exactly the same R -links and whose resolvent is $Q \vee \text{CCR}(\&_1)$. Again, the same argument can be repeated for the remaining chains of R . The result is an SLELS deduction R' of the unit clause Q . We will now consider the set S_3 formed by deleting all clauses containing Q and adding the unit clause Q . Let S_4 be a minimally unsatisfiable subset of S_3 . Again, Q must occur in S_4 . Because there is at least one non-unit clause from S containing Q (namely $P \vee Q \vee A'_1$), $k(S_4) < k(S)$, by induction, there is an SLELS refutation RQ of S_4 starting from Q . Then, the refutation formed by appending RQ to R' is an SLELS refutation of S starting from P in D . \square

Theorem 3: Let S be a minimally unsatisfiable set of clauses, and let D be a clause in S with literal P . Then, there is an LELS refutation of S starting from P in D .

Proof: According to the previous theorem, there is an SLELS refutation of S starting from P in D . In each chain $\&_i$ we can trace the chain to find possible cycle termination conditions. Whenever one is found, the chain is broken at a certain clause, a CCR is formed, and a new chain starting from that chain resolvent using the same links as in the original chain is begun. Clearly, the final resolvent for any such broken chain is the same as before. Of course, the resolvent for the last chain of the original refutation is, then, still the empty clause. \square

3.2 First Order Cases

The proof for ground clauses may be lifted to first-order clauses by means of the standard “lifting lemma” explained in [7]. This lemma states that *for any ground instances of clauses C and D , each of their resolvents are instances of some resolvent of C and D* . Since,

according to Herbrand's theorem, a set of clauses is unsatisfiable if and only if there is a set of ground instances of them that is unsatisfiable, the lifting lemma assures that use of the most general unifier is sufficient to achieve completeness. Note that the standard lifting lemma did not handle any merging situation. As stated in section 2, the ground LELS carries the left merging and possible cycle termination in a chaining process. For first order cases, there will be no forced merging on two literals unless they are identical.

Lemma 2: If C_0', \dots, C_r' are instances of C_0, \dots, C_r , respectively, and if R' is a CCR of a clause chain \mathcal{E}' of C_0', \dots, C_r' , then there is a CCR R of \mathcal{E} of C_0, \dots, C_r such that R' is an instance of R .

Proof: Assume that the variables are all distinct. Let R_0', \dots, R_r' be the literals in the RLST of \mathcal{E}' . Let γ be the substitution that maps C_i onto C_i' , $i = 0..r$. We construct a chain \mathcal{E} starting from C_0 . Let R_0 be the set of literals from C_0 mapped onto R_0' in C_0' by γ . Let $\neg R_0'$ be the set of literals from C_1 mapped onto $\neg R_0'$ in C_1 by γ . We form the resolvent in the normal way with σ_0 as mgu. If there were any left-merged literals, then it is necessary to unify the corresponding literals by additional substitution. Let τ_0 be the final substitution. Clearly, $\tau_0 \cdot \lambda_0 = \gamma$ from some λ_0 . Now, if there was a termination condition in \mathcal{E}' , then there will be a corresponding termination condition in the first-order chain being constructed. For example, if all *free*-literals of C_i' had been left-merged, then all the literals of C_1 would have been optionally merged too. Similarly, if a literal in C_i' formed a cycle, the corresponding literal(s) of C_1 would (after unification) still form a cycle. If there was no termination in \mathcal{E}' , then we extend the first-order chain using the same process. At each step, there will be a λ_i such that $\tau_i \cdot \lambda_i = \gamma$. The *free*-literals remaining in the first-order chain will map onto R' by means of the last λ, λ_r . \square

Theorem 4: A set S of clauses is unsatisfiable $\Leftrightarrow S \succ_{\text{lels}} \square$.

Proof: (\Leftarrow) As a consequence of the Herbrand theorem, there exists a finite unsatisfiable set S' of ground instances of S since S is unsatisfiable. Then, the theorem holds by lifting Theorem 2 and Theorem 3. (\Rightarrow) Clearly, first-order LELS is also sound. \square

Example 2: The following clause set is transformed from the premises: Custom officials searched everyone who entered this country who was not a VIP. Some drug pushers entered this country, and they were only searched by drug pushers. No drug pusher was a VIP. Conclusion: Some of the custom officials were drug pushers.

- | | |
|---|---|
| 1. $Pusher(a)$ | 2. $\neg Enter(x) \vee VIP(x) \vee \neg Custom(f(x))$ |
| 3. $Enter(a)$ | 4. $\neg Enter(x) \vee VIP(x) \vee Searched(x, f(x))$ |
| 5. $\neg Searched(a, y) \vee Pusher(y)$ | 6. $\neg Pusher(x) \vee \neg VIP(x)$ |
| 7. $\neg Pusher(x) \vee \neg Custom(x)$ | |

Solution by LELS: Note that clause 7 is the denial of the conclusion. That is, if someone is a drug pusher, then he is not a custom official. Using standard resolution methods, a proof was found in eight steps, and seven new clauses were generated and added back to the clause set. By applying LELS to this problem, the proof was found in two chains with $Pusher(a)$ as the START clause. Following is the constructed chain from C_0 to C_5 after renaming of the variables:

$$\begin{array}{l}
\text{Pusher}(a) \xrightarrow{R} \neg\text{Pusher}(x) \neg\text{VIP}(x) \\
\xrightarrow{R} \text{VIP}(y) \neg\text{Enter}(y) \text{Searched}(y, f(y)) \\
\xrightarrow{R} \neg\text{Searched}(a, z) \text{Pusher}(z) \\
\xrightarrow{R} \neg\text{Pusher}(w) \neg\text{Custom}(w) \\
\xrightarrow{R} \text{Custom}(f(u)) \text{VIP}(u) \neg\text{Enter}(u).
\end{array}$$

After the unification process, the most general unifier for (x, y, z, w, u) was found to be $(a, a, f(a), f(a), a)$ and $\text{RLST}(C_5) = \{\text{Pusher}(a), \neg\text{VIP}(a), \text{Searched}(a, f(a)), \text{Pusher}(f(a)), \neg\text{Custom}(f(a))\}$. $\neg\text{Enter}(a)$ of the END clause is a left-merged literal, and $\text{VIP}(a)$ is backward-deleted because $\neg\text{VIP}(a) \in \text{RLST}(C_5)$ and $\text{FREE}(C_5) = \{\neg\text{Enter}(a)\}$. The contradiction is then immediately found by linking $\neg\text{Enter}(a)$ to $\text{Enter}(a)$. We have proved the conclusion that some of the custom officials were drug pushers in two chaining steps, and only one new clause was generated.

4. RESULTS AND COMMENTS

The problems we experimented with are taken from group theory, Boolean algebra, and Puzzle problems. They were chosen because they have received repeated attention in the literature [1, 4, 8, 9]. The primary purpose of the experimentation was to investigate, if possible, the conditions under which LELS would perform better than other hyper methods. The second purpose was to determine a better way of exploring the search space. All of the experiments were performed on a Sparc-10 machine. The LELS-based automated theorem prover was implemented in C language.

The problem shown in the following is the ancestor problem. From the relationships among Alex, Beth, Carl, Elliott, David, and Frank, we seek to prove that Alex is Frank's ancestor. $\text{Parent}(x, y)$ means that x is y 's parent and $\text{Ancestor}(y, z)$ means that y is an ancestor of z . In our experiment, the empty clause can be reached by forming 3 chains if we use $\text{Parent}(\text{David}, \text{Frank})$ as the START clause. Only 3 new clauses were produced while other unlinked methods generated more than 10 clauses. The result and comparison are shown in Table 1.

1. $\text{Parent}(\text{Alex}, \text{Beth})$
2. $\text{Parent}(\text{Alex}, \text{Carl})$
3. $\text{Parent}(\text{Beth}, \text{Elliott})$
4. $\text{Parent}(\text{Carl}, \text{David})$
5. $\text{Parent}(\text{David}, \text{Frank})$
6. $\neg\text{Ancestor}(\text{Alex}, \text{Frank})$
7. $\neg\text{Parent}(x, y) \text{Ancestor}(x, y)$
8. $\neg\text{Parent}(x, y) \neg\text{Ancestor}(y, z) \text{Ancestor}(x, z)$

Table 1. A comparison of results of the ancestor problem.

Method	LELS	UR-resolution	Hyper-resolution
Start clause	5	-	-
Clause generated	3	13	10
Clause kept	11	21	16
Empty clause	1	1	1

Another problem we included comes from Group theory: *In a group, if the square of every element is the identity, the group is commutative.* The statement $P(x, y, z)$ will mean that the product of x and y is z , in the group being studied. a, b and c are constants. x, y and z are variables. e is identity. The results and a comparison are shown in Table 2.

1. $P(e, x, x)$	left identity
2. $P(x, e, x)$	right identity
3. $P(x, x, e)$	$x^2 = e$
4. $\neg P(x, y, u) \neg P(y, z, v) \neg P(u, z, w) P(x, v, w)$	associativity
5. $\neg P(x, y, u) \neg P(y, z, v) \neg P(x, v, w) P(u, z, w)$	associativity
6. $P(a, b, c)$	denial
7. $\neg P(b, a, c)$	denial

Table 2. A comparison of results of the group theory problem.

Method	LELS	UR-resolution	Hyper-resolution
Start clause	6	-	-
Clause generated	39	270	350
Clause kept	45	39	39
Empty clause	1	1	1

From the above results of the two selected problems, one can easily observe that LELS generates many less new clauses than do the other un-linked methods. The LELS inference rules clearly comprise a hyper-type inference system. Many researchers have listed potential benefits of hyper-type inference rules: larger inferences are made in a single step; fewer clauses are actually added to long-term memory; clause processing, such as demodulation and subsumption checks, are made less often and on potentially more “useful” clauses [3, 4, 9]. LELS is, however, quite different from other hyper methods. It does not have the syntactically oriented restrictions of positive/negative hyperresolution. UR resolution has similar restrictions [2, 10], which require that the end result be a unit clause. There are some aspects in common with the linking methodology because LELS can link in a chain through any number of 2-clauses (clauses with exactly 2 literals). However, unlike linked-UR resolution [4], LELS concentrates only on one selected literal of the START clause in a clause chain.

LELS explores the search space according to the termination conditions, the *free*-literals on hand and the current merging situations. LELS has several aspects in common with linear resolution and its variations. Any one chain is a linear deduction in which the selected literal always comes from the most recent side clause. However, by terminating a chain and forming a resolvent, we make a breadth-first component available to the search. LELS allows all chains starting from the original start clause to interact after they are generated. In this sense, the LELS method is a cross between pure linear [11] and set-of-support resolution [12]. As observed from the experimental results shown in Table 1 and Table 2, every newly generated clause became a candidate for a the future chaining targets (linking support). Table 3 lists the LELS preliminary experimental results.

Table 3. The preliminary LELS experimental results.

Tested Problems [3]	Clause used	Clause produced	Clause demodulated	Clause subsumed	Clause kept
t1(Horn)	9	4	0	2	9
t2(1)	8	4	0	9	3
School puzzle	15	15	0	16	15
Ancestor	8	3	0	0	11
G1(group)	7	39	0	2	45
G2(group)	9	124	6	69	64
G3(subgroup)	16	338	45	169	173
Sam's lemma	31	3689	0	3402	287
Salt & muster	63	1780	0	1039	741

5. CONCLUSIONS

In this paper, we have focused on the extended linking strategy with left merging for automated theorem proving. We have presented the detailed formalism and definition of the strategy. LELS is sound for any clause set. It has also been proved that LELS is refutation complete for any ground sets. We have also proved that LELS can be lifted and applied to first order calculus. The use of LELS can have a dramatically positive effect on program performance by occasionally reducing the searching time required to complete a proof by a large factor. Also, it can reduce the search space of a program because a vast amount of rarely needed information is avoided (not generated) during the linking process. We have included our experiment results for selected problems and found that LELS proves theorems by generating fewer new clauses than other un-linked methods. Compared to the restrictions of the existing reasoning rules, LELS has relatively higher control over which clauses the program focuses on. Finally, we suggest that LELS be used with the help of other heuristic strategies, such as weighting and targeting, to ensure the efficiency of the automated theorem proving programs.

ACKNOWLEDGMENT

This research was supported in part by Taiwan NSF grant NSC 84-2213-E-014-007.

REFERENCES

1. C. J. Ho, "Extended linking strategy for automated theorem proving," Ph. D. Thesis, Department of Electrical Engineering and Computer Science, Northwestern University, 1994.
2. L. Wos, D. Carson and G. Robinson, "The unit preference strategy in theorem proving," in *Proceedings of Fall Joint Computer Conference*, 1964, pp. 615-621.

3. J. A. Robinson, "Automatic deduction with hyper resolution," *International Journal of Computer Mathematics*, Vol. 1, No. 1, 1965, pp. 227-234.
4. L. Wos, R. Veroff, B. Smith and W. McCune, "The linked inference principle, II: The user's viewpoint," in *Proceedings of the 7th Conferences of Automated Deduction*, 1984, pp. 316-332.
5. C. L. Chang and R. C. T. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.
6. M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the Association for Computing Machinery*, Vol. 7, No. 3, 1960, pp. 201-215.
7. D. W. Loveland, "A simplified format for the model elimination procedure," *Journal of the Association for Computing Machinery*, Vol. 15, No. 2, 1969, pp. 349-363.
8. J. D. McCharen, R. A. Overbeek, and L. Wos, "Problems and experiments for and with automated theorem-proving programs," *IEEE Transaction on Computer*, Vol. C-25, No. 8, 1976, pp. 773-782.
9. N. Eisinger, H. J. Ohlbach and A. Pracklein, "Reduction rules for resolution-based systems," *Artificial Intelligence*, Vol. 50, No. 4, 1991, pp. 141-181.
10. L. Henschen and L. Wos, "Unit refutations and Horn sets," *Journal of the Association for Computing Machinery*, Vol. 21, No. 3, 1974, pp. 509-605.
11. R. Kowalski and D. Kuehner, "Linear resolution with selection function," *Artificial Intelligence*, Vol. 2, No. 3, 1971, pp. 227-260.
12. L. Wos, G. Robinson and D. Carson, "Efficiency and completeness of the set of support strategy in theorem proving," *Journal of the Association for Computing Machinery*, Vol. 12, No. 1, 1965, pp. 536-541.

Chuen-Hsuen Jeff Ho (何春勳) was born in Taichung, Taiwan, in 1957. He received the B.S. degree in Electrical Engineering from the Chung-Cheng Institute of Technology (CCIT), Ta-Hsi, Taiwan, in 1980, and the M.S. and Ph.D. degrees in Computer Science from Northwestern University (NWU), USA in 1987 and 1994, respectively. Since 1994, he has been an Associate Professor in the Department of Computer Science, CCIT. He was a Visiting Scholar in the Department of Electrical and Computer Engineering, NWU from July 1997 through December 1997. In the summer of 1998, he became the Chairman of the Department of Computer Science, CCIT. He received the Teaching Excellence Award from the Department of Defense in 1996 and the Research Excellence Award from CCIT in 1997. His current research interests include Theorem Proving, Meta-level Reasoning, and Military Expert Systems.