A MECHANISM FOR RESOLUTION REFINEMENTS
BASED ON BACK SUBSTITUTIONS
by
David M. Sandford

DCS-TR-77
#192

December 1980

Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, N.J.
ABSTRACT

Many of the known refinements of the first order resolution rule of J. A. Robinson suffer from various redundancies which have a common underlying cause, which we identify as "lack of ground faithfulness". We say a refinement is ground faithful iff for each deduction, D, in the refinement, there exists at least one "ground instance" of D which is also in the refinement. In this report it is shown how certain refinements which are not ground faithful can be modified, both in a theoretical and an implementation sense, into ground faithful versions of themselves. This modification yields a stronger refinement than the original refinement. In practice ground faithfulness can be achieved for many refinements by the addition to each clause of a notational device, called the constraint of the clause. The approach emphasizes that resolution clauses are the result of deductions, and the notational device constrains clauses based on information contained in the deductions of the individual clauses.

Also presented are some overall characteristics and properties both of refinements, and of the methods of defining refinements, which provides a common qualitative framework for understanding the various specific refinements presented.

ACKNOWLEDGMENTS

Part of this work was done while the author was a visiting assistant professor at the Laboratory for Computer Science Research at Rutgers University. I thank Professor C. V. Srinivasan of Rutgers for helpful comments on this material.
CONTENTS

Abstract........................................................................................................... 11
Acknowledgments......................................................................................... 11
List of Figures.............................................................................................. iv

1.0 Introduction.............................................................................................. 1
2.0 Theory and Definitions I........................................................................ 9
3.0 Examples of Redundancy Elimination Using HUSL's.......................... 24
4.0 Theory and Definitions II........................................................................ 36
5.0 Some Specific U-Clause Refinements.................................................... 47
6.0 Evaluating and Simplifying Constraints............................................... 60
7.0 Other Refinements and General Discussion.......................................... 72

Bibliography.................................................................................................. 89
Index of Definitions....................................................................................... 92
LIST OF FIGURES

Figure 1----------------------------------------------- 4
Figure 2----------------------------------------------- 5
Figure 3----------------------------------------------- 26
Figure 4----------------------------------------------- 30
Figure 5----------------------------------------------- 35
Figure 6----------------------------------------------- 48
Figure 7----------------------------------------------- 58
Figure 8----------------------------------------------- 59
Figure 9----------------------------------------------- 79
Just at this moment Alice felt a very curious sensation, which puzzled her a good deal until she made out what it was: she was beginning to grow larger again, and she thought at first she would get up and leave the court; but on second thoughts she decided to remain where she was as long as there was room for her.

"I wish you wouldn't squeeze so," said the Dormouse, who was sitting next to her. "I can hardly breathe."

"I can't help it," said Alice very meekly: "I'm growing."

"You've no right to grow here," said the Dormouse.

"Don't talk nonsense," said Alice more boldly: "you know you're growing too."

"Yes, but I grow at a reasonable pace," said the Dormouse: "not in that ridiculous fashion." And he got up very sulkily and crossed over to the other side of the court.

Lewis Carroll,
Alice's Adventures In Wonderland
1.0 Introduction.

This report assumes a working knowledge of resolution theory and nomenclature [Robinson, 1965a] [Chang and Lee, 1973] [Loveland, 1978], and presents a more complete discussion of ideas in [Sandford, 1977]. A notational mechanism (in two variant forms) is discussed which accomplishes two objectives:

1. allows a strengthening of many of the resolution refinements already known;
2. makes natural the expression of certain refinements not commonly known, or not ordinarily thought of as refinements.

The major advance in the state of the art in first order mechanical theorem proving which was provided by the resolution inference rule was that it coupled the instantiation process and the truth functional testing process through the notion of most general unifier [Robinson, 1965a]. The notion of the most general unifier, and the unification process, are fundamental to first order theorem proving, and have been incorporated subsequently into a variety of inference rules besides resolution. However, the unification process, as used in resolution and these other systems, is a strongly local operation. As a result it is often the case that a resolution search produces deductions which, when viewed globally, are seen to be obviously redundant (or unnecessary, or useless). In this report we will exhibit several examples of simple syntactic criteria by which we can generate global constraints (i.e. constraints over entire deductions) on the allowed substitution instances of variables. These constraints act as a restriction on the unification process in the
sense that certain unifications will be forbidden because they violate the constraints.

The central idea of this report is as follows.

Let the set of all ground instances of a clause be called the \textit{ground extension (set)} of the clause. We focus our attention on some arbitrarily chosen clause in a resolution search space, and we call it the target. Suppose the target has variable symbols in it. We retain the target in the search space iff some member of its ground extension set is of potential use in obtaining a refutation (the criteria of "potential use" may depend upon the particular refinement of resolution we are considering). Suppose the target is retained, and clause $C$ is some descendant of the target which contains variables which are "descendants" of variables in the target. In general it will be possible to see that only a subset of the ground extension set of the target contains possible ancestors of the elements of the ground extension set of $C$. Suppose that we are able to discern that in this subset of the target instances there are no ground instances of "potential use" in obtaining a refutation (how we can discern this is a major concern treated in this report). In such a case we can eliminate the clause $C$ from the search space. The reason is that, while the target is a potentially useful clause, the particular way it was used to derive $C$ did not incorporate any of its potentially useful instances. Notice that in general we cannot eliminate the target from the search, since it may have another descendant, $C'$, where the target may be contributing at least one of its potentially useful instances to the derivation of $C'$.

We will now illustrate this central idea by a specific set of initial clauses, given in Figure 1, and with Look Resolution (LR) [Boyer, 1971] as the particular refinement. Our notion of "potential use" will be that a clause is useful iff it is not a tautology. Our "target" for the example will be clause 5 of Figure 1.

In Figure 1 the lowest look numbered literal in each non-unit clause is underlined. At first sight the only thing wrong with the refutation in Figure 1 is that it is more complex, in depth and in
number of clauses, than the refutation obtained from using just clauses 1, 2 and 4 (which would give a depth 2 refutation). In Figure 2 we have the deduction tree corresponding to the refutation of Figure 1. At each node of Figure 2 we have written in the ground instance (unique, in this case) that the clause at that node represents in the deduction. Notice that the ground instance, 5g, of clause 5, is a tautology. Thus, by our explanation given above, we can say that this deduction of the empty clause has used clause 5 for the "wrong" instances. This is reflected by the fact that there is another refutation of clauses 1 through 5 which does not use clause 5 at all. Although LR is incomplete with tautology elimination, it is not hard to see that LR is complete if the only tautologies eliminated are initial clauses. When we have a situation, such as in Figure 2, where an initial clause is not itself a tautology, but its only use in a particular deduction is to represent tautologous ground instances, then we say that the clause, e.g. clause 5 of Figure 2, is a latent initial tautology in that particular deduction.

In this report we will treat both the theory and the practical aspects of a group of refinements. The elimination of latent initial tautologies is one member of this group. In Chapters 4 and 5 of this report it will be shown how, for the situation shown in Figure 2, it is possible to detect that in the formation of clause 6, we have already "thrown away" all potentially useful members of the ground extension set of clause 5. We now continue this introduction with some general statements about the background and environment in which the current work is to be interpreted.
1. $Q(a,b)^7$;
2. $\neg Q(b,a)^2$;
3. $Q(u,u)^3$;
4. $\neg Q(v,w)^4, Q(w,v)^9$;
5. $\neg Q(x,y)^7, \neg Q(y,z)^9, Q(x,z)^6$;

**level 1**

4x5 = 6. $\neg Q(x,y)^7, \neg Q(y,z)^9, Q(z,x)^6$;

**level 2**

6x1 = 7. $\neg Q(b,z)^9, Q(z,a)^8$;

**level 3**

2x7 = 8. $\neg Q(b,b)^9$;

**level 4**

8x3 = 9. \Box;

**FIGURE 1**

An LR-Refutation.
FIGURE 2

Refutation Tree Corresponding to Figure 1
With Unique Grounding Indicated.
There is precedent in resolution theory for an orientation toward looking at the possible ground instances of clauses (and entire deductions) with the objective of basing general level refinements on the nature of the ground instances. At the inception of resolution, Robinson [Robinson, 1965b] points out that for certain properties, the property will be true of a clause, \( C \), iff that property also is true for all of the ground instances of \( C \). Such a property is called an instantiation invariant of the clause, and examples are such properties as "is an empty clause", "contains no negative literals", etc. Similarly, an instantiation invariant of an entire deduction would be a property such as "is a refutation". Shortly thereafter, with the introduction of model based strategies [Slagle, 1967] [Luckham, 1970], we see a new type of property of clauses being used, namely that of truth value. Here a clause is said to have the property "false" relative to a model iff there exists at least one ground instance of the clause which has the property "false" relative to that model. More recently [Loveland, 1978, p. 110] there has been the identification of an interaction of the unifier used in a resolution step with ordering rules for literals in clauses. This interaction occurs as a result of the fact that when the parent clauses in an inference step are viewed after the unifier is applied to them, they then represent a more limited class of possible ground instances. In this limited class of ground instances the manifest literal ordering of the original parent clauses may be forbidden. In [Sickel, 1974] some substitution constraint criteria are presented which we will illustrate later in this report (in section 7.1.4).
The most important difference between previous work and what is presented here is that the emphasis here is on detecting the relevant ground instances of entire deductions, in distinction to doing this for individual clauses or for individual inference steps, coupled with the fact that we will concentrate not on instantiation invariants, but rather on properties of deductions which are true for only some of the ground instances that a deduction represents. This allows much stronger refinements to be stated and implemented, since, in effect, a clause does not represent a single set of ground instances, but rather different (and generally smaller) sets of ground instances for each deduction of which it is a part. The way in which this increased discrimination is implemented is by adding to clauses a new data structure which specifies a constraint on the substitution instances that the clause may represent. This constraint "remembers" certain relevant information about the deduction leading to a clause, and is in effect a mechanism for looking backwards at the deduction of a clause and "substituting" for variables in ancestors according to how the deduction has progressed after the use of those ancestors. Hence the title of this report.

The method of constraining clauses presented in this report is a direct outgrowth of two other techniques in resolution:

1. the constraints presented in this report are syntactic constraints which are analogous in several ways to a previously developed semantic constraint notion [Sandford, 1980];
2. the syntactic constraint method arose out of an attempt to strengthen some of the refinements of Overbeek et al. [Overbeek, 1976] which were originally implemented through the use of schemata.
There is some connection also to the global substitution constraints discussed in [Prawitz, 1969] and [Sickel, 1975], but the overall emphasis is different. This difference of emphasis is twofold: First we deal with constraints attached to explicitly generated clauses, and second we consider a variety of different criteria as the basis of forming such constraints.

This report is organized as follows. We have already seen an example of a redundancy of resolution caused by a latent initial tautology. In Chapter 2 we give the basic definitions and notions which will allow us, in Chapter 3, to give partial solutions to the problem of eliminating some redundancies, such as the latent initial tautology redundancy. In Chapter 4 further theory is presented which provides the basis for a strengthening of the refinements seen in Chapter 3, as well as some new applications, examples of which are discussed in Chapter 5. Chapter 6 discusses the practical aspects of implementing the methods presented. Chapter 7 is a general discussion of the overall approach, and briefly describes some additional refinement conditions.
2.0 **Theory and Definitions**

We are oriented towards refinements of resolution which are based more strongly on entire deductions rather than individual clauses. Since every deduction determines a clause (the deduced clause), but a single clause may have many deductions leading to it, it is obvious that deductions allow finer discriminations to be made than do clauses. In this chapter we give the basic definitions and notions that will be needed to exploit this greater discriminatory capability.

We use the following notational conventions. For common notions in resolution, such as "unifiable", which are not defined in this report, we will, without explicit definition, use a construction such as "(unifiable (11,12))" to mean "true" if 11 and 12 are unifiable, and to mean "false" otherwise. Individual variables in first order logic are denoted by lower case letters from among

\[ x, y, z, u, v, w, q, r, s, t \]

possibly with indices or prime marks, while individual constants are lower case letters earlier in the alphabet. Substitutions are denoted by \( \sigma \) or \( \tau \), and are represented by a set of substitution components in the usual fashion [Chang and Lee, 1973]. Often definitions are written which include existential conditions on substitutions such that a whole family of substitutions meet the stated conditions. In these cases we (usually implicitly) include the condition that the substitution is to be the **most general** substitution which meets the stated conditions. This most general substitution will always be unique except for its resultant variable names. For example, \( P(x) \) and \( P(y) \) are unified by any of \( \{a/x, z/y\}, \{z/x, z/y\}, \{x/y\} \), etc. The
first of these is not a most general unifier, while the second and third are each a most general unifier.

2.1 We assume the standard notions and nomenclature of finite, connected, rooted binary trees. Trees are rooted at the "bottom". For tree T, \text{root}(T) is the root node of T. By a branch is meant a complete branch, i.e. a linear subtree of T with the same root as T and extending to and including a leaf of T.

The internal nodes of a tree are all of the nodes except the leaf nodes. A tree is binary iff every node has at most two parents. A tree is a strict binary tree iff every internal node has exactly two parents.

If A, B and C are binary trees and N and N' are nodes, then \text{graft}(A,B,N) is the tree which is diagrammed as

\[
\begin{array}{c}
A \\
\downarrow \\
B \\
\downarrow \\
N
\end{array}
\]

and \text{stretch}(C,N') is the tree

\[
\begin{array}{c}
C \\
\downarrow \\
N'
\end{array}
\]

where the nodes N and N' are the roots of the new trees, and the links attach N to root(A) and root(B), and N' to root(C).

2.2 A clause tree is a finite, connected, rooted, binary tree with a clause assigned to each node. We will often just refer to the tree as having clauses as its nodes, it being understood that in such a case it means "occurrences of clauses" are the nodes. We assume that no
two leaf occurrences of clauses in a clause tree have any variable symbols in common. When appropriate a single clause, C, is considered to be the same as a clause tree whose only node is C.

2.3 The inference rules considered will for the most part be refinements of the original resolution rule [Robinson, 1965a], which we designate as UNrestricted Resolution (UNR). Thus deductions can be defined to be binary trees.

Let T be a clause tree. Then T is said to be a **UNR-resolution tree** (hereafter designated as either a resolution tree, or a deduction tree, or simply a deduction) iff each internal node with two parent nodes is a UNR-resolvent of those parents, and each internal node with one parent node is a UNR factor of the parent. If K is any superset (of clauses) of the set of leaf clauses of the resolution tree T, and J is the root of T, then T is said to be a deduction of J from K.

If we choose **binary resolution** [Chang and Lee, 1973] and factorizing as the two generating rules for clauses, then each deduction has one or two parents for each internal node (i.e. the deductions are not necessarily strict binary trees). This choice is called the **explicit factorizing** form of UNR. If factorizing is taken as part of a single generating rule, then the resulting deductions are strict binary trees. This choice is called the **implicit factorizing** form of UNR. There is no uniform advantage for either choice, so we make no commitment to either, and use whatever is convenient for the immediate context. Similar choices for factorizing also occur for various refinements of resolution.
2.4 For UNR resolution with implicit factoring, an inference step is an ordered triple of clauses, \(<C_1, C_2, C_3>\) such that \(C_3\) is a UNR-resolvent of \(C_1\) and \(C_2\) (i.e., \(\text{graft}(C_1, C_2, C_3)\) is a resolution tree).

2.5 For UNR resolution with explicit factoring, an inference step is either

i. an ordered triple of clauses, \(<C_1, C_2, C_3>\) such that \(C_3\) is a UNR binary resolvent of \(C_1\) and \(C_2\);

or

ii. an ordered pair of clauses \(<C_1, C_2>\), such that \(C_2\) is a UNR factor of \(C_1\).

For various refinements we can define inference steps analogously to that done for UNR.

2.6 For resolution tree \(T\), we say inference step \(I\) is in \(T\) iff either

i. \(I = \langle C_1, C_2, C_3 >\), and there exists nodes \(N_1\), \(N_2\) and \(N_3\) in \(T\) such that \(N_1\) and \(N_2\) are the parents of \(N_3\), and \(C_i\) is the clause labelling \(N_i\), for \(i = 1, 2, 3\);

or

ii. \(I = \langle C_1, C_2 >\), and there exists nodes \(N_1\), \(N_2\) in \(T\) such that \(N_1\) is the only parent of \(N_2\), and \(C_i\) is the clause labelling \(N_i\), for \(i = 1, 2\).
2.7 Major Subdeductions. Let $k$ be a node in the resolution tree $T$. If $k$ has only one parent, then the largest subtree of $T$ whose root is that parent of $k$ is the major subdeduction of $k$. If $k$ has two parents, then the two largest subtrees of $T$, each with a parent of $k$ as its root, are the two major subdeductions of $k$. If $k$ is a leaf node, then there are no major subdeductions of $k$.

2.8 $L$ will be the generic symbol for the first order language in which the initial set of clauses is written. $L$ is required to contain at least one individual constant symbol, but otherwise is to be the smallest language sufficient to express the initial set of clauses. Thus the set of ground terms of $L$ is precisely the Herbrand universe of the initial clause set.

We define

$$\text{desc}(L) = \langle \Delta(L), \Gamma(L) \rangle$$

as the description of $L$, where $\Delta(L)$ is the non-empty set of relation symbols of $L$, and $\Gamma(L)$ is the set of function symbols of $L$. We let arity be a function mapping each symbol of $\Delta(L) \cup \Gamma(L)$ to its degree (i.e. the number of terms that symbol uses as arguments). Function symbols of degree zero are constant symbols, and $\Gamma(L)$ contains at least one constant symbol.

In what follows assume that some specific language $L$ and set of initial clauses of $L$ have been chosen first, and then the definitions are made in this context. We assume $L$ does not use the symbol "$\exists$", since later we will want this to be an interpreted symbol of the language in which constraints will be written.
When it is necessary to view a clause (i.e. a set of literals of \(L\)), \(K\), as a sentence in \(L\), we assume it is in prenex form, with a pure universal prefix, is universally closed, and has a matrix which is the disjunction of the literals of \(K\). Thus the negation of a clause (set of literals), \(\overline{K}\), is a prenex form, closed, pure existential conjunction of the negations of literals in \(K\).

2.9 A clause is said to be ground (or a ground clause) iff none of its literals contains a variable symbol. The term general clause is used to emphasize that variables may be present. If \(C'\) is ground and is an instance of the clause \(C\), then we say \(C'\) is a grounding of \(C\). We say that \(\sigma\) is a minimal grounding substitution for \(C\) iff \(C\sigma\) is ground and every variable occurring in \(\sigma\) also occurs in \(C\). It is assumed that all terms involved in substitutions are terms of the language \(L\). When and where convenient (e.g. in the definition of \(G\) below) we take the liberty of treating clauses, particularly ground clauses, as multi-sets of literals. This allows a simpler connection between a clause and its ground instances (specifically as in the notion of "direct instance" [Boyer, 1971]).

A resolution tree, \(T\), is ground iff every clause in it is ground, and \(T\) is said to be general to emphasize that it is not necessarily ground. \(T\) is said to be in \(L\) iff all of its clauses are in \(L\).
2.10 We let $G$ be a function mapping general resolution trees to sets of ground resolution trees, defined as follows:

Let $T$ be a resolution tree,

1. If $T$ consists of a single clause (i.e. a single node), then $G(T)$ is the set of all 1-node resolution trees whose only clause is a grounding of the clause in $T$.

2. If root($T$) has only 1 major subdeduction, $T'$, i.e. $T = \text{stretch}(T', t)$, for $t = \text{root}(T)$, then $G(T)$ contains the ground resolution tree $\overline{T} = \text{stretch}(\overline{T'}, \overline{t})$, for each $\overline{T'}$ and $\overline{t}$ such that
   a. $\overline{T'} \in G(T')$,
   b. $\overline{t}$ is a grounding of $t$,
   c. $\overline{t}$ is a UNR factor of root($\overline{T'}$).

3. If root($T$) has two major subdeductions, such that $T = \text{graft}(T', T'', t)$, for $t = \text{root}(T)$, then $G(T)$ contains the resolution tree $\overline{T} = \text{graft}(\overline{T'}, \overline{T''}, \overline{t})$, for each $\overline{T'}$, $\overline{T''}$, $\overline{t}$ such that
   a. $\overline{T'} \in G(T')$ and $\overline{T''} \in G(T'')$,
   b. $\overline{t}$ is a grounding of $t$,
   c. $\overline{t}$ is a UNR-resolvent of root($\overline{T'}$) and root($\overline{T''}$).

Each $\overline{T}$ in $G(T)$ is called a grounding of $T$. $G$ is called the ground extension function, and $G(T)$ is called the ground extension of $T$.

2.11 For $S$ a set of clauses, define

$$G(S) = \bigcup \{ G(C) \mid C \in S \}.$$
2.12 A resolution tree $T$ is said to lift the resolution tree $T'$ iff there exists a 1-1 total mapping, $m$, from nodes of $T$ onto the nodes of $T'$ which preserves the parenthood relation, and such that for each clause $C$ in $T$, $m(C)$ is an instance of $C$. For any resolution tree $T$, $T$ lifts every member of $G(T)$.

2.13 We will denote the set of all resolution trees in $L$ by $\mathcal{T}_L$. A refinement, $\mathcal{R}$, over $L$ is a subset of $\mathcal{T}_L$. A deduction in $\mathcal{R}$ is called an $\mathcal{R}$-deduction or an $\mathcal{R}$-resolution tree. For a refinement to be of use in controlling search space growth it must be specified in some form that allows reasonably efficient testing of deductions to determine if they are or are not in the refinement. It will be useful to name and characterize some properties of the ways in which refinements can be specified, and properties of refinements themselves.

2.14 $\mathcal{C}$ is said to be a defining property for the refinement $\mathcal{R}$ over $L$ iff

$$\mathcal{R} = \{ T \mid (T \in \mathcal{T}_L) \land \mathcal{C}(T) \},$$


2.15 A refinement is said to be locally negatable iff for all $T$ in the refinement, the major subdeductions of $\text{root}(T)$ are also in the refinement. The importance of this property is that, in an actual search, a deduction which is formed and which is not in the refinement can be eliminated entirely from consideration; specifically we need not consider extending the deduction. It seems to be the case that all of the refinements known that are of utility are locally negatable.
2.16 A refinement is said to be **locally defined** iff the following holds:

there exists a one place relation, \( P \), defined on single inference steps (either resolution, or binary resolution and factoring steps) such that a deduction \( D \) is in the refinement iff \( P(I) \) is true for every inference step, \( I \), in \( D \).

Thus it would seem that a locally defined refinement cannot recognize global properties of deductions in the sense of how individual inference steps are put together to form the deduction. However, it will be obvious from the examples given later that, under suitable modifications of the notion of a clause and inference step, locally defined refinements do exist which are sensitive to global properties of deductions. The way that this occurs is that the modified clause contains within itself the necessary global information about the deduction tree of which it is the root.

Refinements such as resolution with merging \([\text{Andrews, 1968}]\), The Model Strategy (TMS) \([\text{Luckham, 1970}]\), and lock resolution \([\text{Boyer, 1971}]\) are all locally defined refinements.

\[N.B. \text{ We leave it to the reader to see in detail how } P \text{ (in the definition of "locally defined") is to be constructed, but point out that,}
\]

i. in TMS, \( P \) is dependent upon the Herbrand interpretation chosen as the model;

ii. in resolution with merging, and other refinements where initial clauses are treated in a special way, \( P \) must be relative to the choice of initial set;

iii. in resolution with merging, each clause must include attributes indicating whether it is a merge or not, and sufficient information to indicate which literals, if any, must be factored together.\]
An example of a non-locally defined refinement is the refinement consisting only of minimal deductions [Kowalski and Kuehner, 1971]. This is discussed in section 7.1.1. Notice that a locally defined refinement is of necessity locally negotable.

2.17 Suppose $S$ is a set of clauses, and $S'$ is the closure of $S$ under substitutions. A refinement, $R$, is said to be liftable on $S$ iff for every $R$-deduction from $S'$, there exists an $R$-deduction from $S$ which lifts it. A refinement is liftable over $L$ iff it is liftable on $S$ for every $S$ a set of clauses in $L$. A refinement is liftable iff for all $L$ it is liftable over $L$.

2.18 An existential ground based property, $E$, on deduction trees is a property on deduction trees which is true for a deduction, $T$, iff there exists a grounding of $T$, $T_g$, such that $E$ is true on $T_g$. If, in addition, $E$ is such that there exists no deduction, $T$, and grounding $T_g$ of $T$, such that $T$ has property $E$ and $T_g$ does not have property $E$, then $E$ is said to be insensitive to substitutions. Otherwise $E$ is substitution sensitive. The instantiation invariants of Robinson [Robinson, 1965b] are existential ground based properties which are insensitive to substitutions.

There are several hidden parameters included in the above definition of existential ground based property. The first is the language $L$. We assume that some language $L$ is known, and we are considering deduction trees $T$ which are in $L$, and groundings $T_g$ of $T$ which are also in $L$. Likewise, when we say "deduction trees", we may wish to qualify this in one, or both, of two possible ways:
i. the deduction trees $T$ could be restricted to be $\mathcal{R}$-deductions, for some chosen refinement $\mathcal{R}$;

ii. the deduction trees $T$ could be restricted to be deductions from some specified set of clauses, $S$, in $L$.

2.19 Let $\mathcal{R}$ be a refinement over $L$, i.e. $\mathcal{R}$ is a subset of the set of all resolution trees with leaves in $L$. $\mathcal{R}$ is said to be (existentially) ground faithful (over $L$) iff for every deduction, $T$, in $\mathcal{R}$, there exists a grounding of $T$ which is also in $\mathcal{R}$.

In this report we will often define refinements relative to an explicitly mentioned initial set of clauses. In such a case it is more appropriate to view the refinement as a family of refinements, $(\mathcal{R}^S_i)$, one for each initial set $S_i$. We then say that $\mathcal{R}^S_i$ is ground faithful over $S_i$ iff for each $T$ a deduction in $\mathcal{R}^S_i$, there exists at least one $\mathcal{T} \in G(T)$ such that $\mathcal{T} \in \mathcal{R}^S_i(S_i)$. Then $\mathcal{R}$ is said to be ground faithful over $L$ iff for each clause set $S$ in $L$, $\mathcal{R}^S$ is ground faithful over $S$.

If we specify a refinement, $\mathcal{R}$, by the defining property $E$, where $E$ is an existential ground based property of deductions, then $\mathcal{R}$ will be ground faithful. Conversely, if $\mathcal{R}$ is ground faithful, then there will always exist a defining property for $\mathcal{R}$ which is an existential ground based property.
example 2.19.1: Let TAUT be the refinement

\[ \text{TAUT} = \{ T \mid (T \in \tau) \land \phi(T) \} \]

where \( \phi(T) \) is true iff no clause in \( T \) contains a pair of
complementary literals, and \( \tau \) is given by

\[ \text{desc}(\tau) = \{ \{ \phi(-) \}, \{ a, b \} \} . \]

[N.B. In examples we indicate the arity of a symbol by the number of
dashes in the parentheses after the symbol.]

For our purposes here it doesn't matter if implicit or explicit
factoring is used. TAUT obviously is locally negatable and, if we
consider only initial sets with no tautologies, it is also locally
defined. We can show that there does not exist any existential
ground based property on deductions which can be used as the defining
property for TAUT, as follows. Assume such an existential ground
based property, \( E \), does exist. Then

\[ \text{TAUT} = \{ T \mid (T \in \tau) \land \phi(T) \} \]

Consider the clause \( q = \neg P(a), \neg P(b), P(x) \); and the deduction tree, \( T_q \),
consisting only of the node \( q \). \( T_q \) is an element of TAUT, since it is
a resolution tree of \( \tau \) and \( \phi(T_q) \) is true. Therefore \( E(T_q) \) must
be true also. But if \( E(T_q) \) is true, then \( E(g) \) must be true for some \( g \) an
element of \( G(T_q) \). Thus

\[ G(T_q) = \{ \{ \neg P(a), \neg P(b), P(x) \}, \{ \neg P(a), \neg P(b), P(b) \} \} . \]

Thus \( E \) is true for at least one \( g \), where \( g \) is a resolution tree
consisting of a single ground clause of \( \tau \) which has a complementary
pair of literals. Therefore this \( g \) is in TAUT. But \( \phi(g) \) is false,
and thus \( g \) cannot be in TAUT. Thus there cannot exist such an
existential ground based property, \( E \). This also implies that TAUT is
not ground faithful. Note that if \( \Gamma(L) = \{ a, b, f(-) \} \), instead of just
\( \{ a, b \} \), then the above argument could not be carried out with such a
simple deduction as \( T_q \). However it is easily shown that for any
choice of language \( \tau \), the refinement TAUT as defined in this example
is not ground faithful.

example 2.19.2: The Model Strategy. We assume implicit factoring for
this example. TMS has the defining property \( \phi_m : \)

\[ \phi_m(T) = \text{true iff each non-leaf node in } T \text{ has at least one parent} \]

clause false in \( M \),

where \( M \) is some arbitrarily chosen Herbrand interpretation for the
language \( L \). TMS is locally negatable and locally defined, but is not
ground faithful in general. In the special case of homogeneous
settings (models) [Loveland, 1978], TMS is ground faithful. This
occurs because the property "truth value with respect to a homogeneous
setting" is an instantiation invariant of clauses. In [Sandford, 1980]
it is shown how TMS can be made ground faithful for arbitrary \( L \) and
arbitrary Herbrand interpretations.
2.20 Many refinements of resolution which have been proposed are specified by defining properties which are not existential ground based properties. These refinements are usually shown to be refutation complete in general by first showing them to be refutation complete at the ground level, and then showing that they are liftable. When this is the case, then it is possible to define a new refinement whose defining property is itself defined in terms of the original defining property, and the new refinement will be ground faithful (because the new defining property will be an existential ground based property).

**THEOREM**

Suppose \( R \) is a refinement over \( L \) which is refutation complete for all ground sets of \( L \), and is liftable over \( L \), and suppose that \( \Phi \) is a defining property for \( R \). Define a new refinement, \( R' \), by

\[
R' = \{ T \mid (T \in \mathcal{F}_L) \land E(T) \}
\]

where

\[
E(T) = \text{true iff } \exists t : t \in G(T) \land \Phi(t).
\]

Then \( R' \) will be refutation complete and ground faithful.

The proof of this is obvious. Notice that if \( \Phi \) was already an existential ground based property, then \( R' \) is exactly \( R \). It is easy to show that if \( R \) was locally negatable, then \( R' \) will be locally negatable. However, \( R \) being locally defined does not guarantee that \( R' \) will be a locally defined refinement. Also note that \( R' \) is a subset of \( R \). The analogous theorem and remarks hold for refinements \( R^S \) relativized to an explicitly mentioned initial set.
Using the above theorem is one way of obtaining a strengthened refinement, $R'$, of a known refinement, $R$, such that $R'$ is guaranteed to be refutation complete if $R$ is refutation complete, and $R'$ will be ground faithful. In order to implement $R'$ however, some effective way of evaluating $E(T)$ is required for $T$ a general resolution tree. Depending upon the nature of the defining property for $R$, the property $E$ may be very easy to detect in a deduction $T$, or it may be quite difficult. Most of the remainder of this report is concerned with two mechanisms which may be employed in practice to determine if a particular resolution tree has the property $E$ or not. These mechanisms attempt, with varying degrees of success, to modify the notion of a clause so that the new refinement $R'$ is not only ground faithful, but also locally defined and computationally feasible to implement. The first mechanism, called HUSL constraints, is an approximate method, and is presented next. The name "HUSL" stands for Herbrand Universe Syntactic List.

2.21 We define a HUSL-clause, $C$, to be an ordered pair of sets of literals

$$C = < S(C), U(C) >$$

where $S(C)$ is the set of standard literals of $C$, and is a non-empty set of literals of $L$ (possibly look literals). $U(C)$ is the set of HUSL literals of $C$, and is a finite set of literals, each of the form

$$\neg S((t1, t2, \ldots, tn),(s1, s2, \ldots, sn))$$

where each $t_i$ and $s_i$ is a term of $L$, and $n$ may vary for different elements of $U(C)$. We call this set of HUSL literals of $C$ the HUSL of $C$, or the constraint of $C$. 
For a HUSL-clause, C, and substitution \( \sigma \),

\[ C\sigma = \langle (\mathcal{L}(C))\sigma, (\mathcal{U}(C))\sigma \rangle. \]

2.22 Let \( X = \neg \mathcal{S}((t_1, t_2, \ldots, t_n), (s_1, s_2, \ldots, s_n)) \) be a HUSL
literal. Define \( \text{size}(X) \) to be \( n \). We define \text{infeasibility} of \( X \) by

\[ (\text{infeasible } X) \]

iff

\[ (\forall i: (1 \leq i \leq \text{size}(X)) \implies t_i \neq s_i) \]

where "\( t_i \neq s_i \)" is defined to hold iff \( t_i \) is syntactically identical

to \( s_i \).

Also we define

\[ (\text{feasible } X) \]

iff

\[ \neg (\text{infeasible } X). \]

\text{example:} \quad \neg \mathcal{S}(\langle a, f(x) \rangle, \langle a, f(y) \rangle) \text{ is feasible},

while \quad \neg \mathcal{S}(\langle a, f(x) \rangle, \langle a, f(y) \rangle) \text{ is infeasible.}

2.23 A HUSL-clause, \( C \), is said to be \text{feasible} (and also \( \mathcal{U}(C) \) is said
to be feasible) iff each HUSL literal in \( \mathcal{U}(C) \) is feasible. \( C \) is
\text{infeasible} (also \( \mathcal{U}(C) \) is infeasible) iff \( C \) is not feasible. If \( \mathcal{U}(C) \)
is empty then \( \mathcal{U}(C) \), and \( C \), are feasible. Obviously, the feasibility
of a HUSL-clause, \( C \), can be determined in time linear with respect to
the length of \( \mathcal{U}(C) \).
3.0 Examples of Redundancy Elimination Using HUSL's.

In this chapter we use the HUSL-clause notion to define three refinements. It will be seen that the use of HUSL-clauses does eliminate some unnecessary deductions, but that the HUSL-constraint is not strong enough to achieve ground faithfulness for the refinements considered here. The HUSL-clause notion is of interest because of the simplicity of feasibility testing of HUSL constraints. In Chapter 4 we will present the stronger notions required to achieve ground faithfulness, both for the three example refinements of this chapter and for some other refinements. The definitions of refinements here and later will be phrased in terms of recursive definitions of deduction trees. We leave implicit in each of these definitions the exclusivity condition, which is that each refinement is to be the smallest set of deductions satisfying the explicitly stated conditions in the definitions.

3.1 Our first example is that of eliminating the latent initial tautologies for LR. The first definition we need is one that will be used several times later.

3.1.1 Define distinctness.l literal to be a function from ordered pairs of literals to HUSL literals as follows. Let P, Q be relation symbols of L such that arity(P) = arity(Q). Let k1 = P(t1, ..., tn) and k2 = Q(s1, ..., sn) be atoms of L. Then

\[ \text{distinctness.l literal}(k1, k2) = \neg \varepsilon((t1, ..., tn), (s1, ..., sn)) \]

i.e., the two lists of arguments of the atoms k1 and k2 become the arguments of "\( \neg \varepsilon \)" in the distinctness literal for k1 and k2. For x
and \( y \) literals of \( L \) with the same number of terms,
\[ \text{distinctness.literal}(x,y) = \text{distinctness.literal}(|x|,|y|), \]
where \(|z|\) is the atom of the literal \( z \).

3.1.2 Define \( \Phi_{\text{HUL}} \) to be a function on sets of literals, \( K \), such that
\[
\Phi_{\text{HUL}}(K) = \{ \text{distinctness.literal}(k_1,k_2) \mid (\text{atom } k_2) \\
\wedge k_1 \in K \wedge k_2 \in K \wedge (\text{unifiable } (k_1,\sim k_2)) \}.
\]

If \( K \) is a set of lock literals, we assume that the lock numbers are invisible for the purposes of defining \( \Phi_{\text{HUL}} \).

Example: for \( K = \{ \neg P(x,y,z), \neg P(z,u,v), \neg P(y,u,w), P(x,w,v) \} \)
\[
\Phi_{\text{HUL}}(K) = \{ \neg E((x,y,z),(x,w,v)), \neg E((z,u,v),(x,w,v)) \},
\]
\[
\neg E((y,u,w),(x,w,v)) \}
\]

3.1.3 Suppose \( S \) is a set of lock resolution clauses. Let \( ^{\prime} S_{\text{HUL}} \) be the set of HUSL-clauses:
\[
^{\prime} S_{\text{HUL}} = \{ C \mid \exists C' : C' \in S \wedge C = C', \Phi_{\text{HUL}}(C') \wedge (\text{feasible } C) \}.
\]

3.1.4 We will assume factoring is built into the LR resolution rule for the present purposes. Then we say that HUSL-clause \( C_3 \) is an LR direct \( H \)-resolvent of HUSL-clauses \( C_1 \) and \( C_2 \) iff there exists a \( \sigma \):

1. \( S(C_3) \) is an LR-resolvent of \( S(C_1) \) and \( S(C_2) \), with resolution unifier \( \sigma \);
2. \( U(C_3) = U(C_1)\sigma \cup U(C_2)\sigma \);
3. \( C_3 \) is feasible.

By analogy we define other types of direct \( H \)-resolvents. For example, a UNR direct \( H \)-resolvent of \( C_1 \) and \( C_2 \) would be as above, except that in condition 1 "UNR" replaces "LR".
4. $\neg Q(v, w), Q(w, v)$; HUSL = \{ $\neg \in((v, w), (w, v))$ \}

5. $\neg Q(x, y), \neg Q(y, z), Q(x, z)$;
   HUSL = \{ $\neg \in((x, y), (x, z)),$
             \hspace{1cm}$\neg \in((y, z), (x, z))$ \}

6. $\neg Q(x, y), \neg Q(y, z), Q(z, x)$;
   HUSL = \{ $\neg \in((x, z), (z, x)),$
             \hspace{1cm}$\neg \in((z, x), (x, z)),$
             \hspace{1cm}$\neg \in((y, z), (x, z))$ \}

1. $Q(a, b)$; HUSL = \{ \}

7. $\neg Q(b, z), Q(z, a)$;
   HUSL = \{ $\neg \in((a, b), (z, a)),$
             \hspace{1cm}$\neg \in((a, b), (a, z)),$
             \hspace{1cm}$\neg \in((b, z), (a, z))$ \}

2. $\neg Q(b, a)$; HUSL = \{ \}

8. $\neg Q(b, b)$;
   HUSL = \{ $\neg \in((b, b), (b, b)),$
             \hspace{1cm}$\neg \in((a, b), (a, b)),$
             \hspace{1cm}$\neg \in((b, b), (a, b))$ \}

### FIGURE 3a

An Infeasible HUSL-Clause Tree
(compare with Figure 2).

---

1. $Q(a, b)$; HUSL = \{ \}

   A. $\neg Q(v, w), Q(w, v)$; HUSL = \{ $\neg \in((v, w), (w, v))$ \}

   B. $Q(b, a)$; HUSL = \{ $\neg \in((a, b), (b, a))$ \}

   2. $\neg Q(b, a)$; HUSL = \{ \}

   B. \{ \}; HUSL = \{ $\neg \in((a, b), (b, a))$ \}

### FIGURE 3b

An "LR-FAULT Refutation."
3.1.5 Then we can define the refinement \( \text{LR}^S \) as:

for initial set \( S \),

1. each clause in \( S_{\text{taut}} \) is a deduction in \( \text{LR}^S \);

2. for deductions \( D_1 \) and \( D_2 \) in \( \text{LR}^S \), and \( C \) an LR direct

\( \eta \)-resolvent of \( \text{root}(D_1) \) and \( \text{root}(D_2) \), the deduction

\( \text{graft}(D_1, D_2, C) \) is in \( \text{LR}^S \).

[N.B. Our symbolic notation for refinements is as follows: the notation \( R^z \) means that the basic refinement is \( R \), \( z \) is the addition to \( R \) which requires a constraint list to be added to the clause, \( x \) is either \( \eta \) (for HUSL constraints) or \( U \) (for the constraints to be considered later), and \( y \) names the initial set of clauses (and therefore, implicitly, the relevant language, L).]

3.1.6 Figure 3a shows a clause tree (using HUSL-clauses) corresponding to the deduction of clause 8 in Figures 1 and 2. The upper part of the tree in Figure 3a is a clause tree with clause 7 at its root. This upper part is a \( \text{LR}^S \) deduction of clause 7 from \( S_{\text{taut}} \) (as in definition 3.1.3, where \( S \) is the set of five initial clauses shown in Figure 1). The bottom of the entire clause tree combines clauses 7 and 2 to produce clause 8, but it is seen that clause 8 is infeasible, so that the entire tree is not a deduction in \( \text{LR}^S \). Thus \( \text{LR}^S \) does not contain a deduction which corresponds (in an obvious sense of corresponds) to the refutation shown in Figures 1 and 2. However, \( \text{LR}^S \) does contain the refutation shown in Figure 3b. The completeness of \( \text{LR}^S \) for any initial set \( S' \), should be obvious. However, it is not ground faithful. From Figure 3a it is seen, simply by trying all possibilities, that there is no way to ground the deduction of clause 6 so that neither clause 4 nor 5 in the
grounding is a tautology. This arises because of the limited Herbrand universe available for the initial set of five clauses in Figure 1, namely \( \{a, b\} \). Looking at the HUSL of clause 6,

\[
\mathcal{U}(\text{clause 6}) = \{\neg\mathcal{E}(x, z), (z, x), \neg\mathcal{E}(x, y), (x, z), \neg\mathcal{E}(y, z), (x, z)\}
\]

we see (again by simply trying all 8 possibilities) that no assignment of constants to \( x, y, \) and \( z \) from \( \{a, b\} \) will give a grounding of the HUSL with all literals feasible. The reason that the HUSL of clause 6 is feasible is both because our definition of feasibility looks at each HUSL literal individually and does not depend upon the specific language \( L \). What happens is that different HUSL literals become infeasible (i.e. contain identical argument lists) under the various assignments of \( a \) and \( b \) to \( x, y, \) and \( z \), but no individual HUSL literal is infeasible for every one of its ground instances. Thus the simple definition of a HUSL, and of its feasibility, which we have used is not adequate to yield ground faithfulness, even for such a simple refinement as \( \mathcal{U}_2, \mathcal{E}_2 \). On the other hand, HUSL feasibility as we have defined it does permit both rapid evaluation and easy simplification of a HUSL (to reduce the storage requirement). For example, the HUSL of clause 7 in Figure 3a can be reduced to

\[
\mathcal{H}' = \{\neg\mathcal{E}(a, z), \neg\mathcal{E}(b, z)\}
\]

according to obvious rules of reduction, and these rules guarantee that \( \mathcal{H}' \) is feasible iff \( \mathcal{U}(\text{clause 7}) \) is feasible. Note that such a reduction from \( \mathcal{U}(\text{clause 7}) \) to \( \mathcal{H}' \) is independent of the Herbrand universe, i.e. we do not have to know the Herbrand universe of the initial set in order to assert that

\[
(\text{feasible } \mathcal{U}(\text{clause 7})) \iff (\text{feasible } \mathcal{H}').
\]
We now continue this chapter with definitions of two other refinements using our simple notion of HUSL's.

3.2 Analogous to latent tautologies in initial clauses there can also be latent tautologies in derived clauses. Since the elimination of derived tautologies makes LR incomplete, we will treat the case of latent derived tautologies in the context of UNR. We assume implicit factoring. The refinement $^{s}$UNR$_{taut}$ is given as follows.

3.2.1 The HUSL-clause $C_3$ is a UNR$_{taut}$-resolvent of HUSL-clauses $C_1$ and $C_2$ iff

there exists a $\sigma$:

1. $S(C_3)$ is a UNR-resolvent of $S(C_1)$ and $S(C_2)$, with resolution unifier $\sigma$;
2. $U(C_3) = (U(C_1))_\sigma \cup (U(C_2))_\sigma \cup \{^s\text{UNR}_t S(C_3)\}$;
3. $C_3$ is feasible.

3.2.2 For $S$ an initial set of clauses, the refinement $^{s}$UNR$_{taut}$ is defined by

1. each clause in $^{s}$S$_{taut}$ is a deduction in $^{s}$UNR$_{taut}$;
2. for $D_1$, $D_2$ deductions in $^{s}$UNR$_{taut}$ and $C$ a UNR$_{taut}$-resolvent of root($D_1$) and root($D_2$), the deduction graft($D_1,D_2,C$) is in $^{s}$UNR$_{taut}$.
1. \( \neg P(x,y), T(x,y), EQ(x,y); \)  \( \text{HUSL} = \{\} \)
2. \( \neg S(u,v), P(u,v); \)  \( \text{HUSL} = \{\} \)
3. \( \neg P(r,s), Q(a,f(s,r)); \)  \( \text{HUSL} = \{\} \)
4. \( T(z,z); \)  \( \text{HUSL} = \{\} \)
5. \( \neg EQ(x_1,y_1), S(y_1,x_1); \)  \( \text{HUSL} = \{\} \)
6. \( P(x_2,y_2), R(g(x_2,y_2)); \)  \( \text{HUSL} = \{\} \)

\[ \text{FIGURE 4a} \]

An Initial HUSL set for \( \text{HNR}^\gamma_{\text{tau}} \).

\[ \text{FIGURE 4b} \]

Latent Generated Tautology (clause 8).
Figure 4a lists an initial set of HUSL-clauses, $S_{\text{mut}}$. Figure 4b is a deduction which is not in $H_{\text{NR}_{\text{mut}}}$, but both of its major subdeductions are. The final inference step in Figure 4b involves a unifier which requires that clause 8 represent only ground clauses which are tautologies. This is reflected in the infeasibility of clause 11. For this particular example, the HUSL constraints actually have detected the latent generated tautology at the earliest possible point. In general, however, the same remarks about lack of ground faithfulness for $H_{\text{LR}_{\text{mut}}}$ also apply to $H_{\text{NR}_{\text{mut}}}$.

3.3 Our last example of this chapter concerns using HUSL’s to keep track of factoring alternatives. We use LR with explicit factoring (and allow factoring only among literals of the lowest lock number in a clause) as the particular example case, but a similar thing can be done in almost any other refinement.

3.3.1 For $K$ a set of unifiable literals, we let $\text{mgu}(K)$ be the most general unifier of $K$. Thus $(K)\text{mgu}(K)$ is a singleton set.

3.3.2 For $C$ a lock clause, let $\text{low}\text{-lock}(C)$ be the set of literals in $C$ of lowest lock number in $C$.

3.3.3 For lock clauses $C$ and $C'$, we say that $C'$ is the $k$-factor of $C$ if and only if there exists a $\sigma$:

1. $k$ is a non-empty subset of $\text{low}\text{-lock}(C)$;
2. $(\text{unifiable } k)$;
3. $\text{mgu}(k) = C'$;
4. $C' = C_0'$. 
3.3.4 For a HUSL-clause $C$, whose standard literals have lock numbers, let $\text{low.lock}(C) = \text{low.lock}(\mathcal{S}(C))$. The HUSL-clause $C'$ is the $k$-factor of $C$ iff there exists a $\sigma$:

1. $k$ a non-empty subset of $\text{low.lock}(C)$;
2. (unifiable $k$);
3. $\text{mgu}(k) = \sigma$;
4. $C' = \langle \sigma, (\mathcal{U}(C)\sigma \cup \kappa)\rangle$,
where $s$ is the $k$-factor of $\mathcal{S}(C)$, and
\[
\kappa = \{ x \mid x = \text{distinctness.literal}((k)\sigma, (q)\sigma) \\
\land q \in (\text{low.lock}(C) - k) \}
\]
5. (feasible $C'$).

3.3.5 A HUSL-clause $C3$ is said to be a binary LR direct $H$-resolvent of HUSL-clauses $C1$ and $C2$ iff there exists a $\sigma$:

1. $\mathcal{S}(C3)$ is a binary LR-resolvent of $\mathcal{S}(C1)$ and $\mathcal{S}(C2)$, with resolution unifier $\sigma$;
2. $\mathcal{U}(C3) = (\mathcal{U}(C1))\sigma \cup (\mathcal{U}(C2))\sigma$;
3. $C3$ is feasible.

3.3.6 Given initial set $S$, define the $\mathcal{H}_{\text{LR фак}}$ initial set, $S_{\text{фак}}$:

$$S_{\text{фак}} = \{ \langle \emptyset, \emptyset \rangle \mid C \in S \}.$$
3.3.7 The refinement $^{\mathcal{H}R}_F$ is defined by,

for initial set $S$,

1. Each clause in $S_F$ is a deduction in $^{\mathcal{H}R}_F$;

2. For each deduction $D$ in $^{\mathcal{H}R}_F$, such that root($D$) has either two major subdeductions or no major subdeductions, if for some $k$, $C$ is a $k$-factor of root($D$), then stretch($D, C$) is a deduction in $^{\mathcal{H}R}_F$.

3. For $D_1$ and $D_2$ two deductions in $^{\mathcal{H}R}_F$, each having exactly one major subdeduction of their root nodes, and $C$ a binary $LR$ direct $H$-resolvent of root($D_1$) and root($D_2$), the deduction graft($D_1, D_2, C$) is in $^{\mathcal{H}R}_F$.

Figure 5 gives the entire refinement $^{\mathcal{H}R}_F$ for the initial set, $S$, consisting of three clauses, $C_i = S$(clause $i$ in Figure 5), for $i = 1, 2, 3$. At level 1 there are 3 $k$-factors, one for each initial clause. The value of $k$ for each clause is written at the right. Because each level 0 clause has a unique lowest lock numbered literal, all of the $k$-sets at level 1 are singleton sets, and the clauses at level 1 all have empty $HUSL$'s. All of the $HUSL$ literals that appear in Figure 5 come from the factoring choices made at level 3 in forming $k$-factors of clause 7. The $k$-factor with $k = \{B(y)\}$ does not appear at level 3 because it gives a clause identical to clause 8. Although the refinement is defined in terms of deductions, it is the case that, since the $k$-sets are not actually part of the deductions, that the deduction of clause 8 and the deduction of the other $k$-factor not written at level 3 are in fact identical deduction trees. Thus this
k-set factoring choice does not appear in level 3. Thus Figure 5 is the entire refinement $^S_{\text{LR}_{FAc}}$ (plus two deductions not in the refinement which are marked infeasible).

Actually, $^S_{\text{LR}_{FAc}}$ is not quite as strong a refinement as is possible for factoring. Specifically, there is no mechanism to ensure that the literals factored on are the literals next resolved away, which is a completeness preserving restriction. This additional restriction is easily added. With or without this restriction, $^S_{\text{LR}_{FAc}}$ is not ground faithful for arbitrary S, and the reason is the same as discussed previously.

The next chapter gives the basic definitions needed to utilize a more powerful type of constraint on a clause. This more powerful constraint is still syntactically oriented, like the HUSL-constraint, but accomplishes two things:

1. the HUSL refinements already stated can be strengthened;
2. some refinements which are not expressible at all by using HUSL-constraints will be expressible with the new type of constraint.
level 0
1. $A(x)' , B(y)^3$;  $\text{ HuluL} = \{\}$
2. $\neg A(a)^2 , B(f(b))^3$;  $\text{ HuluL} = \{\}$
3. $\neg B(f(b))^4$;  $\text{ HuluL} = \{\}$

level 1
$k$-factor of 1=4. $A(x)' , B(y)^3$;  $\text{ HuluL} = \{} k = \{A(x)'\}$$
k$-factor of 2=5. $\neg A(a)^2 , B(f(b))^3$;  $\text{ HuluL} = \{} k = \{\neg A(a)^2\}$$
k$-factor of 3=6. $\neg B(f(b))^4$;  $\text{ HuluL} = \{} k = \{\neg B(f(b))^4\}$

level 2
$4x5=7$. $B(f(b))^3 , B(y)^3$;  $\text{ HuluL} = \{\}$

level 3
$k$-factor of 7=8. $B(f(b))^3 , B(y)^3$;  $\text{ HuluL} = \{} k = \{\neg A((f(b)),(y))\}$
$k$-factor of 7=9. $B(f(b))^3$;  $\text{ HuluL} = \{} k = \{B(f(b))^3 , B(y)^3\}$

level 4
$8x6-10$. $B(y)^3$;  $\text{ HuluL} = \{} k = \{\neg A((f(b)),(y))\}$
$8x6-11$. $B(f(b))^3$;  $\text{ HuluL} = \{} k = \{\neg A((f(b)),(f(b)))\}$ infeasible
$9x6=12$. $\Box$;  $\text{ HuluL} = \{\}$

level 5
$k$-factor of 10=13. $B(y)^3$;  $\text{ HuluL} = \{} k = \{B(y)^3\}$

level 6
$13x6=14$. $\Box$;  $\text{ HuluL} = \{} k = \{\neg A((f(b)),(f(b)))\}$ infeasible

FIGURE 5
The Refinement $\text{ HLRFAC}$.
4-0 Theory and Definitions II.

In this chapter we replace the NUSL of a clause with a first order sentence. This gives a new type of clause which we call a U-clause. U-clauses in theory achieve ground faithfulness for strategies such as the elimination of latent initial tautologies, as well as facilitating the statement of refinements which are not expressible at all using just the NUSL-clause notion. In practice the U-clause requires feasibility testing which in general is more difficult to perform than that required for NUSL-clauses. We do not give complete algorithms for the general case of U-clause feasibility testing, as these have not been fully developed as yet. However, some discussion of the feasibility testing on U-clauses is given in Chapter 6, and algorithms for some special cases are given. Somewhat surprisingly, for several refinements of practical interest, the constraint evaluation effort is linear in the size of the constraint when the relevant language contains at least one function symbol of degree greater than zero.

4.1 Given a first order language, L, we use the notation Lu to designate the first order language given by

\[ \text{desc}(L_u) = \langle \{\#(-, -)\}, \Gamma(L) \rangle \]

i.e. Lu is the language with the same functional vocabulary as L, and having only a single two place relation symbol, "\#".

4.2 We define a total semantic function, \( \phi_{u} \), mapping (sentences in) Lu onto the set {true, false}, such that for ground atom \( \#(t_1, t_2) \) of Lu,

\[ \phi_{u}(\#(t_1, t_2)) = \text{true iff } t_1 \text{ and } t_2 \text{ are the same term of } L_u, \]
and the truth values of other sentences in $L_u$ are determined in the standard way starting from this basis, and taking the set of ground terms of $L_u$ as the universe.

$\phi_{e_u}$ is the semantic function for what we call the standard theory of syntactic equality on the Herbrand universe of $L$ (which is also the Herbrand universe of $L_u$). This is obviously a complete theory, and there is a unique Herbrand interpretation, $h_{x_u}$, which satisfies this theory:

$$h_{x_u} = \{ \overline{u}(ti, ti) \mid ti \text{ a ground term of } L_u \} \cup \{ \overline{s}(ti, tj) \mid ti, tj \text{ different ground terms of } L_u \}.$$ 

We say that $\phi_{x_u}$ is the semantic function induced by $h_{x_u}$.

**Example:** Let $\text{desc}(L) = \langle \{ P(-), Q(-,-) \}, \{ a, b \} \rangle$,

then $\text{desc}(L_u) = \langle \{ \overline{e}(-,-), \overline{a}(a, b) \rangle$, 

$$h_{x_u} = \{ \overline{e}(a, a), \overline{e}(b, b), \overline{e}(a, b), \overline{e}(b, a) \}.$$ 

**Example:** Let $\text{desc}(L) = \langle \{ P(-), Q(-,-) \}, \{ a, b, g(-,-) \} \rangle$,

then $\text{desc}(L_u) = \langle \{ \overline{e}(-,-), \overline{a}(a, b), \overline{g}(a, b) \rangle$, 

$$h_{x_u} = \{ \overline{e}(a, a), \overline{e}(b, b), \overline{e}(a, b), \overline{e}(b, a), \overline{g}(g(a, a), g(a, a)), \overline{g}(g(b, b), g(b, b)) \}.$$ 

### 4.3 U-clause $C$, in the language $L$, is an ordered pair $C = \langle S(C), U(C) \rangle$

where $S(C)$ is a set of literals of $L$, called the standard literals of $C$, and $U(C)$ is a sentence of $L_u$, called the U-constraint sentence (i.e. Herbrand Universe constraint sentence) of $C$. The U-clause replaces the previous notion of HUSL-clause, and the U-constraint sentence is the strengthening of the previous notion of the HUSL set of literals. $U(C)$ is also called the constraint, or constraint
sentence of $C$, and intuitively is to be thought of as being a conjunction of sentences which always has at most only one conjunct sentence. Thus, if $U(C)$ is empty, it is understood to be "true".

If $U(C)$ is empty then we say that $C$ is unconstrained, otherwise $C$ is said to be constrained.

$S$ is extended to sets of U-clauses, $X$:

$$S(X) = \{ S(k) \mid k \in X \}.$$  

4.4 We say that a U-constraint sentence, $u$, is feasible (in $L$, or in $L_u$) iff $\phi_u(u) = \text{true}$. A U-clause, $C$, is feasible iff $U(C)$ is feasible. A U-clause or a U-constraint is infeasible iff it is not feasible. A U-clause is feasible if its U-constraint is empty.

4.5 The definition for U-clause trees is the obvious analogue to that for clause trees. If $T$ is a U-clause tree, then $S(T)$ is the same tree except that each U-clause, $C$, attached to a node is replaced by $S(C)$. A U-clause tree is a U-clause resolution tree iff each U-clause is feasible and $S(C)$ is a resolution tree.

4.6 By an equality literal we mean a possibly negated literal using the relation letter "=" and having two lists of terms as its arguments. By a binary equality literal we mean an equality literal where each list contains just one term, e.g. $\neg \psi((x), (f(y)))$ is a binary equality literal (and sometimes we omit the parentheses around the single element lists). A binary equality literal is an equality literal. An equality literal is positive if it has no negation sign, and is negative (or negated) if it has one negation sign.
4.7.1 An **EA sentence** is a sentence consisting of existential quantifiers followed by universal quantifiers followed by a quantifier free matrix (the prefix may be empty), and no existentially quantified variable symbol is also universally quantified.

4.7.2 A sentence, u, in the language $L_\mu$, is in **EAN-normal form** (or is said to be an **EAN-normal sentence**) iff all of the following hold:

1. u is an EA sentence;
2. the matrix of u is a conjunction of disjunctions of negated binary equality literals;
3. for each $q$ a universally quantified variable in u, $q$ appears in at most one conjunct of u (although it may have several occurrences within that single conjunct).

**example 4.7.2.1:** $\exists v \exists w \forall x: (\neg 3(w,x) \lor \neg 2(x,v)) \land \neg 2(w,v)$

is an EAN-normal sentence.

4.7.3 An **EN-normal sentence** is an EAN-normal sentence which has no universally quantified variables.

4.7.4 A **$u$-constraint sentence**, u, is in **EP-normal form** iff u is a sentence consisting of a prefix of existential quantifiers followed by a quantifier free matrix which is a disjunction of conjunctions of positive binary equality literals (the prefix may be empty).

**example:** $\exists r \exists s \exists t: 3(s,t) \lor (3(s,r) \land 3(t,g(s,s)))$

is an EP-normal sentence.

4.8 Let $x_1, x_2 \ldots x_n$ be all of the existentially quantified variables in the EA sentence u, and let $x_1, x_2 \ldots x_n$ be written in
increasing lexical order. Let \( u' \) be the open formula obtained by deleting the existential quantifiers from the prefix of \( u \). Then the gross satisfaction set (in \( L_u \)) of \( u \) is the set of all n-tuples of the form

\[
<g_1|x_1, g_2|x_2 \ldots g_n|x_n>
\]
such that each \( g_i \) is a ground term of \( L_u \), and

\[
\phi_{L_u}'((u')(g_1|x_1, g_2|x_2 \ldots g_n|x_n)) = \text{true}.
\]

The gross satisfaction set of \( u \) is called a satisfaction set of \( u \). The number of components in the members of a satisfaction set is called the order of the satisfaction set. The order of an empty satisfaction set is undefined.

Let \( \Theta \) be a satisfaction set of some RA sentence, \( u \), in \( L_u \), and suppose the order of \( \Theta \) is \( n \), \( n > 0 \), and that \( i \) is an integer between 1 and \( n \), inclusive. We say that \( \Theta \) is reducible on component \( i \) iff

\[
\forall g_1 \forall g_2 \ldots \forall g_n \ [ (g_1|x_1, \ldots g_i|x_i \ldots g_n|x_n) \in \Theta ]
\]

\[ \implies \forall g^* \ [ (g^* \text{ a ground term of } L_u) \implies (g^*|x_1, \ldots g^*|x_i, \ldots g_n|x_n) \in \Theta ] \].

If \( \Theta \) is reducible on component \( i \), then the set, \( \Lambda \):

\[
\Lambda = \{ (g_1|x_1, \ldots g_{i-1}|x_{i-1}, g_{i+1}|x_{i+1}, \ldots g_n|x_n) \}
\]

\[ \exists g: (g_1|x_1, \ldots g_{i-1}|x_{i-1}, g|x_i, g_{i+1}|x_{i+1}, \ldots g_n|x_n) \in \Theta \]
is called the i-component reduction of \( \Theta \).

An i-component reduction of a satisfaction set of \( u \) is a satisfaction set of \( u \). The only satisfaction sets of \( u \) are the gross satisfaction set of \( u \) and those generated from it by successive i-component reductions.
Suppose \( \vartheta \) is an order \( k \) satisfaction set of \( u \), and there is no \( i, 1 \leq i \leq k \), such that \( \vartheta \) is reducible on component \( i \). \( \vartheta \) is then said to be the minimal satisfaction set of \( u \). Starting with the gross satisfaction set for an EA sentence, \( u \), any choice of \( i \)-component reductions will lead to the unique minimal satisfaction set of \( u \).

**Example:** For \( \text{desc}(L_u) = \{ s(-, -) \}, \{ a, b \} \) \( u \) = \( \exists y \exists x \exists z \forall v: s(x, x) \land s(y, z) \land \neg s(y, z) \land ( s(v, y) \lor s(v, z) ) \) has gross satisfaction set \( \{ < a|x, b|y, a|z > , < b|x, b|y, a|z > \} \) and the minimal satisfaction set is \( \{ < b|y, a|z > \} \).

**Example:** For \( L_u \) as above, and \( u = \exists x \exists y \forall z: s(x, y) \lor s(x, z) \lor s(y, z) \) the gross satisfaction set is \( \{ < a|x, a|y > , < a|x, b|y > , < b|x, a|y > , < b|x, b|y > \} \), and this is reducible on component 1 and component 2. The \( i \)-component reduction is \( \{ < a|y > , < b|y > \} \) and this is reducible on component 1, giving its \( i \)-component reduction \( \{ < > \} \). This is the minimal satisfaction set of \( u \).

**Example:** Let \( \text{desc}(L_u) = \{ s(-, -) \}, \{ a, b, \emptyset(-) \} \), and let \( u \) be the same as in the previous example. Then the gross satisfaction set is \( \{ < t|x, t|y > \mid \text{a ground term of } L_u \} \), and this is also the minimal satisfaction set.

Two EA sentences, \( s_1 \), \( s_2 \), are said to be equivalent over \( L_u \) iff the minimal satisfaction sets of \( s_1 \) and \( s_2 \) in \( L_u \) are identical.

4.9 Let \( X \) be a term, atom, literal, sentence, set of terms, set of atoms etc., in a first order language. Then the set of all variable symbols occurring in \( X \) or in elements of \( X \) (or in elements of elements of \( X \), etc.), is denoted by \( \text{variables}(X) \). We use \textbf{there exists} as a function mapping a set of variable symbols to the string of existential quantifiers on those variable symbols such that for \( v_1, v_2, \ldots, v_n \) distinct variable symbols:

\[
\text{there exists}(\{v_1, v_2, \ldots, v_n\}) = \exists v_1 \exists v_2 \ldots \exists v_n .
\]
Similarly, define the function for all:

\[ \text{for all}(v_1,v_2, \ldots, v_n) = \forall v_1 \forall v_2 \ldots \forall v_n. \]

4.10 We use the shorthand notation

\[ \mathcal{E}(t_1,t_2, \ldots, t_n),(s_1,s_2, \ldots, s_n) \]

to stand for

\[ \mathcal{E}(t_1,s_1) \land \mathcal{E}(t_2,s_2) \land \cdots \land \mathcal{E}(t_n,s_n), \]

and

\[ \neg \mathcal{E}(t_1,t_2, \ldots, t_n),(s_1,s_2, \ldots, s_n) \]

to stand for

\[ \neg \mathcal{E}(t_1,s_1) \lor \cdots \lor \neg \mathcal{E}(t_n,s_n). \]

**example:** The sentence \( \exists y \exists z \forall x: \neg \mathcal{E}(w,x),(z,y) \) \( \land \neg \mathcal{E}(w,y) \)

is the shorthand form of the sentence in example 4.7.2.1.

Each pair, \( t_i \) and \( s_i \) in the above literals are called **corresponding terms**, which is short for "corresponding occurrences of terms". For terms \( \alpha \) and \( \beta \), we say that \( \langle \alpha, \beta \rangle \) is a **corresponding pair** of terms in the literal \( Q \) iff there is an occurrence of \( \alpha \), and an occurrence of \( \beta \), both in \( Q \) and the occurrences are corresponding occurrences of terms in \( Q \).

Henceforth we will refer to sentences as being in EAN (EN, EP)-normal form even when they are written with the above shorthand notation, i.e. even when the equality literals in them are not written out explicitly as binary equality literals.

4.11 We say the equality literal \( X \) is **equivalent** to the equality literal \( Y \) iff

\[ \phi_X \text{ (for all)(variables}(X) \cup \text{variables}(Y)) (X \leftrightarrow Y) = \text{true}. \]
Because of the commutivity of "∧" and "∨", and the symmetry of binary equality literals in the standard theory of syntactic equality, we have the following equivalences among equality literals for arbitrary terms \( t_k, s_k, w \) of \( L \):

Let "\( Z \)" be either "\( \bar{s} \)" or "\( \bar{-s} \)".

\[
Z((t_1 \ldots t_j \ldots t_n),(s_1 \ldots s_j \ldots s_n))
\]

is equivalent to

\[
Z((t_1 \ldots s_j \ldots t_n),(s_1 \ldots t_j \ldots s_n)),
\]

and

\[
Z((t_1 \ldots t_i \ldots t_j \ldots t_n),(s_1 \ldots s_i \ldots s_j \ldots s_n))
\]

is equivalent to

\[
Z((t_1 \ldots t_j \ldots t_i \ldots t_n),(s_1 \ldots s_j \ldots s_i \ldots s_n)),
\]

and

\[
Z((t_1 \ldots t_k,w,t_{k+1}, \ldots t_n),(s_1 \ldots s_k,w,s_{k+1}, \ldots s_n))
\]

is equivalent to

\[
Z((t_1 \ldots t_k,t_{k+1}, \ldots t_n),(s_1 \ldots s_k,s_{k+1}, \ldots s_n)).
\]

4.12 For \( X \) an equality literal, let

\[
C_p(X) = \{ <t_i,s_i> | <t_i,s_i> \text{ a corresponding pair in } X \}.
\]

Then \( X \) is said to \textbf{subsume} the equality literal \( Y \) iff \( C_p(X) \) is contained in \( C_p(Y) \) and \( X \) and \( Y \) are both positive or both negative.

4.13 Suppose \( u \) is a \( U \)-constraint sentence in EA form in \( L_u \), and \( \sigma \) is a substitution:

\[
\sigma = \{ t_1/v_1, t_2/v_2, \ldots, t_k/v_k \},
\]

where \( t_i \)'s are terms of \( L_u \), and \( v_i \)'s are (distinct) variable symbols.
of $L_u$. We assume $\sigma$ is in an explicit simple form such that no $vi$ appears as part of any $t_i$. We will be concerned only with applying substitutions, $\sigma$, to $u$, where no $vi$, $1 \leq i \leq k$, is a universally quantified variable symbol in $u$. In such a case $\sigma$ is said to be a pure existential substitution for $u$. All of the known $U$-clause refinements require only the application of substitutions to a $U$-clause such that the substitution is a pure existential substitution for the $U$-constraint sentence of the clause. The following procedure shows how to apply such substitutions to $U$-constraint sentences.

**Procedure 4 — 1**

**Application of a Substitution to a $U$-Constraint Sentence**

For $U$-constraint sentence $u$, where $u$ is an EA sentence in $L_u$, and substitution $\sigma$, where $\sigma$ is a pure existential substitution for $u$ as given above, to compute $(u)\sigma$:

1. For each universally quantified variable, $x$, in $u$, which also appears in $\sigma$, uniformly re-name $x$ in $u$ to a new variable symbol not in $\sigma$ and not already in $u$; call the result $u'$.

2. Replace each occurrence of $vi$, $1 = 1, 2 \ldots k$, in $u'$ by the term $t_i$, including occurrences in quantifiers.

3. For each quantifier in $u'$ of the form $\exists c$, for $c$ a constant symbol of $L_u$, delete the quantifier $\exists c$ from the prefix of $u'$.

4. For each quantifier in $u'$ of the form $\exists f(q_1 \ldots q_m)$, where $f$ is a function symbol of $L_u$, and the $q_i$'s are terms of $L_u$, replace $\exists f(q_1 \ldots q_m)$ in the prefix of $u'$ by $\exists f(q_1, q_2 \ldots q_m)(\exists \text{variables}(f(q_1, q_2 \ldots q_m)))$. If the $q_i$'s are all ground terms then $\exists f(q_1 \ldots q_m)$ is simply deleted from the prefix of $u'$.

5. For each variable symbol, $\exists c$, appearing in the existential quantifier list of $u'$, if "$\exists c$" appears more than once, then delete all occurrences of "$\exists c$" after the first. The resulting string is $(u)\sigma$.

end of procedure.
example:  For \( u = \exists x \forall y \forall z: \exists(x, f(y)) \lor \exists(x, g(z)) \lor \neg \exists(g(x), z) \) \\
and \( \sigma = \{ g(v)/x, z/y \} \)
we have 
\((u)\sigma = \exists v \forall z: \exists(g(v), f(z)) \lor \exists(g(v), g(z)) \lor \neg \exists(g(v), z)\)

4.14 Our ground extension function, \( G \), is defined for a \( U \)-clause, \( C \), when \( \kappa(C) \) is an \( EA \) sentence, as follows.

\[
G(C) = \{ (S(C))\sigma \mid (\sigma \text{ a substitution of } L_U) \\
\land (\sigma \text{ pure existential substitution for } \kappa(C)) \\
\land (\sigma \text{ a minimal grounding substitution for } S(C)) \\
\land (\text{feasible } (\kappa(C))\sigma) \}
\]

We also define \( G^* \) by

\[
G^*(C) = \{ (C)\sigma' \mid (\sigma \text{ a substitution of } L_U) \\
\land (\sigma \text{ pure existential substitution for } \kappa(C)) \\
\land (\sigma \text{ a minimal grounding substitution for } S(C)) \\
\land (\text{feasible } (\kappa(C))\sigma) \},
\]

where we define \((C)\sigma' = < (S(C))\sigma', (\kappa(C))\sigma>\).

Thus the ground extension set, \( G(C) \), of a \( U \)-clause \( C \) is the set of ground instances of \( S(C) \), over the appropriate Herbrand universe, for which the \( U \)-constraint sentence of \( C \) is true after the grounding substitution is applied to it.

\( G \) and \( G^* \) are extended to \( U \)-clause trees analogously to the extension of \( G \) for ordinary resolution trees (see definition 2.10).

4.15 We define a function, \( \chi \), from sets of BUSL literals to \( U \)-constraint sentences as follows. Let

\[ \mathcal{H} = \{ l_1, l_2, \ldots, l_k \} \]
be a set of HUSL literals such that in some (arbitrarily chosen but
fixed) lexical ordering of strings, \( li < lj \iff i < j \). Then define \( M \) by

\[
M(H) = \text{there exists}(\text{variables}(H)) \ 11 \land 12 \land \ldots \land 1k.
\]

**example:** Let \( H \) be the HUSL of clause 6 in Figure 3a. Then \( M(H) \) is

\[
\exists x \exists y \exists z: \neg \bar{e}((x,y),(x,z)) \land \neg \bar{e}((x,z),(z,x)) \land \neg \bar{e}((y,z),(x,z))
\]

Let \( u \) be an EN-normal sentence where the individual conjuncts are
written as equality literals (as opposed to binary equality literals)
so that each conjunct is a single literal. Then we define \( M'(u) \) to be
the set of HUSL literals which are exactly all of the conjuncts in \( u \).
Thus we have, for \( C \) any HUSL-clause, \( \mathcal{U}(C) = M'(M(\mathcal{U}(C))) \).

**example:** For \( u = \exists x \exists y \exists z: \neg \bar{e}((x,y),(x,z)) \land \neg \bar{e}((y,z),(x,z)) \),

\[
M'(u) = \{ \neg \bar{e}((x,y),(x,z)), \neg \bar{e}((y,z),(x,z)) \}
\]

In direct analogy to what was previously done for ordinary
refinements, we define a \( U \)-refinement to be a subset of the set of all
\( U \)-clause resolution trees. The same characterizations apply to
\( U \)-refinements as to ordinary refinements in terms of such notions as
locally negatable, locally defined and ground faithfulness.
5.0 Some Specific U-Clause Refinements.

In general, for each HUSL-clause refinement, we can define a corresponding U-clause refinement which is at least as restrictive as the HUSL-clause refinement. This is one way to obtain the specification of a U-clause refinement, and will be illustrated by our first example.

5.1 U-clauses for Eliminating Latent Initial Tautologies.

We define $\u^e_{\text{LR}_E^{\text{TAUT}}}$ in direct analogy to $\u^e_{\text{LR}_E^{\text{TAUT}}}$ in section 3.1, as follows:

5.1.1 Given $S$ as a set of lock clauses, let $\u^e_{\text{S}\text{TAUT}}$ be given by

$$\u^e_{\text{S}\text{TAUT}} = \{ (\mathcal{S}(C), \mathcal{M}(\mathcal{U}(C))) \mid C \in \u^e_{\text{S}\text{TAUT}} \land \text{(feasible } \mathcal{M}(\mathcal{U}(C)) \text{)} \}.$$  

5.1.2 U-clause $C_3$ is a $\u^e_{\text{LR}_E^{\text{TAUT}}}$-resolvent of U-clauses $C_1$ and $C_2$ iff there exists a $\sigma$:

1. $\mathcal{S}(C_3)$ is an LR-resolvent of $\mathcal{S}(C_1)$ and $\mathcal{S}(C_2)$, with resolution unifier $\sigma$;

2. $\mathcal{U}(C_3) = \mathcal{M}(\mathcal{M}(\mathcal{U}(C_1))\sigma) \cup \mathcal{M}(\mathcal{U}(C_2))\sigma$;

3. (feasible $C_3$).

5.1.3 The refinement $\u^e_{\text{LR}_E^{\text{TAUT}}}$ is defined by:

for initial set $S$,

1. each U-clause in $\u^e_{\text{S}\text{TAUT}}$ is a deduction in $\u^e_{\text{LR}_E^{\text{TAUT}}}$;

2. for $D_1, D_2$ deductions in $\u^e_{\text{LR}_E^{\text{TAUT}}}$, and $C$ a U-clause which is a $\u^e_{\text{LR}_E^{\text{TAUT}}}$-resolvent of $\text{root}(D_1)$ and $\text{root}(D_2)$, the deduction $\text{graft}(D_1, D_2, C)$ is in $\u^e_{\text{LR}_E^{\text{TAUT}}}$. 
4. \( \neg Q(v, w) \wedge Q(w, v) \); \( \mathcal{U} = [ \exists v \exists w: \neg \mathcal{E}(v, w), (w, v) ] \)

5. \( \neg Q(x, y) \wedge \neg Q(y, z) \wedge Q(x, z) \); 
   \( \mathcal{U} = [ \exists x \exists y \exists z: \neg \mathcal{E}(x, y), (x, z), \neg \mathcal{E}(y, z), (x, z) ] \)

6. \( \neg Q(x, y) \wedge \neg Q(y, z) \wedge Q(z, x) \); 
   \( \mathcal{U} = [ \exists x \exists y \exists z: \neg \mathcal{E}(x, y), (z, x), \neg \mathcal{E}(y, z), (x, z) ] \)

**FIGURE 6**

U-Clause Form of Latent Initial Tautology Elimination for Initial Set from Figure 3.
5.1.4 In essence the $\mathcal{U}_{\text{TAUT}}^S$ refinement is identical to $\mathcal{U}_{\text{TAUT}}^S$ but with a stronger notion of feasibility. This is apparent from Figure 6. Figure 6 is the U-clause tree corresponding to the topmost three nodes of Figure 3a. This is not an $\mathcal{U}_{\text{TAUT}}^S$ deduction however, since clause 6 is not feasible, i.e.,

$$\phi_u(\mathcal{U}\text{\{clause 6\}}) = \text{false}.$$  

We see that the U-clause notion is considerably stronger than the HUSL-clause notion (cf. Figure 3a, where the HUSL-clause deduction must continue two levels more before infeasibility occurs). Notice that the deduction tree consisting just of clause 4 in Figure 6, and the deduction tree consisting just of clause 5 of Figure 6, are both in $\mathcal{U}_{\text{TAUT}}^S$.

Now we point out a rather curious situation. Suppose we replace $S$ by the set $S'$:

$$S' = 3.\ Q(u,u)^2;$$

$$4.\ \neg Q(v,w)^4, Q(w,v)^8;$$

$$5.\ \neg Q(x,y)^7, \neg Q(y,z)^9, Q(x,z)^6;$$

i.e. $S'$ is the set consisting of clauses 3, 4 and 5 of Figure 1. Now we form the corresponding U-clauses for $S'$ for $\mathcal{U}_{\text{TAUT}}^S$:

3*.$$ Q(u,u)^2; U = (\ )$$

4*.$$ \neg Q(v,w)^4, Q(w,v)^8; U = (\ \exists v \exists w: \neg \exists ((v,w),(w,v)) \ )$$

5*.$$ \neg Q(x,y)^7, \neg Q(y,z)^9, Q(x,z)^6; U = (\ \exists x \exists y \exists z: \neg \exists ((x,y),(x,z)), \neg \exists ((y,z),(x,z)) \ )$$

and we see that U-clauses 4* and 5* are not feasible, since the relevant language, L, is given by

$$\text{desc}(L) = \langle (Q(-,-)), (a) \rangle$$

which gives

$$\phi_u(\mathcal{U}\text{\{clause 4*\}}) = \phi_u(\mathcal{U}\text{\{clause 5*\}}) = \text{false}.$$
Thus the entire refinement \( \mathcal{U}_{\mathcal{L}_5^{\tau}} \) consists of the singleton set of deductions:

\[
\mathcal{U}_{\mathcal{L}_5^{\tau}} = \{ Q(u,u)^3; \ U = {} \}.
\]

The reader should be able to construct for himself the \( U \)-clause refinements for the elimination of latent derived tautologies in UNR and the factoring choices in LR.

5.2 \( U \)-Clauses for Elimination of Latent Initial Subsumptions.

Analogous to latent tautologies in initial clauses is the notion of latent subsumption among initial clauses. The \( U \)-constraints involved in such a refinement will have universally quantified variables. This is our only example of universal quantification in constraints in this report.

5.2.1 We view ordinary clauses as sets of literals (as opposed to multi-sets). Clause \( C_f \) is a simple factor of clause \( C \) iff there exists \( k_1, k_2 \):

1. \( k_1 \in C \land k_2 \in C \land k_1 \neq k_2 \);
2. (unifiable \((k_1, k_2)\));
3. \( C_f = (C)_{\text{mgu}}(k_1, k_2) \).

[If \( k_1 \) and \( k_2 \) are lock literals, the lock numbers are invisible when deciding if they are unifiable, and in condition 3, \( C_f \) retains the image only of a lowest lock numbered literal among \( k_1 \) and \( k_2 \).]

5.2.2 Clause \( C' \) is a factor of clause \( C \) iff there exists a sequence of clauses, \( C_1, C_2, \ldots, C_n \), such that \( C_1 = C \), \( C_n = C' \), and \( C_{i+1} \) is a simple factor of \( C_i \), for \( i = 1, 2, \ldots, n-1 \).
A set of clauses, $S$, is said to be fully factored iff for every clause $C$ in $S$, every factor of $C$ is also in $S$.

5.2.3 Let "<" be a total ordering on clauses such that $C < D$ implies $D$ has at least as many literals as $C$, and if $C$ and $D$ have the same number of literals some arbitrarily chosen criteria determines if $C < D$ or $D < C$. For a fully factored set of clauses $S$, containing $N$ clauses, we denote by $\mathcal{S}$ the set of clauses whose elements are the same as in $S$ but are labelled by "C1", "C2", ..., CN, such that if $i < j$ then $Ci < Cj$.

If $k$ is a lock literal, then $\text{core}(k)$ is defined to be the literal $k$ without its lock number. If $k$ is an ordinary literal, $\text{core}(k) = k$.

For the elimination of latent subsumptions among initial clauses, we will let $S$ be the initial set, which we assume is already fully factored. As an expedient to simplify the definitions, if $S$ is a lock set of clauses, then we assume that no two literal occurrences in $S$ have the same lock numbers. In addition, optionally, $S$ may be simplified by eliminating explicit tautologies, and any clauses explicitly subsumed by other clauses (where lock numbers are ignored in deciding if a clause is a tautology, and in deciding subsumption relationships). It is permitted that the resulting simplified set be no longer fully factored. Then the "<" relation is defined for $S$, and labels $Ck$ are assigned to these clauses to form the set of labelled clauses, $\mathcal{S}$. We give below the definitions which will allow $\mathcal{S}$ to be transformed into a set of U-clauses, which we will designate as $\mathcal{S}_0$. $\mathcal{S}_0$ will consist of the level 0 deductions in our refinement. As
usual, we assume that no variable symbol appears in more than one clause in $\text{Sub}$. 

5.2.4 We say clause $C$ P-subsumes clause $D$ iff there exists a substitution, $\sigma$, such that $(C)\sigma$ is a proper subset of $D$. We say $C$

letently P-subsumes $D$ iff $C$ P-subsumes some member of $G(D)$. We define P-SUB as a relation on two clauses and a mapping, as follows.

$P$-SUB($C,D,m$) iff

1. $m$ is a many-one mapping, total on literals of $C$, and onto a proper subset of $D$;
2. there exists a substitution, $\sigma$, called the subsumption substitution (for $m$), such that for each $k$ an element of $C$, $\text{core}((m(x))\sigma) = \text{core}((k)\sigma)$;
3. there exists at least one literal, $q$, such that $q \in D$ and $\text{core}((q)\sigma)$ is not equal to $\text{core}(x)$ for any $x$ in $K\sigma$, where $K$ is the proper subset of $D$ mapped onto by $m$, and $\sigma$ is the subsumption substitution for $m$.

Example 5.2.4.1: For clauses $C$ and $D$ and the mapping $m$ given by the arrows shown,

$C = P(x)^{x}, Q(x,y)^{y}, Q(y,x)^{x}$;

$D = P(b)^{b}, P(a)^{a}, Q(b,u)^{u}, Q(v,\alpha)^{\alpha}, R(z)^{z}$;

the relation $P$-SUB($C,D,m$) is true, and the subsumption substitution is \{(x,a,y,a/v)\}. Also we have the mapping $m'$ given by

$P(x)^{x}, Q(x,y)^{y}, Q(y,x)^{x}$;

$P(b)^{b}, P(a)^{a}, Q(b,u)^{u}, Q(v,\alpha)^{\alpha}, R(z)^{z}$;

and $P$-SUB($C,D,m'$) is true with the subsumption substitution \{(b/x,b/y,b/u)\}. 

5.2.5 We define the $\text{P-subsumption conjunct of } J, K \text{ under } m$, denoted by $\text{P-SUB-C}(J,K,m)$, as follows. If $\text{P-SUB}(J,K,m)$ is true, then
\[
\text{P-SUB-C}(J,K,m) = \forall \text{all(variables}(J)) \bigvee \{ \text{distinctness literal}(j,m(j)) \mid j \in J \}.
\]

**Example 5.2.5.1:** For the clauses $C$ and $D$, and mapping $m'$ of example 5.2.4.1, we have
\[
\text{P-SUB-C}(C,D,m') = \forall x\forall y: \sim \mathcal{E}(x),(b) \bigvee \sim \mathcal{E}(x,y),(b,u) \bigvee \sim \mathcal{E}(y,x),(b,u).
\]

5.2.6 For clauses $J, K$, define
\[
\text{P-SUB-C}(J,K,-) = \bigwedge \{ \text{P-SUB-C}(J,K,m) \mid \text{P-SUB}(J,K,m) \}.
\]

**Example 5.2.6.1:** For $C$ and $D$ as in example 5.2.4.1,
\[
\text{P-SUB-C}(C,D,-) = \forall x\forall y: \sim \mathcal{E}(x),(a) \bigvee \sim \mathcal{E}(x,y),(v,a) \bigvee \sim \mathcal{E}(y,x),(v,a)
\bigwedge \forall x\forall y: \sim \mathcal{E}(x),(b) \bigvee \sim \mathcal{E}(x,y),(b,u) \bigvee \sim \mathcal{E}(y,x),(b,u).
\]

5.2.7 We define $\text{E-SUB}$ as a relation on two clauses and a mapping, as follows.

$\text{E-SUB}(C,D,m)$ iff
1. $m$ is a many-one mapping, total on literals of $C$, and onto $D$;
2. there exists a substitution, $\sigma$, called the **subsumption substitution** (for $m$), such that for each $k$ an element of $C$,
   \[
   \text{core}(\{(m(k))\sigma\}) = \text{core}(\{(k)\sigma\}).
   \]

We define the $\text{E-subsumption conjunct of } J, K \text{ under } m$, denoted by $\text{E-SUB-C}(J,K,m)$, as follows. If $\text{E-SUB}(J,K,m)$ is true, then
\[
\text{E-SUB-C}(J,K,m) = \forall \text{all(variables}(J)) \bigvee \{ \text{distinctness literal}(j,m(j)) \mid j \in J \}.
\]
5.2.6 For clauses \( J, K \), define
\[
\text{E-SUB-C}(J, K, -) = \bigwedge \{ \text{E-SUB-C}(J, K, m) \mid \text{E-SUB}(J, K, m) \}.
\]

5.2.9 For each \( i \), \( i = 1, 2, \ldots, N \), define the latent initial subsumption transform, \( \text{Ci}^* \), of an ordinary labelled clause \( \text{Ci} \) in \( \mathcal{S} \) by
\[
\text{Ci}^* = \langle \text{Ci} \rangle \land \text{there exists(variables(Ci))}
\land \bigwedge \{ \text{P-SUB-C}(C_j, \text{Ci}, -) \mid i \neq j \land C_j \in \mathcal{S} \}
\land \bigwedge \{ \text{E-SUB-C}(C_j, \text{Ci}, -) \mid j < i \land C_j \in \mathcal{S} \} \rangle.
\]

5.2.10 Finally, for \( \mathcal{S} \) and \( \mathcal{S}^\text{sub} \) as identified in section 5.2.3, we define
\[
\mathcal{S}^\text{sub} = \{ \text{Ci}^* \mid \text{feasible Ci}^* \}
\land \text{(Ci* a latent initial subsumption transform of Ci in \( \mathcal{S} \))}
\land \text{Ci} \in \mathcal{S}, \text{for } i = 1, 2, \ldots, N \}.
\]

Figure 7 illustrates the initial set transformation for the 5 initial clauses of Figure 1. In Figure 7a these initial clauses are listed in a size-ordered way, and furthermore are already (for our purposes) closed under factoring, since the only clause that can be factored is clause 5, and the factor is a tautology. The clauses 1* through 5* in Figure 7b are the result of the transformation of clauses 1 through 5 in Figure 7a. Clauses 1*, 2* and 3* have empty constraints because for each of them:

a. no earlier clause exists (strictly speaking we should say "no earlier clause exists for their pre-images, clauses 1, 2 and 3") which can have a subsumption map properly into or onto 1, 2 or 3;
b. no later clause has a subsumption map properly into 1, 2 or 3.
No U-clause in Figure 7b has any U-constraint conjuncts arising from an E-SUB relationship. The notations for clauses 4* and 5* on the right of Figure 7b give the origin of the U-constraint conjunct on that line. Thus "m: 1 --> 8" means that the corresponding conjunct is a result of the literal with index 1 being mapped to the literal of index 8.

A careful examination of the constraint of U-clause 5* shows that it is infeasible. Thus U-clause 5* is not a member of $\Sigma_{\text{SUB}_0}$ (which does contain 1* through 4*), and thus 5* is not a deduction in $\mu_{\text{LFS}_0}$. The infeasibility of clause 5* is due in part to the limited Herbrand universe for clauses 1 through 5. Note that no single clause among 1 through 4 subsumes clause 5, but rather that, for this limited Herbrand universe, different groundings of clause 5 are subsumed by different clauses among 1 through 4. We call a situation such as this distributed subsumption.

In Figure 7b we have written the U-constraint sentence for clause 5* in an equivalent, but syntactically different way than it would be generated by the prescription given in by the above definitions. For example, the part corresponding to the map "m: 4 --> 7, 8 --> 8" would initially be obtained as

$$\forall v \forall w: \neg\exists((x, y), (v, w)) \lor \neg\exists((x, z), (w, v)),$$

which can be seen, by definition 4.10, to be equivalent to that given in Figure 7b. The constraint of 5* can be put into FAN-normal form by re-naming universally quantified variables with distinct new names so as to allow the quantifiers to be moved up into the prefix. In this sense the universally quantified variables act as they ordinarily
would in first order logic as dummy symbols. The existentially quantified variables however, are not dummies which can be arbitrarily re-named internally to the U-constraint sentence alone, but rather can only be correctly re-named if the same re-naming is done on the variables appearing in the standard literals to which the U-constraint sentence belongs.

5.2.11 We now describe the inference rule for the $U_{LR_{SUB}}$-refinement. As usual, we assume that clauses have had their variables standardized apart, and this includes the universally quantified variables in U-clauses. We use LR with implicit factoring. Let $C_1$ and $C_2$ be U-clauses each with an EA sentence as its U-constraint, and lock literals as standard literals:

$$C_1 = < S(C_1), \exists X \forall Y Z >$$
$$C_2 = < S(C_2), \exists X' \forall Y' Z' >$$

where $\exists X$ is the string of existential quantifiers of $U(C_1)$, $\forall Y$ the string of universal quantifiers, and $Z$ its matrix. Similarly for $\exists X'$, etc. in $U(C_2)$. Then we say U-clause $C_3$ is an LR direct $U$-resolvent of $C_1$ and $C_2$ iff:

there exists a $\sigma$:

1. $S(C_3)$ is an LR resolvent of $S(C_1)$ and $S(C_2)$, with resolution unifier $\sigma$;
2. $U(C_3) = (\exists X \exists X' \forall Y \forall Y' Z \wedge \exists X' Z')\sigma$;
3. $C_3$ is feasible.

Other types of direct $U$-resolvent are analogously defined.
5.2.12 The refinement $\gamma_{LR}^{E}_{\text{Sub}}$ is defined by:

1. each clause in $S_{\text{Sub}}^*$ is a one node deduction in $\gamma_{LR}^{E}_{\text{Sub}}$;
2. for $D1$ and $D2$ deductions in $\gamma_{LR}^{E}_{\text{Sub}}$, if $C$ is an LR direct
   U-resolvent of root($D1$) and root($D2$), then the deduction
   graft($D1,D2,C$) is in $\gamma_{LR}^{E}_{\text{Sub}}$.

In Figure 8 we show the entire refinement $\gamma_{LR}^{E}_{\text{Sub}}$, where $S_{\text{Sub}}^*$ is
the set of clauses $1^*$ through $4^*$ of Figure 7b. There are exactly 6
deduction trees in $\gamma_{LR}^{E}_{\text{Sub}}$, which are:

1. the 4 feasible clauses in $S_{\text{Sub}}^*$, $1^*$ through $4^*$;
2. one level 1 deduction, of clause $6^*$;
3. one level 2 deduction, which is a refutation.

Notice that trying to resolve $4^*$ with $3^*$ and $6^*$ with $4^*$ gives an
infeasible result in both cases.

This completes the detailed examples of HUSL and U-clause
refinements. Later some other uses of HUSL and U-constraint clauses
will be mentioned in the general discussion.
1. \( Q(a,b)' \);  
2. \( \neg Q(b,a)^2 \);  
3. \( Q(u,u)^2 \);  
4. \( \neg Q(v,w)^6, Q(w,v)^8 \);  
5. \( \neg Q(x,y)^7, Q(y,z)^7, Q(x,z)^6 \);  

**FIGURE 7a**  
Initial Set \( S \).

\[ 1^* \quad Q(a,b)' \quad U = \{ \} \]  
\[ 2^* \quad \neg Q(b,a)^2 \quad U = \{ \} \]  
\[ 3^* \quad Q(u,u)^2 \quad U = \{ \} \]  
\[ 4^* \quad \neg Q(v,w)^6, Q(w,v)^8 \]  
\[ U = \{ \exists v \exists w : \neg \epsilon((w,v),(a,b)) \land \neg \epsilon((v,w),(b,a)) \land \forall u : \neg \epsilon((w,v),(u,u)) \} \]  
\[ m : 1 \rightarrow 8 \]  
\[ m : 2 \rightarrow 4 \]  
\[ m : 3 \rightarrow 8 \]  
\[ 5^* \quad \neg Q(x,y)^7, \neg Q(y,z)^7, Q(x,z)^6 \]  
\[ U = \{ \exists x \exists y \exists z : \neg \epsilon((x,z),(a,b)) \land \neg \epsilon((x,y),(b,a)) \land \neg \epsilon((y,z),(b,a)) \land \forall u : \neg \epsilon((x,z),(u,u)) \land \forall v \forall w : \neg \epsilon((x,y,z),(v,w,w,v)) \land \forall v \forall w : \neg \epsilon((y,z,x),(v,w,w,v)) \} \]  
\[ m : 1 \rightarrow 6 \]  
\[ m : 2 \rightarrow 7 \]  
\[ m : 2 \rightarrow 9 \]  
\[ m : 3 \rightarrow 6 \]  
\[ m : 4 \rightarrow 7, 8 \rightarrow 6 \]  
\[ m : 4 \rightarrow 9, 8 \rightarrow 6 \]  

**FIGURE 7b**  
Transforms of Clauses in Figure 7a  
(clauses 1* through 4* constitute \( S^a \)).
level 0

1*. \( Q(a, b) \); \( U = \{ \} \)

2*. \( \neg Q(b, a) \); \( U = \{ \} \)

3*. \( Q(u, u) \); \( U = \{ \} \)

4*. \( \neg Q(v, w) \), \( Q(w, v) \); \( U = \{ \forall v \exists w: \neg \theta((v, w), (e, b)) \land \theta((v, w), (b, a)) \land \forall x: \neg \theta((v, w), (x, x)) \} \)

level 1

5*. \( Q(b, a) \); \( U = \{ \forall b, a: \neg \theta((b, a), (a, b)) \land \theta((a, b), (b, a)) \land \forall x: \neg \theta((b, a), (x, x)) \} \)

level 2

6*. \( \emptyset \); \( U = \{ \text{same as clause 6*} \} \)

FIGURE 8

The Deductions in the Refinement \( \nu \mathcal{L}^{\mathcal{S}} \mathcal{L}^{\mathcal{S}} \nu \mathcal{S} \) (For \( S \) in Figure 7a).
6.0 Evaluating and Simplifying Constraints

In this chapter we discuss the issues relating to the overhead of using U-constraints: evaluation time and storage. The evaluation time for EP-normal constraints (and, when the Herbrand universe is infinite, the EN-normal constraints also) is linear in the size of the constraint. The EAN-normal constraints can be evaluated by a partial procedure in linear time. Evaluation procedures for other forms of sentences have not been investigated in detail. For all types of constraint sentences there is the potential for extremely large storage requirements, depending upon the exact combination of U-clause refinements being used and the details of the particular set of clauses.

The feasibility testing of HUSL constraint sets of literals can be done in time linear in the length of the HUSL. Thus, assuming the HUSL constraint is not extremely large, there is no difficulty in feasibility testing of HUSL-clauses. However, as seen in section 5.1.4, the HUSL notion does not permit the statement of refinements as strong as that permitted by U-constraint sentences. We assume, in this section, that it is worthwhile to use the U-constraint sentences and consider the problem of feasibility testing of U-constraints.

Thus far in this report, the most general form of a U-constraint sentence that has been employed is the EAN-normal sentence. An example of a refinement condition which generates an EP-normal constraint is given in section 7.1.4. In what follows we treat several special situations where something is known about the task of
feasibility testing. The two main cases to consider are those without 
function symbols of degree greater than zero, which yields a finite 
Herbrand universe, and the infinite Herbrand universe case.

6.1 \( L_u \) has a finite Herbrand universe. Let \( u \) be an arbitrary 
\( U \)-constraint sentence in prenex form. Then obviously there are only a 
finite number of substitution instances of the matrix of \( u \) which need 
be checked to determine the truth of \( u \). A blind backtrack method of 
substitution of individual constants (of \( L_u \)) for the quantified 
variables would probably yield an acceptably efficient evaluation 
mechanism when \( L_u \) has a relatively small number of constants. When 
there are a large number of constant symbols in \( L_u \), then efficiency of 
the evaluation procedure may be critical. In the next section some 
simplifications of the structure of sentences will be given. These 
simplifications may be applicable (it depends upon the structure of 
the sentence) before substitution instances of the matrix are formed, 
thus reducing, in some cases to zero, the number of instances which 
need to be checked. We leave the details of adapting these procedures 
to the special case of a finite Herbrand universe to the reader. 
Procedures 6-1, 6-2 and 6-3, below, are also relevant to this case of 
a finite Herbrand universe.

6.2 There are function symbols in \( L_u \) of degree greater than zero. We 
will first give explicit algorithms for determining the truth value of 
\( EN \)-normal and \( EP \)-normal sentences of \( L_u \) in the standard theory of 
syntactic equality for \( L_u \), and then will give a partial procedure for 
the more general \( EAN \)-normal sentences. The following definitions are 
used in these procedures.
6.2.1 An equality literal, q, is said to be in top level form iff no pair of corresponding terms in q start with the same function letter (although a corresponding pair of terms in which each term is the same constant symbol or variable symbol is allowed).

Suppose q is not in top level form, and tj and sj are corresponding term occurrences in q such that tj = f(α) and sj = f(β), for α and β lists of terms of L_0. By replacing that single occurrence of tj by α, and that single occurrence of sj by β, q is transformed into a literal equivalent to itself. By doing such replacements, any equality literal q can be transformed into top level form. A sentence in L_0 is in top level form iff every literal in it is in top level form. Converting a sentence in L_0 into top level form does not alter its truth value in the standard theory of syntactic equality. If the original sentence was an E_0 sentence, then its satisfaction sets are likewise unaltered, and the top level form is equivalent to the original sentence.

example:
\[ \exists x \exists y \exists z \forall u \forall v: \exists((g(x,y),z),(g(u,y),v)) \land \exists((f(x,n(y)), (f(u,v))) \]
has top level form
\[ \exists x \exists y \exists z \forall u \forall v: \exists((x,y,z),(u,y,v)) \land \exists((x,n(y)), (u,v)). \]

6.2.2 For q an equality literal, q is said to be component reduced iff no pair of corresponding terms in q are occurrences of the same term of L_0. A sentence L_0 is component reduced iff every literal in it is component reduced. Any equality literal (sentence) can be transformed to a component reduced literal (sentence) by simply deleting corresponding identical terms. If an equality literal is converted to component reduced form, the result is equivalent to the
original literal. If an E. A sentence in $L_u$ is converted to component reduced form, the result is equivalent to the original sentence.

**Example:** \[ \exists x \exists y \exists z \forall u \forall v : \exists ((x,y,z),(u,v,v)) \land \exists ((x,\cdot(y)),(u,v)) \]

is not component reduced, but has the equivalent component reduced form
\[ \exists x \exists y \exists z \forall u \forall v : \exists ((x,z),(u,v)) \land \exists ((x,\cdot(y)),(u,v)). \]

**Example:** \[ \exists x \exists y \exists z : \neg \exists ((x,y),(x,y)) \land \neg \exists ((x),(z)) \]

has the component reduced form
\[ \exists x \exists y \exists z : \neg \exists ((,(),()) \land \neg \exists ((x),(z)). \]

6.2.3 For \( q \) an equality literal, \( q \) is said to be **list unifiable** iff there exists a substitution, \( \sigma \), such that for every pair of corresponding terms, \( < t_1, s_1 > \), in \( q \), \( (t_1)\sigma = (s_1)\sigma \). The unifier \( \sigma \) is computed in the usual way, that is without any special consideration as to whether a particular variable symbol is existentially or universally quantified in the \( U \)-constraint from which the equality literal came.

6.2.4 An EAN-normal sentence is said to be in **minimal form** iff it has a matrix which is a conjunction in which each conjunct is a single negative equality literal. An EP-normal sentence is said to be in **minimal form** iff it has a matrix which is a disjunction in which each disjunct is a single positive equality literal. By use of the notational convention of section 4.10, every EAN-normal or EP-normal sentence can be transformed into an equivalent minimal form.

**Example:**
\[ \exists x \exists y \exists z \forall u \forall v : (\exists ((x,z),(u,v)) \lor \exists ((x,\cdot(y)),(u,v))) \land \exists (x,y) \]

has minimal form
\[ \exists x \exists y \exists z \forall u \forall v : \exists ((x,z,x,\cdot(y)),(u,v,u,v)) \land \exists (x,y). \]
Procedure 6 --- 1

Procedure for Reducing and Evaluating EN-Normal Sentences

Let $u$ be an EN-normal sentence in the language $L_u$, where $L_u$ contains at least one function symbol of degree greater than zero. To compute the truth value of $u$ in the standard theory of syntactic equality for $L_u$:

1. Let $u_1$ be the minimal form of $u$.
2. Let $u_2$ be the top level form of $u_1$.
3. Let $u_3$ be the component reduced form of $u_2$.
4. Let $u_4$ be the result of eliminating all of the conjuncts from $u_3$ which are literals which are not list unifiable (also delete "\exists v" from the quantifier list of $u_4$ for each variable $v$ which has no occurrences at all in the matrix of $u_4$).
5. If the literal \( \neg \emptyset (((),())) \) is in $u_4$, then $u$ is false; otherwise $u$ is true.

end of procedure.

The above procedure obviously terminates, and its correctness is easily proven. If $u$ is in fact true, then the $u_4$ computed in step 4 of the procedure can be taken as a simplified form of $u$ which is equivalent to $u$. Note that in step 4 it is only necessary to determine if a literal is list unifiable, and the unifier, if it exists, is not actually applied to the literal. Thus this test for list unifiability can be performed in time linear in the size of the
literal [Paterson and Wegman, 1978]. Thus Procedure 6-1 requires only linear time to evaluate and simplify an EN-normal sentence. Additional simplifications of $u_4$ can also be made, e.g. if $c_i$, $c_j$ are conjuncts in $u_4$, and $c_i$ subsumes $c_j$, then $c_j$ may be deleted. Searching for such subsumption possibilities seems to require more than linear time, but not more than the square of the length of the sentence. An additional simplification that can be done is the deletion of multiple occurrences of the same corresponding pair of terms in a single literal, for example,

$$\exists x \exists y \exists z \exists u: \neg((x,y,z),(z,g(u),x))$$

can be replaced by the equivalent sentence

$$\exists x \exists y \exists z \exists u: \neg((x,y),(z,g(u))).$$

When $L_u$ has only constant symbols in its Herbrand universe, then Procedure 6-1 can be modified by replacing step 5 by 5':

5'. If the literal $\neg \emptyset((),()$ is in $u_4$, then $u$ is false. Otherwise substitute constant symbols for variables in the matrix of $u_4$ in all possible ways; if any substitution makes the matrix a true conjunction, then $u_4$ is true, otherwise $u_4$ is false.

Next we give the procedure for EP-normal sentences which is quite similar to that for EN-normal sentences. Notice, however, that the language $L_u$ does not have to have function symbols for the following procedure to be correct.
Procedure 6 — 2

Procedure for Reducing and Evaluating EP-Normal Sentences

Let $u$ be an EP-normal sentence in the language $L_u$. To compute the truth value of $u$ in the standard theory of syntactic equality for $L_u$:

1. Let $u_1$ be the minimal form of $u$.
2. Let $u_2$ be the top level form of $u_1$.
3. Let $u_3$ be the component reduced form of $u_2$.
4. Let $u_4$ be the result of eliminating all of the disjuncts from $u_3$ which are literals which are not list unifiable (also delete "∃v" from $u_4$ if $v$ does not occur in the matrix of $u_4$).
5. If $u_4$ has an empty matrix, then $u$ is false; otherwise $u$ is true.

end of procedure.

If $u$ is true and $u_4$ does not contain the disjunct "∃((),())", then $u_4$ of step 4 of the procedure gives a simplified version of $u$. If $u_4$ does contain "∃((),())", then $u$ is true, and can be replaced by the empty conjunction sentence. The same thing can be done to $u_4$ as mentioned above, i.e. the elimination of equality literals subsumed by other equality literals in $u_4$, and the deletion of multiple occurrences of the same corresponding pair of terms in a literal. The evaluation time for Procedure 6-2 is also linear in the size of $u$. 
The evaluation of EAN-normal sentences has not been worked out completely. It is seen from the procedures for the EN-normal and EP-normal sentences that the language Lu does not enter explicitly into the evaluation procedures except for the condition that there be at least one function symbol for the EN-normal procedure to be correct. This is no longer true for any procedure able to evaluate the EAN-normal sentences. This is illustrated by the following example EAN-normal sentence, s:

\[ s = \exists x \forall y: \neg \exists((x),(a)) \land \neg \exists((x),(f(y))). \]

s is true if Lu is given by

\[ \text{desc}(Lu) = \langle \exists(-,-),\{a,b,f(-)\} \rangle \]

but is false if Lu is given by

\[ \text{desc}(Lu) = \langle \exists(-,-),\{a,f(-)\} \rangle. \]

We give below a procedure which sometimes can detect when an EAN-normal sentence is false, sometimes detect when it is true, and sometimes will answer "unknown".

6.2.5 Let Y be a term which occurs as an element in one of the two lists of an equality literal, q, i.e., for some term, t, either \( < t, Y > \) or \( < Y, t > \) is a corresponding pair of terms in q. Then Y is said to occur at the top level in q. The term \( Y \) is said to occur at the top level in the sentence \( u \iff v \) if \( u \) contains a literal in which \( Y \) occurs at the top level.

\[ \text{example: } Lu \exists x \exists y \forall v: \neg \exists((x,g(y)),(v,h(v))), \text{ the terms } x, g(y), v, \text{ and } h(v) \text{ all occur at the top level. Specifically, the variables } x \text{ and } v \text{ occur at the top level, but the variable } y \text{ does not occur at the top level.} \]
Procedure 5

Partial Procedure for Reducing and Evaluating EAN-Normal Sentences

For u an EAN-normal sentence in the language L_u, where L_u contains at least one function symbol of degree greater than zero, the truth value of u in the standard theory of syntactic equality for L_u is computed as follows:

1. Let u_1 be the minimal form of u.
2. Let u_2 be the top level form of u_1.
3. Let u_3 be the component reduced form of u_2.
4. Let u_4 be the result of eliminating all of the conjuncts from u_3 which are literals which are not list unifiable (also eliminate "\exists v" or "\forall v" from u_4 if v is not in the matrix of u_4).
5. If there are no universally quantified variables in the matrix of u_4, go to step 6. If there are universally quantified variables in the matrix of u_4, but none of these occur at the top level of u_4, then go to step 7. Otherwise, let v be a universally quantified variable symbol in u_4 which occurs at the top level in u_4, and let Q_v denote its earliest top level occurrence in the matrix of u_4. Let Q_v be the equality literal containing the occurrence Q_v. Let t be the corresponding term to Q_v in the other argument list in Q_v. Delete the occurrence of "\exists v" from the quantifier list of u_4. Let u_1 = (u_4)(t/v). Go to step 2.
6. If the literal \neg \exists(()()) is in u_4, then u is false, otherwise u is true. STOP.
7. If the literal \neg \exists(()()) is in u_4, then u is false, otherwise the truth value of u is unknown.

end of procedure.
Again, when $u$ is true, $u^4$ is a simplified form of $u$ and may be used to replace it as the $U$-constraint sentence. Also, $u^4$ is a simplified form of $u$ when the procedure terminates in step 7 and the truth value of $u$ is unknown. When treating EAN-normal sentences by the above procedure, and the procedure terminates with the value unknown, then one possible expedient is simply to consider the sentence as feasible. Thus the clause it belongs to would be feasible and would be retained. This maintains completeness for refinements which are complete to begin with. The adaptation of Procedure 6-3 to the case of a finite Herbrand universe involves just modifying step 6 so that if $-\hat{e}((().()))$ is not in $u^4$, then the truth evaluation of $u$ is the result of the evaluation of $u^4$ by substituting on the variables in $u^4$, and if $u$ is true, $u^4$ can be taken as its simplified form.

At the present time no complete algorithm has been devised for EAN-normal sentence truth evaluation over arbitrary languages. The evaluation of sentences of $L_u$ of more general form has not been investigated at all. Note that all of the $U$-clause refinements discussed in this report individually generate either EAN-normal sentences (and most of these are just EN-normal) or EP-normal sentences. It will be necessary to develop procedures for evaluating the $U$-constraints that result when refinements using, for example, EP-normal constraint sentences are combined with refinements using EAN-normal sentences.
6.2 HUSL and U-constraint Growth Rate.

Clearly there is the potential for rapid growth in the length of constraints of both types. The discussion here is applicable to both HUSL constraints and U-constraints, and has much in common with the discussion of growth rates for FSL constraints in NL-resolution [Sandford, 1980].

For a refinement such as the elimination of latent initial tautologies, where initial clauses are given constraints, and generated clauses obtain their constraints only by inheritance from parent clauses, a crude approximation to the average length of constraints on clauses at level \( k \) is \( b^k(2^k) \), where \( b \) is the average length of the constraint on an initial clause. This clearly leads to enormous constraints well within the depth range of interest in resolution searches (tens of levels deep). While storage problems for such large constraints can be mostly avoided by the structure sharing technique [Boyer and Moore, 1972], this technique does not help, and in fact exacerbates, the large evaluation effort that would be expected for a large constraint.

The only apparent way around this problem is to institute manipulations on constraints which reduce their size without excessive (or any, if possible) loss of constraining power. As an example, suppose we have \( \text{desc}(L) = \{ \langle P(-) \rangle, \{ a, b, f_1(-), f_2(-), \ldots, f_9(-) \} \} \), and the \( U \)-clause

\[
P(x); \ U = \{ \exists x \forall y_1 \forall y_2 \ldots \forall y_8:
\neg \exists(x, (f_1(y_1))) \land \neg \exists((x), (f_2(y_2)))
\land \ldots \land \neg \exists((x), (f_8(y_8))) \}.
\]
Then this U-clause will be equivalent to the set of 3 U-clauses:

\[ P(\varphi(x)); \ U = \{\} \]
\[ P(a); \ U = \{\} \]
\[ P(b); \ U = \{\}. \]

In this case so many prohibitions on the value of \( x \) are explicitly stated in the U-constraint of the original clause that it requires less space to just split the clause into the individual cases which explicitly show what \( x \) could in fact be. Another alternative is to replace the original clause above by

\[ P(x); \ U = \{\exists x \exists y: \ \varphi((x),(a)) \lor \varphi((x),(b)) \lor \varphi((x),(\varphi(y)))\}. \]

No implementation of a HUSL or U-constraint refinement exists, and so it is difficult to judge just how severe a problem constraint size will be and what type of techniques are appropriate. However, one can assume that in situations such that

1. the Herbrand universe has many terms of low function nesting depth (i.e. many constants, or many function symbols, or function symbols of high degree), or the clauses are such that relatively deep nesting of function symbols occurs early in the search, and

2. the refinement adds to each generated clause new constraints (HUSL literals or equality literals) beyond those inherited from the parent clauses,

it would be unlikely that constraint reductions which lose no information will be adequate to control constraint size. In such cases heuristic choices of what to keep and what to delete will have to be made. This area has not been investigated, and awaits empirical data as a guideline.
7.0 Other Refinements and General Discussion.

7.1 Other Refinements Based on U-Constraints.

The reader should by now have a good intuitive feeling for the way refinements can be stated using HUSL and U-constrained clauses. In this section we mention, without full details, some additional refinement possibilities which generate U-constraints in EAN-normal form. In section 7.1.4 we present the only example in this report which generates U-constraints which are EP-normal sentences.

7.1.1 A non-minimal deduction [Kowalski and Kuehner, 1971], D, is a resolution deduction such that there exists a branch of D along which the same atom is resolved away more than once. A minimal deduction is a deduction which is not non-minimal. The minimal deduction refinement is the refinement consisting of all minimal deductions.

A ground faithful form of the minimal deduction refinement can be effected by using U-clauses. The initial set of clauses is converted to a U-clause set by adding an empty U-constraint to each clause. For each resolution step, the resolvent is first formed as a direct U-resolvent of its parents, and then to the constraint of the resolvent is added the distinctness literals for the atom just resolved on and all of the atoms resolved on earlier in the deduction tree of the newly formed resolvent. (N.B. These previously resolved atoms in the tree must be "updated" by having the appropriate substitutions applied to them before the distinctness literals are formed.) This refinement can be changed into a locally defined refinement by adding a third component to each U-clause, which is a
set consisting of all of the atoms resolved on in deducing that clause. The \( U \)-constraints generated by this refinement are EN-normal sentences. This refinement can be made stronger by adding to the \( U \)-constraint of each newly generated clause the distinctness literals not only for the atom just resolved on and earlier resolved atoms, but also all literals in the generated clause and all earlier atoms resolved on.

7.1.2 In some linear resolution strategies [Kowalski and Ruhner, 1971] [Chang and Lee, 1973], and in the Graph Construction procedure [Shostak, 1976], literals which are resolved away are retained in clauses in a special configuration. These "ancestor" literals contain information which can sometimes signal that two clauses need not be resolved together, and sometimes allows certain "ordinary" literals of a clause to be eliminated without any explicit resolution step (this is called a reduction step). For example, in the clause

\[
C = A(x), B(f(x)), [C(g(y))]_1, D(f(a)), \neg C(g(x))
\]

where the ancestor literal is enclosed in a box and the "active" end of the clause is the right-hand side, it is possible to apply the substitution \( \sigma = \{ y/x \} \) to obtain

\[
C\sigma = A(y), B(f(y)), [C(g(y))]_1, D(f(a)), \neg C(g(y))
\]

which can then be reduced to

\[
C_{R} = A(y), B(f(y)), [C(g(y))]_1, D(f(a))
\]

The original clause \( C \) must also be retained in the inference space. At the ground level, however, it can be shown (for a typical linear strategy) that refutation completeness is maintained under the
restriction that when a reduction is possible, it is in fact mandatory, i.e. no reducible clause is ever resolved on. This allows a refutation complete U-clause refinement of the following kind to be stated (consider C, CR above to now also have U-constraints on them): In a case such as clause C above, replace C by two clauses,

i. CR, as above, but where the U-constraint on CR is the same as that on C with the substitution \( \sigma \) applied to it;

ii. \( C' \), where \( \xi(C') = \xi(C) \) and the U-constraint of \( C' \) is the same as that of \( C \), except that its matrix also contains a conjunct which is the distinctness literal for \( C(g(y)) \) and \( C(g(x)) \), and if necessary existential quantifiers for "x" and "y" are added to the prefix of the U-constraint.

The rule just given would have to be elaborated upon somewhat to cover cases where several different substitutions and ancestor literals could be used in the reduction operation. Here also the constraint sentences generated are EN-normal sentences.

7.1.3 For some complete resolution refinements it is quite difficult to determine if the refinement is complete when general subsumption is added to it (see, for example, the difficulties for semantic refinements with ordered clauses in [Loveland, 1978]). This problem does not occur with the elimination of latent initial subsumptions described earlier. Another, somewhat peculiar form of subsumption, called condensation [Joynar, 1976], also seems to be compatible with many refinements in the sense of maintaining completeness.
Clause \( C \) is said to be a condensate of clause \( D \) iff all of the following hold:

i. \( \exists \sigma : C = D\sigma \),

ii. \( C \) subsumes \( D \),

iii. no substitution instance of \( C \) subsumes \( C \) and has fewer literals than \( C \).

For each clause \( D \) generated in a search, \( D \) is to be replaced by its condensate [N.2. Joyner has shown that all condensates of a clause are identical except for a re-naming of variables, at least for the situation where clauses are sets of literals without any additional structure such as lock numbers [Joyner, 1976]]. In many cases, a generated clause is its own condensate.

Replacement of a clause by its condensate is essentially replacing a clause by a factor of itself which happens to subsume it. For example let \( D \) be a generated clause:

\[
D = P(x), P(f(y)), R(z,a), R(b,w);
\]

then the condensate, \( C \), of \( D \) is:

\[
C = P(f(y)), R(z,a), R(b,w);
\]

Thus we would eliminate \( D \) and keep \( C \) in its place. But more than this can be done, as follows. Suppose now that the above \( C \) is a \( U \)-clause with an empty \( U \)-constraint. If we were also using \( U \)-constraints to keep track of factoring choices then \( C \) would be split into (i.e. replaced by) the two factoring choices:

\[
\begin{align*}
Cf1 &= P(f(y)), R(z,a), R(b,w); \quad U = [ \exists z \exists w: \neg((z,a),(b,w)) ] \\
Cf2 &= P(f(y)), R(b,a); \quad U = []
\end{align*}
\]
Thus Cfl becomes infeasible whenever z becomes "b" and w becomes "a" simultaneously. However, if we apply the notion of condensation more carefully, we can obtain an even more restricted form of Cfl, as follows. C was its own condensate, and thus cannot be replaced by any single substitution instance of itself, unless that instance is a re-naming of C. But there may well be a substitution instance of C, Cfl, such that Cfl is not its own condensate, and thus can be replaced by a subset of itself. Without attempting a complete formal presentation of this notion here, we will just illustrate what happens for the case of clause C above. We notice that for some substitution, \( \tau \), \( S(Cfl) \tau \) is subsumed by \( S(Cf2) \). We are thus led to add to \( U(Cfl) \) the equality literals which prevent \( S(Cfl) \) from representing those substitution instances which are subsumed by \( S(Cf2) \). We produce the new U-constraint of Cfl by considering the two subsumption mappings possible from \( S(Cf2) \) onto proper subsets of \( S(Cfl) \), which gives the new clause Cfl'

\[
Cfl' = P(f(v)), R(z, a), R(b, w);
U = \{ \exists v \exists z \exists w \forall y' \forall y'' : \neg \delta((z, a), (b, w)) \wedge \neg \delta((v, z, a), (y', b, a)) \wedge \neg \delta((v, b, w), (y'', b, a)) \}
\]

(the \( y' \) and \( y'' \) result from re-naming the \( y \) in Cf2; \( y' \) and \( y'' \) are each used once, for separate subsumption mappings). Now \( U(Cf1') \) can be simplified to

\[
U(Cf1') = \{ \exists z \exists w : \neg \delta((z, a), (b, w)) \wedge \neg \delta((z), (b)) \wedge \neg \delta((w), (a)) \}
\]
and further simplified to just

\[ \text{\texttt{U(Cf1')}} = [ \exists x \exists w. \neg \texttt{\$((a),(b))} \wedge \neg \texttt{\$((w),(a))} \ ] \leftarrow \]

This gives Cf1' as a clause which is more constrained than Cf1, since Cf1' is infeasible for any substitution instance which substitutes "b" for \( z \) or "a" for \( w \), independently. Thus, by considering the notion of condensation for instances of a clause, and effecting this through consideration of subsumption relationships between factoring choices, we are able to replace the clause C by the pair of clauses Cf1' and Cf2, instead of the less constrained pair Cf1 and Cf2 which is obtained by only considering factoring choices on C.

7.1.4 Up to this point we have only exhibited refinements which generate U-constraints which are (either explicitly, or are equivalent to) EAN-normal sentences. There are criteria for refinements which do yield U-constraint sentences of a different nature however. One of these is the range restriction and substitution compatibility notion of Sickel [Sickel, 1974], which is an elaboration of a substitution constraint notion in [Prawitz, 1969]. In the following the intention is not to give a complete illustration of the approach in [Sickel, 1974], but rather just to exhibit one instance of a refinement condition which generates a U-constraint which is not equivalent to an EAN-normal sentence.

In Figure 9a we have an example clause set [Sickel, 1974] [Loveland, 1978] axiomatizing a well known and obvious problem situation. Figure 9b is the set of initial U-clauses resulting from the initial set in Figure 9a, by using the following criteria:
For each clause, C, in Figure 9a, clause C* in Figure 9b is to represent only ground instances, Cg*, in which every literal in Cg* is the complement of an instance of at least one literal in the initial set in Figure 9a (i.e. Cg* has no pure literals).

Call the initial set (in Figure 9b) S*. Every U-clause in S* is feasible. A refinement which starts with S* and produces UNR direct U-resolvents would avoid certain unnecessary deductions. For example, if we consider the level 3 deductions which resolve away the three "In_Room" literals of clause 3 in Figure 9a, we see that there are 162 different deductions possible in UNR, and there are 27 different root clauses that can result from these 162 deductions. From Figure 9b however, we see that clause 3* is effectively restricted to a single ground instance. Starting with 3* there are exactly 6 deductions of level 3 which resolve away the "In_Room" literals, all yielding the same root clause. Obviously this type of refinement has the potential for massive search space reduction.

Further work is needed to understand to what extent the methods in [Sickel, 1974] can be combined with the U-clause approach.
1. $\neg \text{Dexterous}(x), \neg \text{Close}(x,y), \text{Reach}(x,y)$;
2. $\neg \text{On}(r,s), \neg \text{Under}(s,\text{banana}), \neg \text{Tall}(s), \text{Close}(r,\text{banana})$;
3. $\neg \text{In\_Room}(u), \neg \text{In\_Room}(v), \neg \text{In\_Room}(w), \neg \text{Can\_Move}(u,v,w), \\
   \text{Close}(w,\text{floor}), \text{Under}(v,w)$;
4. $\neg \text{Climb}(q,t), \text{On}(q,t)$;
5. $\neg \text{Dexterous}(\text{monkey})$;
6. $\neg \text{Tall}(\text{chair})$;
7. $\neg \text{In\_Room}(\text{monkey})$;
8. $\neg \text{In\_Room}(\text{banana})$;
9. $\neg \text{In\_Room}(\text{chair})$;
10. $\neg \text{Can\_Move}(\text{monkey},\text{chair},\text{banana})$;
11. $\neg \text{Close}(\text{banana},\text{floor})$;
12. $\neg \text{Climb}(\text{monkey},\text{chair})$;
13. $\neg \text{Reach}(\text{monkey},\text{banana})$;

FIGURE 9a
The Monkey-Banana-Chair Initial Set.

$\neg \text{Dexterous}(x), \neg \text{Close}(x,y), \text{Reach}(x,y)$;
$U = \{ \exists x \exists y: \#(x,\text{monkey}) \wedge (\#(y,\text{banana}) \vee \#(y,\text{floor})) \wedge (\#((x,y),(\text{monkey},\text{banana}))) \}$

$\neg \text{On}(r,s), \neg \text{Under}(s,\text{banana}), \neg \text{Tall}(s), \text{Close}(r,\text{banana})$;
$U = \{ \exists s: \#(s,\text{chair}) \}$

$\neg \text{In\_Room}(u), \neg \text{In\_Room}(v), \neg \text{In\_Room}(w), \neg \text{Can\_Move}(u,v,w), \\
   \text{Close}(w,\text{floor}), \text{Under}(v,w)$;
$U = \{ \exists u \exists v \exists w: (\#(u,\text{monkey}) \vee \#(u,\text{banana}) \vee \#(u,\text{chair})) \wedge (\#(v,\text{monkey}) \vee \#(v,\text{banana}) \vee \#(v,\text{chair})) \wedge (\#(w,\text{monkey}) \vee \#(w,\text{banana}) \vee \#(w,\text{chair})) \wedge \#((u,v,w),(\text{monkey,chair,banana})) \wedge \#(w,\text{banana}) \}$

$\neg \text{Climb}(q,t), \text{On}(q,t)$;
$U = \{ \exists q \exists t: \#((q,t),(\text{monkey,chair})) \}$

Clauses 5* through 13* all have empty U-constraints.

FIGURE 9b
The Constrained Transform Set corresponding to Figure 9a.
7.2 Completeness of HUSL and U-constraint Refinements.

It is believed that all of the HUSL and U-constraint based refinements presented in this report are refutation complete; however detailed completeness proofs have not been constructed for any of them. The completeness of the elimination of latent initial tautologies is fairly obvious, and holds when it is combined with LR, TMS and other complete semantic refinements, linear resolution strategies, etc. The elimination of latent generated tautologies is also easily seen to be complete when used with refinements which are complete when explicit tautologies are deleted. This, of course, excludes LR.

The elimination of latent initial subsumptions is less obviously complete. A detailed completeness proof for this would have to make it clear that there is no possibility of circularity of subsumption occurrences which results in the transformed set (e.g., $S_{sub}$ in Figure 7b) being overconstrained. To prevent this from happening is why both E-SUB and P-SUB were utilized in the definition of the initial set transformation, and why the original set of clauses (e.g., $S$ in Figure 7a) was assumed closed under factoring and was ordered in a particular way. That these measures are adequate to yield a refutation complete refinement is intuitive, but has not been proven.

7.3 Ground Faithfulness.

As seen by the variety of examples given in this report, the notion of ground faithfulness, and of attaining it by the use of U-clauses, has wide applicability. In general, any syntactic
refinement condition which is substitution sensitive is a potential candidate for implementation as a U-clause refinement in a ground faithful form. Also certain semantic refinement conditions can be implemented as ground faithful refinements by use of a different type of constraint on a clause [Sandford, 1980].

One can view the use of U-constraints (and to a lesser extent HUSL constraints) as a sharpening of the selectivity of the most general unifier (mgu) notion. Originally the mgu was viewed as a way of avoiding the instantiation of clauses to ground instances [Robinson, 1965a] while still being able to detect unsatisfiability. Unfortunately, unsatisfiability is a distributed property over a set of clauses, while the usual notion of the mgu is a local notion over small sets of literals. The use of U-clauses to achieve ground faithfulness is a mechanism whereby the mgu is effectively restricted on the basis of more global criteria than is possible in the mgu itself. The full power of such restrictions on the mgu have yet to be explored [note that all of the U-constraint examples of this report are examples of achieving ground faithfulness for refinements conditions that are already well known, and generally applicable, such as subsumption and tautology elimination, minimal deductions, etc.; it should be obvious that many others exist].

Ground faithfulness is an important notion in preventing the undermining of the intent of a refinement. An example of such undermining is Figure 4. Suppose we consider the clause set in Figure 4a to be an initial set with clause 5 as the set of support [Wos, 1965], and the deductic of Figure 4b (without HUSL's) as a set
of support deduction, with explicit tautology elimination, of clause 11. Notice that clause 11 is an instance of the clause directly obtainable by resolving the two initial clauses, clause 3 and clause 6. But 3 and 6 cannot resolve together because neither is supported. Thus in a very real sense, the failure to detect latent tautologies circumvents the refining power of set of support. Many other examples of undermining of refinements by failure to detect latent generated tautologies are easily discovered. It should be intuitively clear that similar situations arise for the other substitution sensitive refinement conditions, such as latent initial subsumption relationships.

7.4 More Elaborate Refinements.

There are several ways in which the U-constraint approach can be generalized or extended.

The first is that of replacing the notion of truth of a U-constraint sentence in the standard theory of syntactic equality with truth in a different model. For example, in group theory problems, if "I" is the inverse function, one might want the U-sentence

\[ \exists x: \neg \exists (x), (I(I(x))) \]

to be false. While doing this makes intuitive semantic sense, the conditions under which refutation completeness would be preserved remains to be explored. It would seem that refutation completeness would only be maintained with such a different evaluation of U-constraints if the syntactic notion of unification for the
resolution operations was generalized in the appropriate manner.

A second extension is that of allowing interaction between semantic constraints [Sandford, 1980] and U-constraints. If a clause, \( C \), has both a semantic constraint, \( s \), and a U-constraint sentence, \( u \), it is possible that \( s \) and \( u \) are each individually feasible, but there is no (pure existential) grounding substitution, \( \sigma \), for \( C \) such that \( s \sigma \) and \( u \sigma \) are both feasible. Future work is needed to find viable mechanisms so that the constraints \( s \) and \( u \) can be identified as jointly indicating that \( G(C) \) is empty.

A third extension is what we call a priori exclusion, and involves adding U-constraints to initial clauses which are not generated by any generally applicable refinement conditions, but rather arise because of some specific fact we know, or guess, about the particular problem to which they are applied. For example, in problems in reasoning about actions, one might add U-constraints to prevent certain unwanted substitution instances of initial clauses. These usually would be "nonsense" substitutions, which, even if they are involved in refutations, the refutations would not be considered of use in the original context of reasoning about actions. Of course, when such restrictions are added, refutation completeness may be lost.

Finally we mention an extension of the notion of the truth evaluation of a constraint to include accessibility of an existentially quantified variable to classes of ground terms. The definition of truth of a constraint does not presently take into account the possible future course of a deduction with respect to possible instantiations of variables. In its most simplistic form,
such information could be incorporated by "typing" of the variables.

7.5 Generality of the Examples Used.

Our examples have mostly involved clause sets axiomatizing equivalence relations. The combination of two or more among (reflexivity, symmetry, transitivity) for a binary relation leads to instances of tautologies or subsumption relationships in deductions which are shallow, and thus make for easy examples. However the same types of inefficiencies arise for almost any set of clauses (unless, of course, there are no variables at all, or other special conditions hold). In particular, the common axioms of mathematical systems, such as associativity and distributivity of one operator over another, are very rich sources of latent initial tautologies. Until adequate experimental work is done there is no absolute way to know the magnitude of these inefficiencies, but they seem to be a major source of inefficiency in resolution for most domains.

7.6 Forbidden Pairs: A Simplified Refinement With Low Overhead.

In [Sandford, 1980] some empirical results were obtained with a theorem prover using, among other strategies, a refinement which is a weak form of the combination of the latent initial tautology elimination and latent initial subsumption elimination strategies illustrated in this report (LR with explicit factoring was the basic inference rule). We call this refinement FF (for Forbidden Pairs), and FF operates as follows. Each literal occurrence in an initial set, S, is assigned a unique integer. When using FF with LR, if S has all distinct lock numbers on its literal occurrences, then the lock
numbers themselves can be used as the integers on literals. When forming new clauses, the literals in the generated clause inherit the integer assigned to the parent literal (if there are two parent literals for a literal then arbitrarily choose one of the parent literal integers). There is a symmetric binary relation, $\mathcal{R}$, defined on integers, such that in a resolution step, the pair of integers from the two literals resolved upon must not be in the relation $\mathcal{R}$. $\mathcal{R}$ is thus a relation specifying "forbidden pairs" of integers on literals for resolution. $\mathcal{R}$ is constructed as follows.

Let literal(i) be the literal occurrence in $S$ which has been assigned integer i.

Let C(i) be the clause in $S$ containing literal(i).

Then $\mathcal{R}(i,j)$ is true iff there exists a $\sigma$ such that both of the following hold:

1. (unifiable {literal(i), ~literal(j)})
   and $\sigma = \text{mgu}(\text{literal}(i), \text{~literal}(j))$;

2. at least one of the following is true:
   i. (C(i))$\sigma$ is a tautology;
   ii. (C(j))$\sigma$ is a tautology;
   iii. (C(i))$\sigma$ is properly subsumed by some clause in $S$;
   iv. (C(j))$\sigma$ is properly subsumed by some clause in $S$.

(Where C properly subsumes D iff C subsumes D and C has fewer literals than D).

The FP refinement is a mixture of a level 1 and deeper level refinement conditions. It is clearly a level 1 refinement in the sense that it can only "see" classes of substitution instances of initial clauses which result from unifying two level 0 (i.e. initial)
literals. On the other hand, once \( \mathcal{F} \) is computed, it is used at all depth levels of the search.

FP is a low overhead refinement, since generally the cost of constructing \( \mathcal{F} \) for a given \( S \) is small compared to the expected search effort of a resolution search starting from \( S \), and the test is simply a table or matrix look-up on integer pairs once \( \mathcal{F} \) is constructed. A small amount of experimentation was done with FP in the context of a semantic resolution refinement based on LR [Sandford, 1980], in which no other form of subsumption or tautology elimination was involved. The reduction in number of clauses was highly dependent upon the exact nature of the initial set, but on average was sufficiently large so as to unreservedly argue for the desirability of using FP. FP would be of less utility in conjunction with the simplification strategies of explicit tautology elimination and subsumption, particularly if FP is combined with UNK.

In general it is unclear under what conditions FP is sufficiently strong so that it is unnecessary to use a \( G \)-constraint strategy such as \( ^G_{\text{LR}} \), with its larger overhead. In this connection we point out that experiments designed to determine if a strategy such as \( ^G_{\text{LR}} \) is worthwhile or not, are to be judged not relative to LR, but rather relative to LR combined with FP. The FP refinement is clearly a special case of the scheme of [Overbeek, 1976]. and conditions 2 i-iv (above in the definition of \( \mathcal{F} \) can be modified to include other criteria.
The FP type of restriction also has other uses. For example, it has been shown [Boyer, 1971] that for the transitivity axiom with lock numbers,

\[ \neg \langle x, y \rangle ^2, \neg \langle y, z \rangle ^2, \langle x, z \rangle ^2 \]

that completeness is maintained in LR when the restriction is added that no literal of index 1 is ever resolved against a literal of index 2 (assuming no initial clause other than transitivity is using lock numbers 1 and 2). This is obviously nicely implemented as an FP type of refinement.

7.7 General Conclusion.

The problem of constraining a resolution search to a subset of the possible substitution instances of clauses can be accomplished by the mechanisms of U-constraints, HUSL constraints and refinements of the FP type, in decreasing order of restrictiveness. The appropriateness of at least the FP type of refinement is assured simply because of its negligible overhead in certain forms. Experimental work is needed to determine if the increased overhead in U-constraint refinements is paid back in the reduction of the number of clauses generated. Because of the diverse nature of the various refinement conditions, it is likely that some may be appropriate for U-clause implementations, while others are adequately implemented as HUSL clauses or FP type refinements.

The notion of ground faithfulness is seen to be of utility as the principle by which substitution sensitive refinement conditions are to be strengthened. This requires emphasis on entire deductions rather
than individual (ordinary) clauses. The information contained in these deductions can often be summarized as a U-constraint sentence, and included as part of a U-clause. This allows, in implementations, a shift of emphasis back to the individual constrained clauses, instead of the entire deduction.
BIBLIOGRAPHY


INDEX OF DEFINITIONS

arity 13, 20
binary equality literal 38
binary LR direct R-resolvent 32
binary resolution 11
binary tree 10
branch 10

clause tree 10
component reduced 52
condensate 75
constrained 38
constraint 22, 37
core 51
corresponding pair 42
corresponding terms 42

Δ 13
deduction 11
deduction tree 11
defining property 16
desc 13
direct R-resolvent 25
direct U-resolvent 56
distinctness literal 24, 25
distributed subsumption 55

d 25
e-sub 53
e-sub-C (E-subsumption conjunct) 53, 54
EA sentence 39
EAN-normal sentence 39
EN-normal 39
EP-normal 39
equality literal 38
equivalent literals 42
equivalent sentences 41
existential ground based property 18
explicit factoring 11

factor 50
feasible 23, 30
for all (function) 42
forbidden pairs (FP) 84ff.
fully factored 51

G 15, 45
G* 45
Γ 15
general clause 14
graft 10
gross satisfaction set 40
ground 14
ground based property 18
ground extension function 15,45
ground extension set 2,15
ground faithfulness 19
grounding 14,15

HU$SL$ 22
HU$SL$-clause 22
HU$SL$ literal 22

i-component reduction 40
implicit factoring 11
infeasible 23,38
inference step 12
insensitive to substitutions 10
instantiation invariant 8

k-factor 31,32

languages 13
latent initial subsumption transform 54
latent initial tautology 3
latently P-subsumes 52
lift 16
liftable 18
list unifiable 63
locally defined 17
locally negatable 16
low-lock 31,32
LR 2
Lu 36

M (function) 45
M' 46
major subdeductions 13
mgu 31
minimal deduction 18,72
minimal form 63
minimal grounding substitution 14
minimal satisfaction set 41

occur at the top level 67
order (of a satisfaction set) 40

P-SUB 52
P-SUB-C (P-subsumption conjunct) 53
P-subsumes 52

pure existential substitution 44
Page 94

reducible (on component 1) 40
refinement 16
resolution tree 11
root 10

S 22,37
S_{FAC} 32
\omega_{Stat} 25
\omega_{Stat} 47
S_{Sub} 51,54
satisfaction set 40
semantic function 36
simple factor 50
standard literal set 22,37
standard theory of syntactic equality 37
stretch 10
strict binary tree 10
substitution sensitive 18
subsume (equality literals) 43
subsumption substitution (for \omega) 52,53

\therefore \exists (function) 41
TMS 17
top level form 62

\mathcal{U} 22,37
U-clause 37
U-clause resolution tree 38
U-constraint sentence 37
unconstrained 38
UNR 11
UNR-resolution tree 11
variables (function) 41