A Linear Deduction System with Integrated Semantic Guidance

by

Geoff Sutcliffe (BSc, BSc (Hons), MSc)

This thesis is presented for the degree of Doctor of Philosophy
at The University of Western Australia, Department of Computer Science, 1992.

A Linear Deduction System with Integrated Semantic Guidance

Abstract

Guidance systems are central to the success of automated deduction systems. Semantic guidance systems are guidance systems that exploit semantic information when guiding the search of a deduction system. The general objective of this research has been to investigate how semantic guidance can be used to improve the performance of automated deduction systems. More specifically, this research has investigated how semantic guidance can be used to improve the performance of linear deduction systems. As semantic guidance has, until now, been considered unsuitable for use in linear deduction systems, the results presented in this thesis are noteworthy in automated deduction research.

A new chain format linear deduction system, called Guided Linear Deduction (GLD), has been developed as part of this work. GLD improves upon existing linear deduction systems in several aspects. In the context of this research, an important feature in GLD is the provision of an explicit entry point for the incorporation of guidance systems. This entry point has been used to incorporate a semantic guidance system into GLD, to form the Semantically Guided Linear Deduction (SGLD) system. SGLD's semantic guidance system builds upon four separate developments, as follows. (i) Linear-input subset analysis, which is a method of determining some of the structure in GLD deductions. (ii) A truth value semantic deletion system for (chain format) linear deduction systems. This system uses linear-input subset analysis to determine when it can be applied. (iii) A sort value semantic deletion system that has the same format as the truth value deletion system. (iv) A heuristic function that uses semantic information to evaluate the quality of clauses in a deduction. The combination of the latter three of these developments forms the semantic guidance system used in SGLD. For any semantic guidance system to operate, an interpretive structure is required to store the semantic information used. It is desirable that the interpretive structure be representationally powerful, space efficient, effective in supplying semantic information and also user friendly. A new form of interpretive structure which fulfils these criteria has been developed. The new structures are called designations.

SGLD has been implemented in Prolog. Each component of SGLD is of individual interest and their combination is unique in the field of automated deduction. The performance of SGLD has been investigated.

Contents

Chapters and Appendices

Definitions

Acknowledgments

Thanks to (i) Dr C.P. Tsang for supervising my research, (ii) Winston Tabada for inspirational discussions and (iii) fellow PhD students (especially MarkE, James, Nick and MarkN) for tolerating me.

Chapter One

Introduction and Technical Preliminaries

Guidance systems are central to the success of automated deduction systems. Semantic guidance systems are guidance systems that use semantic information to determine search direction for their host deduction systems. Relative to their potential for improving the performance of deduction systems, it seems that semantic guidance systems have been badly neglected. All guidance systems are necessarily formulated to be compatible with their host deduction systems. Beyond this constraint, it is desirable to abstract the issues of syntax, application and implementation, as far as possible. This chapter introduces these issues and presents the notation and terminology used in this thesis.

This chapter contains :

1. A brief background to automated deduction systems.
2. Motivation for semantic guidance of automated deduction systems.
3. The objectives of this research.
4. Definitions of the language and notation used in this thesis.
5. An abstract discussion of the nature and imposition of restrictions in deductions.
6. A preview of the contributions made by this thesis.
7. An overview of the content of the thesis.

1.1 Background

Although automated deduction has yet to prove itself as a commercially viable technique for solving real-world problems, there is no doubt that the time will come when it will play a major and day-to-day role in society. Application areas of automated deduction systems include "expert systems, planning, common sense reasoning, proof checking, instruction and aids to human mathematicians" [Plaisted, 1990a, p. 270]. The impact of transferring the general task of problem solving from man to machine is significant, and research efforts to this end are easily justified on both economic and social grounds. This thesis documents research into the use of *semantic information* (i.e., information that is specific to the problem domain) in guiding automated deduction systems.

There is no point in including a survey of the general topic of automated deduction in this thesis, as this task has been performed by many authors. The surveys of Bledsoe and Hodges [1988], Stickel [1986a] and Plaisted [1990a] are recommended. These surveys also provide references to introductory and advanced texts. Familiarity with the terminology of resolution based automated deduction is henceforth assumed.

It is clear that the basic resolution procedure [Robinson J.A., 1965a] is inadequate for solving anything but the most trivial problems. The combinatorial growth of the resolution procedure's search space soon swamps available computing power, regardless of the efficiency of implementation. Deduction systems that refine the basic procedure are thus of interest. The basic resolution procedure is refined by imposing *restrictions* on the deductions that are built, e.g., unit resolution imposes the restriction that one parent of each resolution operation must be a unit clause. Although refined deduction systems may be categorised according to the nature of the restrictions used (see section 1.5 for example), upon abstraction it is evident that they all have the same essential feature. That is, they impose restrictions that assist in deciding which deduction operation and which of the available clauses will be used at each step of a deduction. In this manner the restrictions guide the search of the deduction system.

A broad spectrum of restrictions have been used to refine the basic resolution procedure. At one extreme of the spectrum are restrictions that require a certain deduction operation or clause to be used in certain circumstances. At the other extreme are restrictions that exclude a deduction operation or clause from use. Both these extreme forms are called *absolute restrictions*. Absolute restrictions guide the search of a deduction system by reducing its search space. An example of an absolute restriction, that requires a certain deduction operation to be used, is compulsory reduction in chain format linear deduction systems (see chapter 2). Examples of absolute restrictions, that prevent certain clauses from being used, are subsumption [Robinson J.A., 1965a], admissibility restrictions (see chapter 2) and semantic deletion (see chapter 3). In between the two absolute extremes are *preferential restrictions*. Preferential restrictions do not reduce the search space of a deduction system, but rather guide a deduction system by indicating a preference for the use of certain deduction operations and/or clauses. Examples of preferential restrictions are the unit preference strategy [Wos, Carson, & Robinson G.A., 1964] and restrictions that use a heuristic function to guide the search of the deduction system, e.g., those described by Overbeek, McCharen and Wos [1976].

When discussing deduction systems, it is useful to separate the restrictions used from the underlying deduction mechanism. The restrictions are considered to form a *guidance*

system, which is used by the *host deduction system*. Restrictions and guidance systems are categorised as either *syntactic* or *semantic*, depending on the type of information used.

1.2 The Necessity for Semantic Guidance

One accepted description of intelligent action is, as expounded by Newell and Simon in their Turing award lecture [1976], the ability to solve problems via state space search. For some problems, algorithms have been devised so that no search is necessary. These are the algorithms of traditional computer science; the so-called strong methods. The balance of intelligent action is necessarily to be achieved by genuine state space search. Methods which employ search (including automated deduction) constitute the weak methods of computing. The strong methods depend on semantic information which is encoded directly into the algorithms' steps. It is reasonable to hypothesize (but no attempt is made to prove) that no strong methods of computing can be domain independent. The weak methods of computing can be divided into two groups; those that use semantic information to guide their search and those do not. It is also reasonable to hypothesize that weak methods that do not use semantic information will never be able to conquer the exponentially growing search spaces of hard problems. This view is supported by Newell and Simon, who claim that physical symbol systems "exercise intelligence by extracting information from a problem domain and using that information to guide their search ..." [Newell & Simon, 1976, p. 126] Semantic information thus appears to be prerequisite to intelligent action[1].

In the area of automated deduction, the need to use semantic information is translated into a need to use semantic guidance systems. The need for semantic guidance has been noted in the literature, e.g. "An emphasis on semantics rather than on syntax has far greater potential for producing a dramatic impact on the power of automated reasoning programs" [Wos, 1988, p. 257] (research towards a "semantically orientated strategy" is Problem 5 in [Wos, 1988]) and "... if searches in symbolic computation are not to fall prey to combinatorial explosion, they must incorporate domain-specific knowledge in such a way so as to give direction to the search." [McRobbie, Meyer, & Thistlewaite, 1988, p. 198]. Despite the acknowledged need for semantic guidance, the overwhelming majority of guidance systems that have been developed to date are syntactic. The lack of attention paid to semantic guidance has been noted by leading researchers. Bledsoe and Henschen, in their contribution to the first issue of the Journal of Automated Reasoning [1985, p. 27],

[1] For readers who do not subscribe to this mechanistic view of intelligent action, it is suggested that "Since humans use semantics (models) extensively in proving theorems, it seems natural that computer theorem provers should also." [Plaisted, 1990a, p. 308]

prophesy that "It seems likely that such approaches [to guiding deduction systems] will have to rely on the semantics of problems to a much larger degree than in the past.". Alan Bundy has stated [1987] that "There has not been nearly as much work on semantic checking as I think it deserves ...". It should be noted that the dearth of semantic guidance systems is not due to their importance being acknowledged only recently. In 1973 it was claimed that "Virtually everyone is now agreed that knowledge about the problem domain must be used in the logic." [Reiter, 1973, p. 41]. Rather, the probable reason for the imbalance is the ease with which syntactic guidance systems can be designed and implemented, compared to the complexity of semantic guidance systems.

In light of the above, research into the semantic guidance of automated deduction systems is well justified.

1.3 Research Objectives

The overall objective of this research has been to investigate how semantic guidance can be used to improve the performance of automated deduction systems. This overall objective has been broken down into four subobjectives. (i) To develop a host deduction system. (ii) To develop a semantic guidance system for the host deduction system. (iii) To develop an interpretive structure for storing the semantic information used by the semantic guidance system. (iv) To combine the deduction system, semantic guidance system and interpretive structure, into a coherent whole.

In fulfilling the first subobjective, to develop a host deduction system, a requirement has been that the host deduction system be effective in its own right. This ensures that any favourable effects of incorporating semantic guidance into the deduction system are not attributable to the inadequacy of the deduction system. If the host deduction system is weak, then the addition of any guidance will improve its performance. On the other hand, if the performance of an effective deduction system is improved through the addition of a guidance system (here a semantic guidance system), then the effectiveness of the guidance system is clearly demonstrated. A preliminary step in this task has been to select a basic deduction format. In the context of this research, it has been necessary that the deduction format produce deductions with a relatively uncomplicated structure, as this eases the design of an appropriate semantic guidance system. The choice has been the linear format (see section 2.1.1), and the subobjective of developing the host deduction system has proceeded from that starting point.

The choice of the linear format for the host deduction system was made independently of known compatibility with semantic guidance. The (now more specific) subobjective of

developing a semantic guidance system for a linear deduction system is interesting, as linear deduction is known to be incompatible with truth value (semantic[2]) deletion (see section 3.2.1). The use of semantic guidance in linear deduction systems has, however, been suggested :

- "Another interesting question ... is whether there exists a decision procedure ... for recognising ... whether there is a proof tree ... satisfying $\tilde{R}_1 \cap \tilde{R}_3$ [model resolution ∩ ancestry-filter format resolution]." [Luckham, 1970, p. 173]
- "As a heuristic, combining the two strategies [the ancestry-filter and model strategies] may often yield efficient searches for refutations." [Nilsson, 1971, p. 227]

As no other semantically guided linear deduction system appears to have been developed and implemented, the fulfilment of this subobjective has been a step towards extending the use of semantic information in automated deduction systems.

The third subobjective, to develop an interpretive structure for storing the semantic information used by the semantic guidance system, is pragmatically motivated; the efficacy of a semantic guidance system is limited by the supply of semantic information. To meet this subobjective, two criteria have had to be met. Firstly, the interpretive structure has to be computationally efficient. If this criteria is not met then the client semantic guidance system is unlikely to be of utility. In order to fulfil this criteria, only bodies of semantic informtion with finite domains have been targeted. Secondly, as the original source of semantic information is typically human, it has been required that it be reasonably easy to specify the required semantic information.

The final subobjective, to combine the outcomes of the first three subobjectives, is aimed at developing a coherent semantically guided deduction system, i.e. one in which the three components fit together in a natural manner. This subobjective has necessarily had to have an influence on the first three. It has, therefore, been a rider throughout that the host deduction system, the semantic guidance system and the interpretive structure should be cognisant of each others features. It has also been an objective to produce a full implementation of the combined system. The implementation has aimed to be portable and easily updated.

---

[2] Throughout this thesis, words such as "semantics", "semantic guidance", "interpretation", etc. are meant in a generic sense, rather than the common usage which associates them with truth values. Wherever a specific type of semantic information needs to be associated with such terms, the association is made explicit. Thus "truth value deletion" refers explicitly to rejection of clauses based upon their truth value interpretation.

In light of the above, the thesis of this research is summarised as :

> **Semantic guidance can be used to improve the performance of a linear deduction system.**

The thesis has been verified through the development of the Semantically Guided Linear Deduction (SGLD) system. SGLD is a chain format linear deduction system with integrated semantic guidance. The host deduction system of SGLD is a new chain format linear deduction system called Guided Linear Deduction (GLD). SGLD's semantic guidance system uses a new form of interpretive structures, called designations, to store the semantic information that it uses.

1.4 Languages and Notation

This section defines the language and notation used in this thesis.

Definition 1.1 - 1st order languages
A *1st order language* consists of *variables*, *functors* and *predicate symbols* (constants are viewed as functors of arity 0). The *terms* of the language are built from the variables and functors, the *universe* is built from the functors, the *atoms* are built from the terms and the predicate symbols and the *base* is built from the universe and the predicate symbols. (See, for example, [Lloyd, 1984] for details.) *Literals* are atoms and their negations, and *clauses* are disjunctions of literals (not sets of literals). Variables in terms, atoms, literals and clauses are (implicitly) universally quantified. As the interpretive structure presented in this thesis treats universe and base elements in the same way, the union of these two sets is called, for convenience, the *unibase*.
Notation : • Variables are written with the first letter in uppercase. • Functors are written in lower case. • Predicate symbols are written in lower case. • Disjunction is represented by "**v**". • Negation is represented by "~".

Example
An example of a 1st order language, named L, is :

Variables_L	Anything beginning with uppercase alphabetic, e.g. `Person`.
Functors_L	{`homer`/0, `spouse_of`/1}
Predicates_L	{`heart_ok`/1, `lungs_ok`/1, `alive`/1, `person`/1}

The value following the /, after each functor and predicate symbol, is the symbol's arity. Examples of the various language elements are :

Terms_L	`homer`, `spouse_of(Person)`
Universe_L	`homer`, `spouse_of(homer)`
Atoms_L	`lungs_ok(homer)`, `alive(spouse_of(Person))`

$Base_L$ `lungs_ok(homer), alive(spouse_of(homer))`

$Literals_L$ `~lungs_ok(homer), alive(spouse_of(Person))`

$Clauses_L$ `~lungs_ok(homer)` **v** `alive(spouse_of(Person))`

$Unibase_L$ `homer, spouse_of(homer), lungs_ok(homer), alive(spouse_of(homer))`

The truth value semantics[3] of a 1st order language is typically specified via a Tarskian style semantics. Such a semantics consists of a domain D, whose elements are constants; a functor-mapping F, from D^n to D, for each functor of arity n; and a predicate-mapping P, from D^n to {TRUE, FALSE}, for each predicate symbol of arity n. (See, for example, [Lloyd, 1984] for details.) A 1st order language may be extended using the domain of such an interpretive structure, to form an *extended 1st order language*.

Definition 1.2 - Extended 1st order languages

An extended 1st order language, formed by extending a 1st order language by the domain of an interpretation, has : the variables of the original language; the union of the domain and the original functors, as functors; the predicate symbols of the original language. The *extended-terms*, *extended-universe*, *extended-atoms*, *extended-base* and *extended-unibase* are built in the usual way. Collectively they are called *extended-expressions*. The domain and the original universe are both subsets of the extended-universe.

Notation : • Domain elements are written in lower case.

Example

The extended 1st order language L+D, formed by extending L by the domain D = {`mr_s`, `mrs_s`, `person`}, is :

$Variables_{L+D}$ Anything beginning with uppercase alphabetic, e.g. `Person`.

$Functors_{L+D}$ {`homer`/0, `mr_s`/0, `mrs_s`/0, `person`/0, `spouse_of`/1}

$Predicates_{L+D}$ {`heart_ok`/1, `lungs_ok`/1, `alive`/1, `person`/1}

Examples of the various extended language elements include the corresponding elements in L, and also :

$Terms_{L+D}$ `mr_s`

$Universe_{L+D}$ `mr_s, spouse_of(mrs_s)`

$Atoms_{L+D}$ `lungs_ok(mr_s)`

$Base_{L+D}$ `lungs_ok(mr_s), alive(spouse_of(mrs_s))`

$Unibase_{L+D}$ `mr_s, spouse_of(mrs_s), lungs_ok(mr_s), alive(spouse_of(mrs_s))`

---

[3] Here the standard truth value semantics of a 1st order language is described. In chapter 4 this is generalised to a generic form, and discussed further.

Definition 1.3 - d-expressions

For a given domain, expressions of the form $r(d_1, ... ,d_n)$, where each d_i is a domain element, are called *d-expressions*. If r is a functor then the expression is a *d-function* and if r is a predicate symbol then the expression is a *d-predicate*.

Example

Examples of d-expressions are :

d-functions$_{L+D}$ `homer`, `spouse_of(mr_s)`, `spouse_of(person)`

d-predicates$_{L+D}$ `lungs_ok(mr_s)`, `person(mrs_s)`

Definition 1.4 - Substitution

Given a set S, an *S substitution* is a finite set of the form $\{X_1/d_1, ... ,X_n/d_n\}$, where each X_i is a distinct variable and each d_i is an element of S not containing X_i. If θ is an S substitution then $T\theta$ is an *S instance* of T obtained from T by simultaneously replacing each occurrence of each variable X_i in T by d_i. If $T\theta$ contains no variables then $T\theta$ is a *ground* S instance of T. (Term and universe substitutions are the standard substitutions of 1st order logic. Application of a domain substitution forms an extended-expression.)

Example

Examples of substitutions are :

A Terms$_L$ substitution, θ, is

`{Man/homer, Person/spouse_of(Father)}`

The Terms$_L$ instance, `alive(Person)`θ, is

`alive(spouse_of(Father))`

A ground Terms$_L$ instance of `alive(Person)` is

`alive(spouse_of(homer))`

A D substitution, σ, is

`{Man/mr_s, Person/mrs_s}`

The ground D instance, `alive(Person)`σ, is

`alive(mrs_s)`

Definition 1.5 - Sort-expressions

In defining sort value interpretations, predicate symbols of arity 1 that determine sort also appear as domain elements. Expressions whose principal symbols are also domain elements, have their names prefixed by *sort-*, e.g sort-literals, sort-d-predicates and sort-expressions.

Example

Examples of sort-expressions are :

Sort-atoms$_L$	`person(homer), person(Person)`
Sort-base$_L$	`person(homer)`
Sort-atoms$_{L+D}$	`person(mr_s)`
Sort-base$_{L+D}$	`person(mr_s), person(person)`

Definition 1.6 - Structures

The *structure* of a function is the double consisting of its functor and arity. The structure of an atom is the double consisting of its predicate symbol and arity. The structure of a literal is the triple consisting of its sign, predicate symbol and arity.

Example

Examples of structures are :

Functions$_L$	`homer` ⇒ `homer`/0,
	`spouse_of(Person)` ⇒ `spouse_of`/1
Atoms$_L$	`lungs_ok(homer)` ⇒ `lungs_ok`/1
Literals$_L$	`~lungs_ok(homer)` ⇒ `~lungs_ok`/1
	`alive(spouse_of(Person))` ⇒ `alive`/1

1.4.1 Input Sets

A problem is presented to a deduction system as a set of input clauses written in a 1st order language. In chain format linear deduction systems (introduced in section 1.4.2) clauses are represented in the chain format.

Definition 1.7 - The chain format

A *chain* is an ordered sequence of literals. Each literal in a chain is classified as either an *A-*, *B-* or *C-literal*. The disjunction of the B-literals in a chain makes up the clause that is represented by the chain. An uninterrupted sequence of B-literals in a chain is called a *cell*. Input clauses are used to form *input chains* which consist entirely of B-literals. Chains that contain a single B-literal is called *unit* chains. Centre clauses of linear deductions (see definition 1.9) are represented by *centre chains*. Centre chains may contain A-, B- and C-literals. Various items of information may be associated with the literals in a chain.

Notation : • A-literals are placed in rectangles. • B-literals stand free. • C-literals are placed in ellipses. Literals in a chain are simply separated by spaces.

Example

An example of a chain format input set, written in L and called S, is :

```
S =  {~heart_ok(P) lungs_ok(P),
      heart_ok(P) ~lungs_ok(P),
      heart_ok(P) lungs_ok(P) ~alive(spouse_of(P)),
     ~heart_ok(P) ~lungs_ok(P),
      alive(spouse_of(homer))}
```

An example of a centre chain is :

~heart_ok(P) [~lungs_ok(P)]^0 (~heart_ok(P))

~alive(spouse_of(P))

The superscript 0 on the A-literal is a piece of associated information.

There is a 1st order language implicit in every input set. The implicit language consists of the variables (with multiple distinct copies available), functors and predicate symbols that appear in the input set. The sets of variables, functors and predicate symbols in the implicit language are subsets of their counterparts in the 1st order language in use. The universe and base of the implicit language are the Herbrand universe and the Herbrand base of the input set.

Example

The 1st order language L[S] implicit in S is :

$\text{Variables}_{L[S]}$	Anything beginning with uppercase alphabetic, e.g., `P`.
$\text{Functors}_{L[S]}$	{`homer`/0, `spouse_of`/1}
$\text{Predicates}_{L[S]}$	{`heart_ok`/1, `lungs_ok`/1, `alive`/1}

1.4.2 Deductions

The basic building blocks of deductions are *deduction operations*. Deduction operations are divided into two categories; *inference operations*, e.g., resolution, factoring, paramodulation, and *bookkeeping operations*, e.g., reordering of literals, truncation. Unification in all inference operations includes an occurs check. It is always understood that any substitutions resulting from unification are applied to the appropriate expressions. Each time an input clause is used in a deduction operation, a new set of variables is substituted for those in the input clause, thus avoiding variable clashes.

The deduced clause of a deduction operation contains literals inherited from the parent clauses and new literals introduced in the deduction operation. Some literals of the parent clauses do not appear, in any form, in the deduced clause.

Definition 1.8 - Discarded literals

Any parent clause literals that are not inherited by the deduced clause of an deduction operation are called *discarded literals*.

Linear Deduction

A refinement of the basic resolution procedure, that is of specific interest in this research, is the linear format. In particular, GLD and SGLD are chain format linear deduction systems. Due to the importance of (chain format) linear deduction in this research, a brief review is provided here.

Definition 1.9 - Linear deduction

Given an *input set* of clauses and a clause C_1 chosen from the input set, a *linear deduction* of C_n from the input set, with *top clause* C_1, is a sequence of *centre clauses* $C_1, \ldots, C_n$. Each *deduced clause* C_{i+1}, $i = 1..n-1$, is deduced from the *centre clause* C_i and *side clauses*. The side clauses are chosen from the input set and $C_1, \ldots, C_{i-1}$. For any C_i, the centre clauses C_1 to C_{i-1} are the *ancestor clauses* of C_i. A deduction operation that uses an ancestor clause is called an *ancestor deduction operation*. A linear deduction of the empty clause is called a *linear refutation*. These terms are used equivalently for chain format linear deduction systems, with reference to chains rather than clauses.

All chain format linear deduction systems have a common core of deduction operations. There are two inference operations based on binary resolution and one bookkeeping operation. The inference operations are *extension* and *reduction*.

1. Extension resolves a B-literal in the rightmost cell of a centre chain against a B-literal in an input chain. The deduced chain is formed by (i) placing the resolved upon centre chain B-literal at the right-hand end of the centre chain and reclassifying it as an A-literal and (ii) adding the non-resolved upon input chain B-literals to the right of the new A-literal.

   Example

   An example of an extension operation is :

   `~heart_ok(P)` `~lungs_ok(P)` ^{0} `~alive(spouse_of(P))`

   `heart_ok(P)`

   • Extends with `~heart_ok(P) lungs_ok(P)` to produce :

   `~heart_ok(P)` `~lungs_ok(P)` ^{0} `~alive(spouse_of(P))`

   `heart_ok(P)` ^{0} `lungs_ok(P)`

2. Reduction unifies a B-literal in a centre chain with a complementary A-literal to its left. The deduced chain is formed by removing the B-literal from the centre chain. Reduction implements ancestor resolution, followed by a sequence of factoring

operations. The implemented ancestor resolution is restricted so that all the ancestor B-literals added to the centre chain are identical to B-literals still existing in the centre chain. These identical instances are automatically factored by the reduction. The reduction of a B-literal against the A-literal immediately to its left may also be viewed as factoring of the corresponding input chain.

Example

An example of a reduction operation is :

```
~heart_ok(P) [~lungs_ok(P)] 0 ~alive(spouse_of(P))
    [heart_ok(P)] 0 lungs_ok(P)
```

• Reduces to produce :

```
~heart_ok(P) [~lungs_ok(P)] 1 ~alive(spouse_of(P))
    [heart_ok(P)] 0
```

The bookkeeping operation is *truncation* (also called *contraction*).

3. *Truncation* removes an A- or C-literal from the right-hand end of a centre chain. In some deduction systems, the truncation of an A-literal may cause the insertion of another A- or C-literal.

Example

An example of a truncation operation is :

```
~heart_ok(P) [~lungs_ok(P)] 1 ~alive(spouse_of(P))
    [heart_ok(P)] 0
```

• Truncates to produce :

```
~heart_ok(P) [~lungs_ok(P)] 0 (~heart_ok(P))
    ~alive(spouse_of(P))
```

In GLD and SGLD, deductions are built from *deduction chunks*, each of which may contain multiple deduction operations.

Below is an example of a chain format linear refutation of the input set S given in section 1.4.1. The example uses a very simple form of chain format linear deduction, and therefore does not illustrate all aspects of chain format linear deduction. Rather it serves to confirm the fundamental ideas.

Example

An example of a chain format linear refutation of S is :

`~heart_ok(P) ~lungs_ok(P)`

- Extends with `heart_ok(P) lungs_ok(P) ~alive(spouse_of(P))` to produce :

`~heart_ok(P)` `~lungs_ok(P)` `~alive(spouse_of(P))`
`heart_ok(P)`

- Extends with `~heart_ok(P) lungs_ok(P)` to produce :

`~heart_ok(P)` `~lungs_ok(P)` `~alive(spouse_of(P))`
`heart_ok(P)` `lungs_ok(P)`

- Reduces to produce :

`~heart_ok(P)` `~lungs_ok(P)` `~alive(spouse_of(P))`
`heart_ok(P)`

- Truncates to produce :

`~heart_ok(P)` `~lungs_ok(P)` `~alive(spouse_of(P))`

- Extends with `alive(spouse_of(homer))` to produce :

`~heart_ok(homer)` `~lungs_ok(homer)`
`~alive(spouse_of(homer))`

- Truncates twice to produce :

`~heart_ok(homer)`

- Extends with `heart_ok(P) ~lungs_ok(P)` to produce :

`~heart_ok(homer)` `~lungs_ok(homer)`

- Extends with `heart_ok(P) lungs_ok(P) ~alive(spouse_of(P))` to produce :

`~heart_ok(homer)` `~lungs_ok(homer)`
`~alive(spouse_of(homer))` `heart_ok(homer)`

- Reduces to produce :

`~heart_ok(homer)` `~lungs_ok(homer)`
`~alive(spouse_of(homer))`

- Extends with `alive(spouse_of(homer))` to produce :

`~heart_ok(homer)` `~lungs_ok(homer)`
`~alive(spouse_of(homer))`

- Truncates thrice to complete the refutation.

Linear-input Deduction

Linear-input deduction is a refinement of linear deduction which does not permit any form of ancestor resolution, i.e., (chain format) linear deduction using only the extension and

truncation operations. Linear-input deduction is complete for input sets of Horn clauses, but is not complete for input sets that contain non-Horn clauses. Linear-input deduction is of interest in this thesis because truth value deletion in linear-input deduction underlies many of the semantic guidance systems developed in chapter 3.

Definition 1.10 - Linear-input deduction
Given an *input set* of clauses and a clause C_1 chosen from the input set, a *linear-input deduction* of C_n from the input set, with *top clause* C_1, is a sequence of *centre clauses* C_1, ... ,C_n. Each *deduced clause* C_{i+1}, i = 1..n-1, is deduced from the *centre clause* C_i and *side clauses*. The side clauses are chosen from the input set. For any C_i, the centre clauses C_1 to C_{i-1} are the *ancestor clauses* of C_i. A linear-input deduction of the empty clause is called a *linear-input refutation*.

Ringwood [1988] provides an interesting synopsis and references for the history of linear-input deduction systems.

1.5 Deduction Faithfulness

The restrictions of a guidance system are imposed when a deduction operation is performed in the host deduction system. The restrictions are expressed in terms of the clauses involved in the deduction thus far. Restrictions can be divided into three categories, in increasing order of the extent of their effect. (i) *Operation restrictions*, which must hold at each deduction operation when it is performed. The restrictions are allowed to become violated as the deduction progresses. (ii) *Independent deduction restrictions*, which must hold independently at each deduction operation in a completed deduction. (iii) *Simultaneous deduction restrictions*, which must hold simultaneously at each deduction operation in a completed deduction. Independent deduction restrictions whose satisfaction is established without instantiating any variables, e.g., admissibility restrictions, have the same effect as simultaneous deduction restrictions. Operation restrictions that once satisfied always remain satisfied, are equivalent to deduction restrictions.

There are two principal approaches to establishing the satisfaction of restrictions. The first approach is to examine the clauses involved directly. The second approach, called the *ground approach*, is to find a ground universe instance of the clauses involved, that satisfies the restrictions. The ground approach can sometimes be used to check restrictions expressed in terms of the direct approach, e.g., an atom is FALSE if a FALSE ground universe instance of that atom can be found. The ground approach is typically suitable for use in semantic guidance systems. For semantic restrictions, the ground approach also

alleviates the problem that "if the domain of the interpretation contains more than a few elements the computation required to fully evaluate a non-ground clause may be too time consuming." [Henschen, 1976, p. 816]. In the ground approach only one suitable ground instance need be found at each check point.

For deduction restrictions, establishing that a restriction is satisfied when a particular deduction operation is performed, does not establish that the restriction will be satisfied in the completed deduction. The instantiation of variables later in the deduction may cause the restriction to become violated. Thus, after any instantiation of variables when building a deduction, it is necessary to recheck deduction restrictions at every deduction operation that has been performed thus far. This repeated rechecking entails a large amount of effort which may be prohibitive, or at least of negative utility. An alternative approach is to use deduction restrictions as operation restrictions, i.e., without rechecking at previous deduction operations, and to supply supplementary mechanisms to detect violations caused by the instantiation of variables. With the use of supplementary mechanisms, the *operational* imposition of deduction restrictions can be formulated to have the same or very nearly the same effect as full imposition. The extent to which operational imposition achieves the effects of full imposition is measured in terms of *deduction faithfulness*. Operationally imposed deduction restrictions are deduction faithful if they ensure that completed deductions conform to the deduction restrictions. Deduction faithfulness is a generalisation of "ground faithfulness" [Sandford, 1980, p. 209].

Two methods have been used to make operationally imposed deduction restrictions deduction faithful. (i) The deduction restrictions are formulated so that they have retrospective effect, i.e., so that their operational imposition at one deduction operation also has the effect of imposing restrictions at previous deduction operations. This is achieved by examining expressions that are available at the current deduction operation. For example, the admissibility restrictions on A- and C-literals in the Graph Construction procedure [Shostak, 1976] impose restrictions on B-literals in earlier deduction operations. This retrospective detection of deduction restriction violations also has a converse. Deduction restrictions can be formulated so that they prospectively detect inevitable violations. Such restrictions are designed by analysing possible sequences of deduction operations from a given point in a deduction. (ii) Auxiliary data structures are maintained specifically to enforce deduction faithfulness, e.g., the False Substitution Lists used in Hierarchical Lock Resolution [Sandford, 1980]. The auxiliary data structures need keep only sufficient information to detect violations and do not need to store a complete history of the deduction.

A factor which affects the extent to which operationally imposed deduction restrictions are deduction faithful is the timing of their use. Delaying the imposition of such restrictions, until after the instantiation of variables, increases the level of deduction faithfulness. This is because violations caused by the instantiation will be detected. On the other hand, delaying the imposition of the restriction may delay the detection of a violation. The effort expended from the point where the violation arose to the point of detection, is wasted. When determining the timing of the operational imposition of deduction restrictions, careful examination of the restrictions in terms of the host deduction system is therefore necessary. In linear deduction systems the timing decision is somewhat simplified. The deduction restrictions in linear deduction systems are, almost exclusively, based on the nature of the centre clauses. By operationally imposing deduction restrictions between deduction operations, both a delayed check on the preceding operation and a preemptive check on the next operation are performed .

1.6. Contributions of this Thesis

Section 1.3. describes the objectives of this research. In fulfilling the objectives, the research described in this thesis has contributed to the area of automated deduction. The major contributions are as follows.

1. The chain format linear deduction system, GLD, has been developed. GLD improves upon existing chain format linear deduction systems in various aspects. The notable improvements are (i) the provision of an explicit entry point for the incorporation of search guidance systems, (ii) the use of coarse grain deduction steps, (iii) a combined lemma/C-literal mechanism and (iv) an extended suite of admissibility restrictions. GLD is described in sections 2.3 and 2.4.
2. Three original methods of analysing input sets, that partially predict the structure of chain format linear deductions, have been developed. These methods are collectively called linear-input subset analysis. Linear-input subset analysis is described in section 2.5.
3. The implementational issues associated with truth value semantic deletion in linear-input deduction have been clarified, thus broadening the field of applicability. This work is described in section 3.3.
4. Semantic guidance systems have been developed for linear deduction systems. These include (i) a truth value deletion system (this is especially significant, as truth value deletion has previously been considered incompatible with linear deduction), (ii) a truth value preference strategy and (iii) combinations of sort value deletion with (i) and (ii). The semantic guidance systems are described in sections 3.4, 3.5, 3.6 and 3.7.
5. Designations (the new interpretive structures) have been developed for storing semantic information. The domains of designations are limited to be finite. Designations improve

upon existing structures by incorporating a generalised form of property inheritance. An efficient way of extracting the semantic information, stored in designations, has been formalized. Designations are described in section 4.4 and the extraction procedure is given in section 4.5.

6. The semantically guided linear deduction system, SGLD, has been developed. The combination of features integrated in SGLD is new. In particular, the semantic guidance in SGLD is original. SGLD is described in sections 5.1 and 5.2.
7. Testing of SGLD (described in section 5.4) has highlighted its strengths and weaknesses, thus giving direction for future research. These directions are noted in chapter 6, as part of the thesis' conclusion.

1.7. Thesis Structure

The chapters of this thesis are divided to cover the modular development of SGLD. There are two streams of development. Chapter two covers the first stream, describing the development of GLD. Chapters three and four cover the second stream. Chapter three investigates semantic guidance and chapter four describes designations. Each of these chapters is, to a large extent, self contained and has only this introductory chapter as prerequisite reading. Each chapter contains its own literature survey of related work. The two streams join in chapter five with the description of SGLD, merging the work of the preceding three chapters. The thesis is concluded in chapter six. At the end of the thesis are the reference list and two appendices. The first appendix contains examples that are too bulky to be retained in the main text. The second appendix contains the statements of the problems used to test SGLD and descriptions of the designations used in testing SGLD's semantic guidance system. Each chapter of the thesis starts with a brief introduction. These introductions have been duplicated below to give an overview of the thesis.

Chapter One - Introduction and Technical Preliminaries

Guidance systems are central to the success of automated deduction systems. Semantic guidance systems are guidance systems that use semantic information to determine search direction for their host deduction systems. Relative to their potential for improving the performance of deduction systems, it seems that semantic guidance systems have been badly neglected. All guidance systems are necessarily formulated to be compatible with their host deduction systems. Beyond this constraint, it is desirable to abstract the issues of syntax, application and implementation, as far as possible. This chapter introduces these issues and presents the notation and terminology used in this thesis.

Chapter Two - Guided Linear Deduction

This chapter introduces a new linear deduction system, called Guided Linear Deduction (GLD). GLD has been developed as a linear deduction system in which semantic guidance can be used and tested. To fulfil its role successfully, GLD must be an effective deduction system in its own right. This ensures that any favourable effects of incorporating semantic guidance are not attributable to the inadequacy of GLD. GLD must also have an appropriate entry point through which semantic guidance can be incorporated. GLD has been designed after an examination of existing linear deduction systems. GLD improves upon existing systems.

Chapter Three - Semantic Guidance

This chapter investigates and describes ways of using semantic guidance in deduction systems, particularly in linear deduction systems. As a first step, the underlying structure of truth value (semantic) deletion in linear-input deduction systems has been investigated. Understanding this structure has facilitated the development of (i) effective implementations of truth value deletion for linear-input deduction systems, (ii) a truth value deletion system for linear deduction systems and (iii) a truth value guidance strategy that can be used in a wide range of deduction systems. Sort value (semantic) deletion has also been seen to be effective in guiding deduction systems. This observation has motivated a reformulation of sort value deletion so that it has the same format as truth value deletion. In turn, this reformulation has facilitated the development of combined sort and truth value guidance systems.

Chapter Four - Designations

This chapter describes a new interpretive structure suitable for storing and supplying the semantic information used by semantic guidance systems. The new structures are called designations. The difficulty of storing and supplying semantic information is one of the factors that has discouraged the use of semantic guidance systems. There is a need for an interpretive structure that is expressive, space efficient, effective in supplying semantic information and also user friendly. A common approach is to store the semantic information as semantic functions. Interpretation of ground expressions is then performed using recursive descent. Designations generalise this approach, inheriting its good properties and remedying some of its faults. The domains of designations are limited to be finite.

Chapter Five - Semantically Guided Linear Deduction

This chapter describes the Semantically Guided Linear Deduction system (SGLD). SGLD is a semantically guided implementation of GLD. The implementation, in Prolog,

combines GLD with a semantic guidance system. Designations are used to store the semantic information used. SGLD has some features that are not specified in GLD. These features improve the real time performance of the implemented system without changing the structure of the deductions or the search space. The performance of SGLD has been investigated.

Chapter Six - Conclusion

This chapter reviews the outcomes of this research. SGLD has combined GLD, a semantic guidance system and designations, to form a unique deduction system. The components of SGLD are individually of interest and their combination into SGLD has confirmed the thesis of this research. Areas worthy of further investigation have also been noted.

Chapter Two

Guided Linear Deduction

This chapter introduces a new linear deduction system, called Guided Linear Deduction (GLD). GLD has been developed as a linear deduction system in which semantic guidance can be used and tested. To fulfil its role successfully, GLD must be an effective deduction system in its own right. This ensures that any favourable effects of incorporating semantic guidance are not attributable to the inadequacy of GLD. GLD must also have an appropriate entry point through which semantic guidance can be incorporated. GLD has been designed after an examination of existing linear deduction systems. GLD improves upon existing systems.

This chapter contains :

1. A historical survey of linear deduction systems.
2. The design criteria for GLD.
3. The formal definition of GLD.
4. Discussion of the deduction operations, deduction chunks and search strategy in GLD.
5. The description of three methods of analysing input sets, that partially predict the structure of GLD deductions.
6. A brief description of how equality may be embedded into GLD.
7. Concluding comments.

2.1. Linear Deduction Systems

Background

The linear refinements of the basic resolution procedure combine a range of restrictions, resulting in a deduction format that has some fundamentally desirable properties :

- Linear refutations are based on a sequence of modus ponens and contradiction arguments. This is a simple and natural structure for a proof. The structure makes it possible to extract an 'answer' from a refutation, thus making linear deduction systems suitable for question-answer systems.

- Linear deduction systems have "relatively uncomplicated" [Kowalski & Kuehner, 1971, pg 230] search spaces, thus making it easy to impose search guidance.
- The chain format linear deduction systems use a stack style data structure, thus permitting efficient implementation.

These, and other lesser, considerations have focused attention onto linear deduction systems. Most of the development of linear deduction systems appears to have taken place in the years (approximately) 1968 to 1974. After 1974, there has been little reported development.

Although different in presentation, the connection graph proof methods [Bibel, 1987] have parallels in linear deduction systems. The tableau format [Letz, Schumann, Bayerl, & Bibel, 1992] in particular is very similar to Selective Linear Model deduction [Brown, 1974] and LUST-resolution [Minker & Zanon, 1982]. The connection graph methods are, however, distinct enough to place them beyond the scope of this discussion.

Ancient History

The earliest linear deduction systems were the $\tilde{R}_3$ refinement [Luckham, 1970], s-linear resolution [Loveland, 1970] and the strategy of preference of a 'new' conjunction [Zamov & Sharonov, 1969]. These three systems were devised independently by their respective authors. The $\tilde{R}_3$ refinement is the simplest of these early systems, and proves the completeness of the linear format using full resolution. The $\tilde{R}_3$ refinement is also referred to as ancestry filter form resolution [Nilsson, 1971]. As well as proving the completeness of the linear format, s-linear resolution (full resolution is used) describes an important restriction for linear deduction systems. That is, the resolvant of an ancestor resolution operation must, possibly after factoring, subsume an instance of the parent centre clause. This restriction introduced the idea that ancestor resolution need be performed only if all the non-resolved upon literals in the ancestor clause can factor against non-resolved upon literals in the parent centre clause. The strategy of preference of a 'new' conjunction parallels s-linear resolution, but is expressed for input sets in prenex disjunctive normal form. The strategy uses the equivalent of full resolution. The equivalent of s-linear resolution's subsumption requirement is introduced in terms of "absorption" [Zamov & Sharanov, 1969, p. 9]. Because the development of deduction systems has been almost exclusively for input sets in conjunctive normal form, this latter system has been largely ignored. As an early work on linear deduction it is, however, equally as noteworthy as s-linear resolution.

Each of the first three systems is compatible with the Set of Support (SoS) strategy [Wos, Robinson G.A., & Carson, 1965]. Although not mentioned explicitly in some

presentations, the SoS strategy can be used with each of the linear deduction systems described in this section.

After the three initial systems, two streams of development emerged. One stream developed refinements based on resolution with merging [Andrews, 1968], while the other developed the chain format systems. Two significant common features emerged in the two streams of development. (i) The incorporation of the subsumption restriction of s-linear resolution. (ii) A mechanism for selecting the ancestor literal to resolve upon in an ancestor resolution. GLD is a chain format linear deduction system, and hence greater emphasis will be placed on such systems.

2.1.1. Resolution with Merging

Resolution with merging [Andrews, 1968] (where full resolution is used) was reported prior to the introduction of linear deduction systems. Resolution with merging imposes a restriction on the non-resolved upon literals of a resolution operation. A resolvant is a merge if, after factoring, it contains a literal that is descended from both of the parent clauses. This literal is called a merge literal. Andrews showed that it is never necessary to resolve two non-merges, and that this restriction is compatible with the SoS strategy.

A connection between resolution with merging and the subsumption restriction of s-linear resolution was noted by Loveland [1970], and several researchers have developed linear deduction systems that take advantage of resolution with merging. The first developed was the merge fishtail restriction, informally presented by Raphael [1969] and later formalized by Yates, Raphael and Hart [1970]. The merge fishtail restriction demonstrates the compatibility of linear deduction (using full resolution), resolution with merging and the SoS strategy. It also shows that, in an ancestor resolution, it is only necessary to resolve against a merge literal of the ancestor clause.

Anderson and Bledsoe [1970] extended the work of Raphael, to include the subsumption restriction of s-linear resolution and also added a no tautologies restriction. The completeness of this system was also independently reported by Yates et al. [1970]. This system is significant within this stream of development. It defines a linear deduction system which incorporates the subsumption restriction and provides a mechanism for selecting the ancestor literal to resolve against in an ancestor resolution.

Reiter [1971] imposed ordering strategies on a merging/linear/SoS type deduction system. An unfortunate side effect of the addition of ordering is that it is necessary to omit the subsumption restriction on ancestor resolution operations. The merge tight ordered s-linear

deduction with subsumption rule (MTOSS) system [Loveland, 1978] overcomes this problem by allowing descendants of merge literals to be resolved upon in ancestor resolution operations. The MTOSS system appears to be the last system in this stream of development.

2.1.2. Chain Format Linear Deduction Systems

All the chain format linear deduction systems use the chain format for clauses and the three core deduction operations, described in section 1.4.2. The reduction operation of chain format systems incorporates the subsumption restriction of s-linear resolution and the selection of a literal in an ancestor resolution is made using A-literals. These features provide commonality with the systems based on resolution with merging.

Besides the common data structure and deduction operations, all the chain format systems also use *admissibility restrictions*. These are restrictions on the extent to which atoms in a centre chain may be identical. The general effects of admissibility restrictions are to prevent loops in deductions, to prevent the use of tautologies in deductions and to make the use of certain deduction operations compulsory in certain circumstances. Each individual system provides its own specific admissibility restrictions to obtain the desired effects. In some systems the admissibility restrictions are defined as operation restrictions, while in other systems the admissibility restrictions are deduction restrictions.

The first chain format system developed was the Model Elimination (ME) procedure [Loveland, 1969a]. The first presentation of the ME procedure [Loveland, 1968] did not resemble a linear deduction system, but subsequent presentations [Loveland, 1969a, 1969b, 1972] set the standard for all the chain format systems that followed. Interesting features in the ME procedure are listed below.

- The ME procedure's extension operation always extends against the rightmost B-literal of the centre chain, while the reduction operation may use any B-literal in the centre chain. (In [Loveland, 1972] this flexibility is removed and only the rightmost B-literal of a centre chain may be used in a reduction operation.)
- The contraction operation of the ME procedure includes a mechanism by which lemma chains are produced and added to the input set. The lemma mechanism is a significant feature of the ME procedure, but has often been found to be of low utility. The persistent nature of lemma chains often increases the size of the search space unacceptably. As the ME procedure is complete without the lemma mechanism, restrictions can be used to reduce this problem. This approach has been taken by Fleisig, Loveland, Smiley and Yarmush [1974], whose implementation of the lemma mechanism includes a simplified subsumption test.

- The admissibility restrictions used in the ME procedure are operation restrictions. They restrict which centre chains can be used in deduction operations.

Developed independently of the ME procedure, Linear resolution with Selection function (SL-resolution) [Kowalski & Kuehner, 1971] was the second major chain format system presented. Interesting features in SL-resolution are listed below.

- The major contribution of SL-resolution was the introduction of a selection function. The selection function selects a B-literal in the rightmost cell of the centre chain for use in an extension operation. The use of a selection function facilitates some search guidance. A selection function or a selection rule has been used in all post-SL-resolution chain format systems. (A selection function is distinct from a selection rule, in that a selection rule "may depend on the history of the derivation" [Ringwood, 1988, p. 6]. A selection function provides an independent selection mechanism.) Note that no selection function is used in SL-resolution's reduction operation.
- As well as the three core deduction operations, SL-resolution incorporates m-factoring [Kowalski, 1970]. Although the added operation makes shorter refutations possible, it also increases the size of the search space. The necessity of selecting a B-literal from the rightmost cell of the centre chain in extension operations also prevents the m-factoring from having maximum effect. As was noted by Plaisted [1982, p. 235], "... the failure to economise on repeated subgoals seems to be a serious problem with SL-resolution ...".
- The admissibility restrictions in SL-resolution are deduction restrictions. Of the chain format systems described, SL-resolution is the only one in which the admissibility restrictions reject all tautologous centre chains. This is possible due to the inclusion of m-factoring.

Ordered Linear (OL)-deduction [Chang & Lee, 1973] combines features of the ME procedure and SL-resolution. Interesting features in OL-deduction are listed below.

- As in the ME procedure, the rightmost B-literal of a centre chain is always used in OL-deduction extension and reduction operations. The input chains of OL-deduction are ordered, thereby implementing a selection rule for extension and reduction operations.
- From SL-resolution, OL-deduction has adopted factoring (general factoring, not m-factoring) and a no tautologies deduction restriction.
- No admissibility restrictions beyond the no tautologies restriction are specified for OL-deduction.

The Graph Construction (GC) procedure [Shostak, 1976] introduced the use of C-literals in chain format systems. The C-literal mechanism overcomes SL-resolution's "failure to economise on repeated subgoals" to a large extent. Interesting features in the GC procedure are listed below.

- C-literals are inserted into the deduced centre chain whenever an A-literal is truncated. B-literals may reduce against C-literals as well as A-literals. Reduction against a C-literal recalls a portion of the deduction for reuse and effects a retrospective form of factoring. There is an equivalence (discussed further in section 2.4.4) between the lemma mechanism of the ME procedure and the C-literal mechanism.
- As in SL-resolution, the admissibility restrictions of the GC procedure are deduction restrictions. In comparison with the admissibility restrictions of other chain format systems, those of the GC procedure are slow to achieve the desired effects.

Selective Linear Model (SLM) deduction [Brown, 1974] was the last of the chain format systems developed in the prolific period up to 1974. Interesting features in SLM are listed below.

- SLM chains have a tree structure. A-literals are internal to the trees and B-literals are leaves of the trees. The tree structure of centre chains is brought about by a new deduction operation called *spreading*. Spreading, under certain conditions, spreads B-literals in a rightmost cell of a centre chain (each branch has such a cell) onto new branches of the centre chain.
- Any B-literal in any rightmost cell of a centre chain may be selected for use in an extension operation, thus providing more flexibility in the system.
- SLM provides a partial C-literal mechanism.
- A notable feature of SLM is the use of semantic information. Semantic information is used to control the use of the spreading and reduction operations and to determine admissibility.
- The admissibility restrictions of SLM are deduction restrictions. They are less strict than those of other chain format systems.

Linear resolution with Unrestricted Selection based on Trees (LUST)-resolution [Minker & Zanon, 1982] is based on SL-resolution, but (apparently unknown to its authors) incorporates features found in SLM deduction. Interesting features in LUST-resolution are listed below.

- LUST-resolution, like SLM deduction, uses a tree structure for its chains.
- LUST-resolution always spreads the B-literals in the rightmost cell of a centre chain.
- General factoring is used.
- LUST-resolution has deduction admissibility restrictions similar to those of SL-resolution.

There have been many implementations of chain format systems, e.g., [Fleisig et al., 1974; Stickel, 1986b; Tarver, 1990], which have incorporated various combinations of features, including some beyond those mentioned above. Of interest is the Prolog Technology Theorem Prover (PTTP) [Stickel, 1986b], which made minimal changes to the ME procedure (upon which it is based), but instead focused on a highly optimised implementation. It has been noted that "Although PTTP is one of the fastest theorem provers in existence when evaluated by its inference rate ... its high inference rate can be overwhelmed by its exponential search space ..." [Stickel, 1990, p. 674]. The failure of 'brute force' to produce a completely successful linear deduction system is an indicator of the necessity for search guidance.

2.2. The Design of GLD

The basic structure and core of deduction operations of chain format linear deduction systems provide a suitable foundation upon which to build powerful deduction systems. Other non-core features, found in chain format systems developed to date, provide a rich selection that can be used in such systems. Beyond the core deduction operations, notable features of the chain format systems discussed above, are (i) the use of a selection function/rule in extension operations, (ii) the compulsory use of certain operations in certain circumstances (not explicitly mentioned above, but enforced via admissibility restrictions), (iii) the incorporation of a factoring operation, (iv) the addition of lemma chains to the input set, (v) the use of the C-literal mechanism, (vi) the incorporation of a spreading operation and (vii) the imposition of admissibility restrictions. Notable by the their absence from the systems discussed are (i) more extensive use of a selection function/rule, (ii) explicit methods for ordering alternative successor centre chains, (iii) coarse grain deduction steps, (iv) specification of an overall search strategy.

The use of a selection function/rule in extension operations is the dominant search guidance mechanism used in existing chain format deduction systems. OL-deduction is the only chain format system that extends selection to its reduction operation, in the form of a selection rule. More general search guidance mechanisms are needed. Firstly, a selection rule should be used in all inference operations. It is important that a selection rule rather than a selection function be used, as this permits more flexibility in the choice of B-literal. Secondly, alternative successor centre chains should be evaluated and ordered for use.

The compulsory use of certain deduction operations (with a specific input chain in some cases) plays an important role in pruning the search space of chain format deduction

systems. Further emphasis on detecting situations in which the use of a certain operation can be compulsory, would result in an improved deduction system.

The incorporation of a separate factoring operation in linear deduction systems has both advantages and disadvantages. Factoring makes shorter refutations possible, but also increases the size of the search space. The reduction operation, lemma mechanism and C-literal mechanism, each implement some factoring. Reduction of a B-literal against the A-literal immediately to its left, implements factoring of input chains. The C-literal mechanism and, to a lesser extent, the lemma mechanism implement a retrospective form of factoring. A separate factoring operation thus seems to be redundant.

In addition to their factoring effects, the lemma and C-literal mechanisms implement reuse of previously deduced information. This has evident advantages. The lemma mechanism is more powerful than the C-literal mechanism, but often increases the size of the search space unacceptably. A mechanism for reusing deduced information, but which avoids increasing the size of the search space, would be of significant benefit.

Empirical evidence indicates that the spreading operation used in SLM deduction and LUST-resolution has some benefits [Tabada & Sutcliffe, 1990]. A drawback of the spreading operation is that it destroys the stack like nature of the centre chains, thus complicating implementation. There is insufficient firm evidence available to make a statement about the desirability of incorporating the spreading operation.

The deduction operations used in existing chain format systems are very fine grained. Several authors, e.g., Wos [1988], Bledsoe & Hodges [1988], have noted the importance of taking deduction steps of an appropriate size. If the deduction steps are too small, an excessive amount of intermediate information may be generated and stored. On the other hand, taking too large deduction steps may by-pass a path to a refutation. Optimally, deduction steps that are as large as possible should be taken so long as completeness is maintained. Coarser grain operations, such as hyper-resolution [Robinson J.A., 1965b] and linked-UR-resolution [Wos, Verhoff, Smith, & McCune, 1984], have proved to be successful in other deduction systems. The use of coarse grain deduction steps would enhance a chain format linear deduction system.

Many of the deduction admissibility restrictions defined for the chain format systems would have retrospective effect if imposed operationally. There is thus some redundancy in the restrictions. An admissibility checking system that uses deduction faithful, operationally imposed, admissibility restrictions would be desirable. The restrictions should retrospectively and prospectively detect admissibility violations.

None of the chain format systems discussed explicitly specifies an overall search strategy. A chain format linear deduction system (like all deduction systems) must have a fair [Lloyd, 1984, p. 52] search strategy, i.e., one that does not ignore any part of the search space that may contain a solution. It is desirable that the search strategy be specified as part of the deduction system.

Based on the above comments, an improved chain format linear deduction system would have the following features :

- No deduction operations beyond the core three.
- Use of a selection rule wherever possible.
- A method of ordering alternative successor centre chains.
- Maximised detection of situations in which the use of a certain deduction operation is compulsory.
- Coarse grain deduction steps.
- A mechanism for reusing deduced information. The mechanism must not increase the size of the search space dramatically.
- Operationally imposed deduction restrictions, formulated to have retrospective and prospective effect.
- An appropriate overall search strategy.

GLD is a chain format linear deduction system, designed to satisfy these goals. GLD is based broadly on the GC procedure. Many of the GC procedure's features have been enhanced and new features have been added. The following features make GLD an improvement over existing chain format linear deduction systems :

- A selection rule can be used in GLD's extension and reduction operations, and alternative successor chains can be ordered for use. Both the selection rule and ordering are controlled by a heuristic function. The heuristic function thus supplies an explicit entry point for the incorporation of guidance systems. There are many possible heuristic functions that could be used, some of which are listed by Chang and Lee [1973, p. 154]. A semantically based heuristic function is presented in chapter 3.
- Coarse grain deduction steps, called deduction chunks, are formed.
- A combined lemma/C-literal mechanism has been developed. The mechanism limits the increase in the size of the search space.
- An extended suite of operationally imposed admissibility restrictions is specified.

In designing specific details of GLD features, four basic maxims were adopted, as follows.

1. GLD would be 'tuned' to find deductions that end with the empty chain, i.e., refutations.
2. Unit input chains would be given preferential treatment. This idea is not new and underlies several refinements to the resolution procedure, e.g., the unit preference

strategy [Wos et al., 1964], unit resolution [Henschen & Wos, 1974] and UR resolution [Overbeek et al., 1976]. In GLD, several features have been structured to exploit the advantages of using unit input chains. For all chains in GLD, this maxim was extrapolated to a "fewest-literals preference strategy". Such a generalisation was originally proposed by Slagle [1965], as cited in [Chang & Lee, 1973, p. 153] and is based on the idea that shorter chains are closer to a refutation than longer ones.

3. 'A stitch of pruning is worth nine of search.' Any overhead introduced by a mechanism that could prune the search space, would be considered justified. This maxim is well motivated by Wos [1988].
4. The various components of the system should integrate smoothly. Given alternative ways of incorporating a feature, the method that is least disruptive to the overall structure of the system would be chosen.

The formal definition of GLD is given in a dynamic style, in the manner of [Schumann, Letz, & Kurfess, 1990]. This is in contrast to the static style used in some other system definitions. Thus, as well as specifying the nature of a GLD deduction, the definition also specifies how such a deduction is to be obtained.

2.3. The Formal Definition of GLD

2.3.1. Input and centre chain representation

GLD chains may contain A-, B- and C-literals, as defined in section 1.4.1. Extra items of information are associated with A- and C- literals, as follows.

- A-literals have an integer *scope* value. For a given A-literal with scope value N, the N A-literals to the right of the given A-literal are *within the scope* of the given A-literal.
- C-literals have a list of (references to) *scope A-literals*. The scope A-literals of a C-literal are to the left of the C-literal.

The input chains to a GLD deduction are created by assigning any convenient ordering to the literals in each input clause and classifying the literals as B-literals. Lemma chains that are created in the course of a deduction may be added to the input set.

2.3.2. Subsumption

Subsumption is used by GLD's admissibility checking and reuse of deduced information features. The principal use is in the addition (or non-addition) of lemma chains to the input set. GLD's preference for shorter chains and considerations concerning linear-input subset analysis (see section 2.5), have led to a subsumption algorithm that requires that a

subsumed chain has at least as many literals as the subsuming chain. This is a common modification to subsumption, and is called θ-subsumption by Loveland [1968, p. 97]. Thus, a chain C subsumes a chain D if :

1. The number of B-literals in C is less than or equal to the number in D.
2. There exists a substitution θ such that each B-literal in Cθ is a B-literal in D.

2.3.3. The Structure of GLD Deductions[4]

The building of GLD deductions is driven by several parameters. They are :

1. An input set of clauses. GLD converts the input clauses to input chains.
2. Definition of a support set, from which the top chain is chosen.
3. A search bound for the first iteration of GLD's consecutively bounded search. (An *iteration* of the consecutively bounded search is a complete search within one bound.)
4. A heuristic function. The heuristic function returns a value indicating the perceived quality of a centre chain.
5. A search style, one of literal-selected, literal-ordered, cell-selected or cell-ordered.

For a given input set, support set, initial search bound, heuristic function and search style, a GLD deduction of C_n is a sequence of centre chains $C^*_1, C_1, \ldots, C^*_n, C_n$ built such that :

1. Each C^*_{i+1}, i=1..n-1, is deduced from C_i by one of extension, A-reduction or C-reduction.
2. Each C_{i+1}, i=0..n-1, is deduced from C^*_{i+1} by a (there is only one) maximal sequence of Unit subsumed extensions, Identical A-reductions, Identical C-reductions, A-truncations and C-truncations (descibed below).
3. Only the C_i are stored.
4. C^*_1 is chosen from the support set. The order in which support set elements are used as top chains is :
    1. The order in which they appear in the input set, for the literal-selected and cell-selected search styles.
    2. An order such that the alternative C_1s are used in order of worsening heuristic value, for the literal-ordered and cell-ordered search styles.
5. For each C_i, the selected B-literal from the rightmost cell of C_i is :
    1. The rightmost B-literal, for the literal-selected and literal-ordered search styles.
    2. The B-literal that leads to the set of C_{i+1}s with the worst heuristic value, for the cell-selected and cell-ordered search styles. The heuristic value of a set of chains is the best of its elements' heuristic values, or the worst possible heuristic value if the set is empty.

---

[4]Explanatory comments on aspects of this formal definition are to be found in section 2.4.

6. For each C_i, given the selected B-literal, the order in which the possible C_{i+1}s are considered is :
    1. A default order (see section 2.4.3), for the literal-selected and cell-selected search styles.
    2. Worsening order of their heuristic values, for the literal-ordered and cell-ordered search styles.
7. In every C^*_i, when it is deduced :
    1. No two non-B-literals have identical atoms.
    2. No A- or C-literal is to the left of an identical B-literal.
    3. No B-literal is complementarily[5] identical to another B-literal in the same cell, or to the A-literal immediately to the right of its cell.
    4. No A-literal is complementarily subsumed by a unit input chain, unless the A-literal is the rightmost literal in C^*_i.

    These are GLD's admissibility restrictions. They are operation restrictions.
8. No C_i has more A- and B-literals than the search bound. Any C_i created with an excess of A- and B-literals is rejected. If C_n cannot be deduced using a given search bound, then the deduction may be restarted from C_1, as follows. If any lemma chains have been added to the input set in the iteration, then the search is restarted from C_1 with the search bound increased by one. If no lemma chains have been added to the input set in the iteration and the search bound has been exceeded, then the search is restarted from C_1 with the search bound increased by the minimum amount by which it was exceeded.

$C_{deduced}$ is deduced from C_{centre} by extension against an input chain C_{input} if :

1. The selected B-literal in C_{centre} is complementarily unifiable with a B-literal in C_{input}.
2. C'_{centre} is C_{centre} with the selected B-literal moved to the rightmost end, C'_{input} is C_{input} with the used B-literal removed and $C_{deduced}$ is the juxtaposition of C'_{centre} and C'_{input}. The selected B-literal is reclassified as an A-literal, with scope 0.
3. If C'_{input} is a unit input chain then the extension is called a unit extension.

$C_{deduced}$ is deduced from C_{centre} by A-reduction if :

1. The selected B-literal in C_{centre} is complementarily unifiable with an A-literal to its left.
2. $C_{deduced}$ is C_{centre} with the selected B-literal removed. The scope of the A-literal is reset to the number of A-literals to its right.

---

[5]The term *complimentarily* is used to specify a literal of opposite sign. Thus, for example, a complementarily identical literal is a literal of opposite sign with an identical atom, and a complementarily unifiable literal is a literal of opposite sign whose atom unifies with the atom of the literal in question.

$C_{deduced}$ is deduced from C_{centre} by C-reduction if :

1. The selected B-literal in C_{centre} is complementarily unifiable with a C-literal to its left.
2. $C_{deduced}$ is C_{centre} with the selected B-literal removed. The scope of each scope A-literal (of the C-literal) is reset to the number of A-literals to its right.

$C_{deduced}$ is deduced from C_{centre} by Unit subsumed extension against a unit input chain C_{input} if :

1. The rightmost literal of C_{centre} is a B-literal.
2. There is a target B-literal in C_{centre} which is complementarily subsumed by C_{input}.
3. No identical A-reduction or identical C-reduction is possible against a B-literal to the right of the target B-literal.
4. $C_{deduced}$ is C_{centre} with the target B-literal removed.

$C_{deduced}$ is deduced from C_{centre} by Identical A-reduction if :

1. The rightmost literal of C_{centre} is a B-literal.
2. There is a target B-literal in C_{centre} which is complementarily identical to an A-literal to its left.
3. No unit subsumed extension, identical A-reduction or identical C-reduction against a B-literal to the right of the target B-literal is possible; unit subsumed extension against the target B-literal is not possible; identical A-reduction of the target B-literal against an A-literal to the left of this A-literal is not possible; identical C-reduction of the target B-literal against a C-literal to the left of this A-literal is not possible.
4. $C_{deduced}$ is C_{centre} with the target B-literal removed. The scope of the A-literal is reset to the greater of its current scope and the number of A-literals between itself and the position of the (removed) target B-literal.

$C_{deduced}$ is deduced from C_{centre} by Identical C-reduction if :

1. The rightmost literal of C_{centre} is a B-literal.
2. There is a target B-literal in C_{centre} which is complementarily identical to a C-literal to its left;
3. No unit subsumed extension, identical A-reduction or identical C-reduction against a B-literal to the right of the target B-literal is possible; unit subsumed extension against the target B-literal is not possible; identical C-reduction of the target B-literal against a C-literal to the left of this C-literal is not possible; identical A-reduction of the target B-literal against an A-literal to the left of this C-literal is not possible.
4. $C_{deduced}$ is C_{centre} with the target B-literal removed. The scope of each scope A-literal (of the C-literal) is reset to the greater of its current scope and the number of A-literals between itself and the position of the (removed) the target B-literal.

$C_{deduced}$ is deduced from C_{centre} by A-truncation if :

1. The rightmost literal of C_{centre} is an A-literal.
2. An A-literal is a lemma A-literal of the truncation if its scope is equal to the number of A-literals to its right. The scope A-literals of the truncation are the lemma A-literals other than the rightmost A-literal. The C-point of the truncation is immediately to the right of the rightmost scope A-literal, or at the leftmost end of C_{centre} if there are no scope A-literals. A lemma chain of B-literals is formed from the negations of the lemma A-literals.
3. C'_{centre} is C_{centre} with the rightmost A-literal removed and with the scope of each scope A-literal decremented by 1.
4. If the lemma chain is subsumed by an input chain then :
    1. $C_{deduced}$ is C'_{centre}.

    If the lemma chain is a unit chain that is not subsumed by an input chain then :
    1. All input chains that are subsumed by the lemma chain are removed from the input set.
    2. The lemma chain is added to the input set.
    3. $C_{deduced}$ is C'_{centre}.

    If the lemma chain is a non-unit chain that is not subsumed by an input chain and the lemma chain subsumes at least one input chain then :
    1. All input chains that are subsumed by the lemma chain are removed from the input set.
    2. The lemma chain is added to the input set.
    3. $C_{deduced}$ is C'_{centre}.

    If the lemma chain is a non-unit lemma chain that is not subsumed by an input chain and the lemma chain does not subsume any input chains then :
    1. $C_{deduced}$ is C'_{centre} with a C-literal inserted at the C-point. The C-literal is the complement of the removed A-literal, with scope A-literals as above.

$C_{deduced}$ is deduced from C_{centre} by C-truncation if :

1. The rightmost literal of C_{centre} is a C-literal.
2. $C_{deduced}$ is C_{centre} with the C-literal removed.

Below is a GLD refutation of the input set S given in section 1.4.1. The refutation should be contrasted with that given in section 1.4.2.

Example

An example of a GLD refutation of S is :

`~heart_ok(P) ~lungs_ok(P)`

- Extends with `heart_ok(P) lungs_ok(P) ~alive(spouse_of(P))` to produce :

`~heart_ok(P)` `[~lungs_ok(P)]` ^{0} `~alive(spouse_of(P))`
`heart_ok(P)`

- Extends with `~heart_ok(P) lungs_ok(P)` to produce :

`~heart_ok(P)` `[~lungs_ok(P)]` ^{0} `~alive(spouse_of(P))`
`[heart_ok(P)]` ^{0} `lungs_ok(P)`

- Reduces to produce :

`~heart_ok(P)` `[~lungs_ok(P)]` ^{1} `~alive(spouse_of(P))`
`[heart_ok(P)]` ^{0}

- Truncates to produce :

`~heart_ok(P)` `[~lungs_ok(P)]` ^{0} `~alive(spouse_of(P))`

  - The lemma `~heart_ok(P) lungs_ok(P)` is produced and subsumed. The C-point of the truncation is immediately to the right of `[~lungs_ok(P)]` . If a C-literal were to be inserted, it would be (`~heart_ok(P)`) at that location. It would have `[~lungs_ok(P)]` as its single scope A-literal.

- Extends with `alive(spouse_of(homer))` to produce :

`~heart_ok(homer)` `[~lungs_ok(homer)]` ^{0}
`[~alive(spouse_of(homer))]` ^{0}

- Truncates twice to produce :

`~heart_ok(homer)`

  - The lemma `alive(spouse_of(homer))` is produced and subsumed.
  - The lemma `lungs_ok(homer)` is added to the input set.

- Extends with `heart_ok(P) ~lungs_ok(P)` to produce :

`[~heart_ok(homer)]` ^{0} `~lungs_ok(homer)`

- Unit subsumed extends with `lungs_ok(homer)` to produce :

`[~heart_ok(homer)]` ^{0} `[~lungs_ok(homer)]` ^{0}

- Truncates twice to complete the refutation.

  - The lemma `lungs_ok(homer)` is produced and subsumed.
  - The lemma `heart_ok(homer)` is added to the input set.

2.4. The Deduction Operations and Search Strategies in GLD

2.4.1. The Deduction Operations

GLD's eight deduction operations are comprised of six inference operations and two bookkeeping operations. The inference operations are the extension and reduction operations and the bookkeeping operations are the truncation operations. Orthogonal to the operational divisions, the deduction operations may be split into two groups dependent on whether or not alternative successor centre chains need to be considered once the operation has been completed. If alternatives do not need to be considered the operation is called a *compulsory* operation. The following table summarises the divisions :

Operation	Mode	
	Compulsory	Non-compulsory
Extension		Extension
	Unit subsumed extension	
Reduction		A-reduction
		C-reduction
	Identical A-reduction	
	Identical C-reduction	
Truncation	A-truncation	
	C-truncation	

Table 2.1 - The GLD deduction operations

An important difference between the non-compulsory and compulsory inference operations is that the non-compulsory inference operations operate on a selected B-literal in the rightmost cell of a centre chain, while the compulsory inference operations may use any B-literal in a centre chain. The truncation operations can and must be used when the rightmost literal in the centre chain is an A- or C-literal.

The extension group of operations are, in combination, equivalent to the extension operations of other chain format systems. If a B-literal can be removed from a centre chain without any instantiation of variables in the centre chain, it is unnecessary to consider alternative ways of removing the B-literal. Thus unit subsumed extension is a compulsory operation. Stickel [1986b] used 'unit subsumed extensions' in the PTTP. Unit subsumed extension in GLD is an efficient operation, in that no A-literal is created. The creation of an A-literal would be redundant as it would immediately be A-truncated and the unit lemma chain created would necessarily be subsumed by the unit input chain used.

GLD's reduction operations combine features from the ME procedure's and the GC procedure's reduction operations. A significant feature in GLD's non-compulsory reduction operation is the use of a selection rule. This provides a search guidance point that is not available in most other chain format systems. As the identical reduction operations do not instantiate any variables in the centre chain, they, like unit subsumed extension, are compulsory operations. Making identical reduction a compulsory operation is a common feature of linear deduction systems. The lack of clarity in the literature over this issue has been discussed by Sutcliffe and Tabada [1991]. The reduction operations work in tandem with A-truncation to implement reuse of deduced information.

GLD's A-truncation operation combines features of the ME procedure's and the GC procedure's truncation operations, providing facilities to add lemma chains to the input set as well as to insert C-literals. This reuse of deduced information is discussed fully in section 2.4.4. The C-truncation operation is directly that of the GC procedure.

Although it would be possible to include a factoring operation in GLD, this has not been done. Both general factoring and factoring of identical B-literals were tested in GLD. The positive effects of using either of these operations were outweighed by a detrimental effect on the lemma/C-literal mechanism of GLD and an expanded search space. The absence of a separate factoring operation is compensated for by the combined effects of the lemma/C-literal mechanism and A-reduction.

2.4.2. Chunking

GLD is the first chain format system to explicitly build coarse grain deduction steps. In GLD multiple deduction operations are combined into indivisible deduction chunks. The philosophy underlying GLD's operation chunking is that no centre chain is stored while it contains a literal that can be removed by a compulsory operation. This approach does not destroy the deduction completeness of the system. Thus, after each non-compulsory operation, a maximal sequence of compulsory operations is performed before the resulting centre chain is stored. The intermediate chains deduced are discarded. The initial non-compulsory operation and the sequence of compulsory operations form a deduction chunk. The chunk is *based* on the non-compulsory operation. A side effect of chunking is that every stored centre chain necessarily has a cell at its right-hand end, and can therefore have a non-compulsory inference operation performed on it.

The building of the maximal sequence is arranged so as to avoid changing scope values when possible. If this is not possible then the scope values of A-literals which are as far to

the left as possible, are increased. To this end, unit subsumed extension is performed in preference to identical reduction, and reduction is performed against A- and C-literals which are as far to the left as possible (no distinction is made between A- and C-literals). This latter preference, also used in st-linear resolution [Shostak, 1976], has two effects. Firstly, shorter lemmas are created. Secondly, subsequently inserted C-literals exist longer and are more effective. To maximise the use of unit extension, any unit lemma chains created and added to the input set within the sequence, are created as soon as possible. This is achieved by examining the centre chain literals from right to left.

2.4.3. B-literal Selection and Successor Ordering

In each chunk of a GLD deduction, two choices have to be made. The first is to select a B-literal for the base operation and the second is to choose an order in which alternative successor centre chains are to be considered. GLD uses two methods for selecting a B-literal and two methods for ordering alternative successors. These, in combination, provide four possible search styles, as summarised in the following table.

Successor Order	Literal Selection	
	Rightmost literal	Most likely to fail
Default	Literal-selected	Cell-selected
Decreasing quality	Literal-ordered	Cell-ordered

Table 2.2 - Search styles

The determination of which B-literal is most likely to fail and the order of decreasing quality, are done in terms of the heuristic function supplied. The heuristic function provides an explicit entry point for search guidance. Such an entry point is absent in existing chain format systems. This feature is crucial in GLD, as it is via this entry point that semantic guidance is incorporated into GLD.

The names of the search styles are derived from how the B-literal is selected and how alternative successor chains are ordered. These issues are described below.

B-literal Selection

The literal-selected and literal-ordered search styles use a trivial B-literal selection method, as in the ME procedure, simply taking the rightmost B-literal. This provides no search guidance.

In the cell-selected and cell-ordered styles, the B-literal is selected from the rightmost cell. The selection aims to make that which is most likely to lead to failure. This selection criteria is motivated by noting that each B-literal in a cell must be used eventually. There is no point in selecting a B-literal that is easily dealt with, only to fail later on another. This approach has been motivated by various authors, e.g. Naish [1986], Plaisted [1990b]. To make the selection that is most likely to lead to failure, the set of successor centre chains is deduced for each B-literal in the rightmost cell. For each successor set, the heuristic value of each successor in the set is calculated. The best of the successors' values is assigned as the heuristic value of the set. If there are no successors then the set is assigned the worst possible heuristic value. The B-literal whose successor set has the worst value, is selected. This technique of looking ahead is better than making a selection based on the nature of the B-literals themselves.

Successor Ordering

The literal-selected and cell-selected search styles use a default ordering of alternative successor centre chains. The default ordering is guided by (i) the fewest-literals maxim and (ii) by avoiding changing scope values if possible, or if not possible increasing the scope values of A-literals which are as far to the left as possible (as in the maximal sequences of compulsory operations). Unit extension and reduction operations always deduce shorter centre chains and extension operations do not change scope values. Therefore preference is given first to unit extension based chunks, second to reduction based chunks and third to non-unit extension based chunks. Within reduction based chunks, preference is given to reductions against A- and C-literals which are as far to the left as possible (no distinction is made between A- and C-literals).

The literal-ordered and cell-ordered search styles use the alternative successor centre chains in order of worsening heuristic value.

The four search styles produce different search trees, due to their different methods of B-literal selection and of ordering alternative successor centre chains. The computational effort required to select a B-literal and order the alternative successors differs over the four styles. Effort is expended in two areas, firstly in deducing alternative successors and secondly in evaluating of the heuristic function for the alternative successors.

- The literal-selected search style incurs the least overhead, as no heuristic values are used and only one successor is deduced at a time. Correspondingly, no search guidance other than the default ordering of alternative successors is provided by the literal-selected search style.

- The literal-ordered search style deduces all successors for a trivially selected B-literal and calculates the heuristic value for each. The calculated values are used to provide search guidance (by ordering the successors). Each successor may be used if those with better heuristic values lead to failure. Thus the effort required to deduce all the successors may be justified.
- The cell-ordered search style deduces every successor for every B-literal in the rightmost cell of the centre chain and calculates a heuristic value for each. These heuristic values are used to select a B-literal and then to sort the successors for the selected B-literal. Each of the successors for the selected B-literal may be used, as in the literal-ordered search style. However, the successors for the non-selected B-literals are discarded.
- The cell-selected search style is a downgraded version of the cell-ordered search style, in that the successors for the selected B-literal are left unsorted.

It is evident that the literal-ordered and cell-ordered search styles use heuristic values most effectively. The literal-ordered search style is called a hill climbing search by Winston [1984, p. 93] and a modified depth-first method by Chang and Lee [1973, p. 151].

An issue closely related to ordering of alternative successor chains is the order in which support set elements are used as top chains. In GLD the elements of the support set may be viewed as alternative intermediate chains deduced by alternative base operations of initial deduction chunks. The order in which they are used as top chains is similar to the ordering of alternative successor centre chains. A default order (their order in the input set) is used for the literal-selected and cell-selected search styles. In the literal-ordered and cell-ordered search styles the elements are used in an order such that the centre chains deduced by the 'alternative chunks' are deduced in order of worsening heuristic value.

2.4.4. Reuse of Deduced Information

Three mechanisms for reusing deduced information have been developed in existing chain format linear deduction systems. They are the lemma mechanism in the ME procedure, the C-literal mechanism in the GC procedure, and the caching mechanism used by Astrachan and Stickel [1992]. Caching is appropriate only in purely linear-input deductions, and is thus not considered further here. The lemma mechanism adds lemma chains to the input set and these may be used in extension operations. The C-literal mechanism inserts C-literals into the centre chain and the C-literals may be reduced against. Using a lemma chain is equivalent to duplicating the sequence of deduction operations that lead to the creation of the lemma. The same is true for C-literals. Using a lemma or a C-literal produces a shorter deduction. Caching is a generalised variant of the lemma mechanism.

Lemmas and C-literals hold very similar information, as may be observed from an examination of the two mechanisms. In the ME procedure, lemma chains are formed from the negations of the A-literal being truncated and certain other A-literals to its left. The other A-literals are, in GLD parlance, the scope A-literals. An A-literal is a scope A-literal by virtue of its participation in a prior reduction against a B-literal which was to the right of the A-literal now being truncated. In the GC procedure such a prior reduction would have ensured that the C-point (of the A-literal now being truncated) is to the right of all such reduced against (scope) A-literals. Thus the C-point is immediately to the right of the rightmost such (scope) A-literal. The A-literals that contribute to a lemma chain in the ME procedure thus also determine the C-point in the GC procedure. Each C-literal inserted in a GC procedure deduction corresponds to a lemma chain created in an ME procedure deduction. C-reduction is equivalent to extension against the corresponding lemma chain followed by A-reduction of the remaining lemma chain B-literals against the scope A-literals.

The lemma mechanism has two distinct advantages over the C-literal mechanism. The first is that lemma chains remain in the input set even if the branch of the search which creates the lemma chain leads to failure. The lemma chain may be used in another branch of the search, or, in the environment of a consecutively bounded search, it may be used in the next iteration of the search. In contrast, a C-literal is available only until it is truncated from the centre chain. The second advantage is that each time a lemma chain is used in an extension, a fresh set of variables is created. Thus one use of a lemma chain does not affect the next use. When a C-literal is used its variables may be instantiated, thus making it unsuitable for further use. An associated effect is that after an extension against a lemma chain, the variables in the lemma chain B-literals are not necessarily unified with other variables in the centre chain. The effects of subsequently instantiating the lemma chain variables are thus less widely felt. Variables in a C-literal typically share with other variables in the centre chain and their instantiation has effects elsewhere in the centre chain.

The C-literal mechanism has one distinct and critical advantage over the lemma mechanism. The persistent nature of lemmas (which, as discussed above, is an advantage in some situations) typically leads to a debilitating increase in the size of the search space. The detrimental effect of this has been noted in various places, the principal problem being cited that "lemmas tend to be highly redundant - they are often subsumed by other lemmas and input chains" [Shostak, 1976, p. 63]. The C-literal mechanism does not suffer from this problem. A second advantage of the C-literal mechanism is that C-reduction combines

the multiple operations that would have to be done separately if the equivalent extension against a lemma chain and following A-reductions, were to be performed.

GLD introduces a combination of the lemma and C-literal mechanisms, retaining the best features of each. The GLD lemma/C-literal mechanism can add lemma chains to the input set and can also insert C-literals. Because of the evident advantages of the lemma mechanism over the C-literal mechanism, preference is given to the creation of lemmas. However, safeguards are provided against the proliferation of non-unit lemma chains. To implement the combined mechanism, A-literals maintain a scope value as in the ME procedure. The C-point of an A-truncation is determined from the scope values. Because a C-reduction implements several A-reductions, it is necessary to update scope values in C-reductions as well as in A-reductions. The A-literals whose scope values need updating in a C-reduction are those which determined the insertion point of the C-literal, i.e., the scope A-literals of the C-literal. In a C-reduction, each such scope A-literal is updated as if it had A-reduced against the B-literal involved.

GLD's lemma/C-literal mechanism gives preferential treatment to unit lemmas. If a unit lemma is not forward subsumed it is always added to the input set. The alternative of inserting a C-literal is never taken. A redundancy in the GC procedure, of inserting C-literals immediately after a unit extension (such C-literals are subsumed by the unit input chain used), is eliminated here. In GLD no C-literal is inserted and the lemma chain created will necessarily be subsumed by the unit input chain used in the extension. Although the unit lemma chain strategy permits the number of unit input chains to grow, this is in line with the 'preferential treatment of unit chains' maxim of GLD. As well as their use in GLD's unit orientated extension operations, unit lemma chains are also used by one of GLD's admissibility restrictions.

A non-unit lemma chain is added to the input set only if it is not forward subsumed and it backward subsumes at least one existing input chain. The latter restriction prevents the number of non-unit input chains from increasing. Further, because a subsuming chain has no more literals than the subsumed chain, the total number of B-literals in non-unit input chains never increases. This simplifies the set of input chains. The use of C-literals rather than adding 'new' non-unit lemma chains maintains the advantages of reusing deduced information, but does not have the deleterious effect of proliferating non-unit input chains. It is, in some circumstances, possible for an inserted C-literal to be redundant. This occurs if the corresponding lemma chain would be subsumed by an existing input chain. However, the C-literal is still inserted so as to take advantage of the efficiency of the C-reduction operation. Overall, this non-unit lemma chain strategy maintains the

advantages of adding lemma chains to the input set wherever it is possible to do so without increasing the size of the input set.

There are several possible variations of the lemma/C-literal mechanism, such as inserting a C-literal only if the corresponding lemma chain is not subsumed by any existing input chain. The chosen variation combines well with other aspects of GLD and empirical evidence suggests that the choice is a good one. The lemma/C-literal mechanism achieves its design aims of reusing deduced information, without dramatically increasing the size of the search space. The manipulation of scope values in GLD has also tightened up the rather loose specifications given in the ME procedure. Overall, the combined C-literal/lemma mechanism improves upon existing approaches to reusing deduced information in chain format linear deduction systems.

2.4.5. The Admissibility Restrictions

The admissibility restrictions in GLD are based on those of the GC procedure. The GC procedure specifies the deduction restriction that no two non-B-literals in any centre chain may have identical atoms. GLD imposes those and five new restrictions, in an operational manner. The new restrictions are :

1. No A-literal may be to the left of an identical B-literal. This is a prospective version of the existing GC restriction. An extension against the B-literal would create two identical A-literals and a reduction against the B-literal would create two complementarily identical A-literals or complementarily identical A- and C-literals. This restriction prevents loops in deductions and, in the case of an A-literal immediately to the left of the cell containing the B-literal, prevents the use of tautologous instances of input chains (this is never necessary). This restriction is also used in the ME procedure and the PTTP.
2. No C-literal may be to the left of an identical B-literal. As in item 1, this is a prospective version of the existing GC restriction. An extension against the B-literal would create identical A- and C-literals and a reduction against the B-literal would create two complementarily identical C-literals or complementarily identical A- and C-literals.
3. No B-literal may be in the same cell as a complementarily identical B-literal. This situation indicates that a tautologous instance of an input chain has been used. This restriction is also used in the ME procedure.
4. No B-literal may be in the cell immediately to the left of a complementarily identical A-literal. This also indicates that a tautologous instance of an input chain has been used.

5. No A-literal may be complementarily subsumed by a unit input chain, unless the A-literal is the rightmost literal of the centre chain (i.e., it was formed in a unit extension, in which case the unit input chain used would complementarily subsume it.). This restriction is also used in the PTTP. It maximises the use of unit subsumed extensions in the completed deduction.

GLD's operationally imposed restrictions detect many deduction restriction violations retrospectively and prospectively. It is worth highlighting those restrictions, inherited from the GC procedure, that have retrospective effect when imposed operationally. It is these retrospective effects that make it possible to impose the restrictions operationally, with a high level of deduction faithfulness. (The notation XY means an X-literal to the left of an identical Y-literal) :

1. AC retrospectively checks A~A.
2. CA retrospectively checks A~A.
3. C~A retrospectively checks AA.
4. CC retrospectively checks C~A.
5. C~C retrospectively checks CA.

If a chain is admissible, no compulsory operation will deduce a chain that is inadmissible. Thus GLD imposes the admissibility restrictions on the chains deduced by the base operations in deduction chunks. This prevents GLD from unnecessarily performing compulsory operations, while still ensuring that all stored centre chains are admissible.

2.4.6. The Consecutively Bounded Search

The overall search strategy of GLD is a modified consecutively bounded search. Stickel [1986b] gives the arguments for the use of a consecutively bounded depth first search in a linear deduction system. A standard consecutively bounded depth first search [Stickel & Tyson, 1985; Korf, 1985] truncates any long deduction sequence, whereas GLD improves on this by truncating only long deduction sequences that are not making progress towards a refutation. This is achieved by placing a bound on the number of centre chain A- and B-literals. Variations of this bounding scheme have been used in other deduction systems, e.g., the $C^{(2)}$ and $C^{(3)}$ variants of the ME procedure, which place a bound on the number of extensions.

As well as providing a complete search strategy, the consecutively bounded search is also used to counteract a side effect of GLD's lemma/C-literal mechanism. The removal and addition of chains to the input set within the lemma/C-literal mechanism can cause an iteration of the GLD search to fail, even if a refutation can be built within the search

bound. Such failure can occur when the input set is modified after successor centre chains have been deduced, in the literal-ordered, cell-selected and cell-ordered search styles.

Thus an iteration of GLD's search may fail either because centre chains that exceed the search bound are needed to build the desired deduction, or because the lemma/C-literal mechanism has interfered with the search. In both cases another iteration of the search is necessary. In the former case the search bound is increased by the minimum amount by which it was exceeded. This increment is also used in the PTTP. In the second case the search is reiterated with the search bound incremented by one. It is necessary to minimally increase the search bound in the second case to prevent GLD simply generating a sequence of lemmas which differ only in their literals' arguments. The strategy of permitting a deduction system to be incomplete within one iteration of a search, but complete over successive iterations, is a general one. If the advantage of imposing stringent restrictions outweighs the overhead of repeating the search, then the strategy is justified. Empirical evidence indicates that this is the case in GLD.

Although the lemma/C-literal mechanism can interfere with GLD's search, the lemma/C-literal mechanism also acquires added power within the consecutively bounded search. Because of their persistent nature, lemma chains added to the input set in one iteration of the search are carried over to, and may be used in, the next iteration.

2.5. Linear-Input Subset Analysis

The use of A-reduction in linear deduction systems makes them complete for sets of non-Horn clauses. There are, however, syntactically identifiable situations in which A- and C-reduction do not occur in GLD (and other linear deduction systems), i.e., situations in which linear-input deduction is performed. Three methods of analysing sets of input chains have been developed for detecting these situations. The first method focuses on Horn input chains while the second and third are successive generalisations of the first method. The detection of situations in which reduction does not occur is a new idea in linear deduction systems. It is useful for (and was largely motivated by) the imposition of truth value deletion (see section 3.4). Wakayama and Payne [1990] have also noted that ancestor resolution and factoring are not always necessary for obtaining a refutation when the input set is non-Horn. Their analysis is, however, restricted to entire input sets and the greater flexibility of linear-input subset analysis is desirable.

2.5.1 Horn Subsets

It has been noted that "... in many proofs, most of the input clauses are Horn clauses ..." [Plaisted, 1982, p. 231]. In linear refutations of some such input sets, once the positive B-literal of a Horn input chain has been extended against, no reductions are performed until that B-literal (in the guise of an A-literal) is truncated. Horn subset analysis detects such subdeductions in GLD deductions. Intuitively, Horn subset analysis detects those negative literals in the input set that can only resolve against Horn clauses in the input set, such that the negative literals in those Horn clauses also conform to this restriction. Then extension against such a negative literal can only lead to further extensions against Horn input clauses. This notion is now formalized.

The *Horn subset* , of an input set, contains atom structures that appear in the input set. To detect situations in which reduction does not occur in a GLD deduction from a negative top chain, the Horn subset of the input set is extracted. An atom's structure is in the Horn subset iff (i) it does not occur positively in a non-Horn input chain and (ii) for every Horn input chain in which the predicate structure occurs positively, every predicate structure in the chain is in the Horn subset.

Example

The Horn subset of {`~r~p~q, ~pq, p~q, pq, r~t~s, t~u, u, s`}, with `~r~p~q` as the top chain, is {`r/0, t/0, u/0, s/0`}.

The Horn subset divides the input chains into two groups, dependent on whether or not all literals in the chain have predicate structures that are in the Horn subset. Any predicate structures, literals, or input chains which contain only predicate structures that are in the Horn subset, are called *Horn subset objects*, e.g. `~r` is a Horn subset literal, and `r~t~s` is a Horn subset clause.

Horn subset analysis reveals three structural properties of GLD deductions from a negative top chain. Firstly, no A- or C-reductions against Horn subset literals are performed. Secondly, only the positive B-literal of a Horn subset input chain is ever resolved against in an extension operation. Finally, once a Horn subset B-literal has been selected, no reductions against literals rightwards from the selected B-literal are performed until that B-literal (in the guise of an A-literal) is truncated. These properties are now proved. (Concepts similar to those used here were informally introduced in [Sutcliffe, 1989].)

Lemma 2.3
In a GLD deduction from a negative top chain (i) no positive Horn subset A- or B-literal occurs in a centre chain, (ii) negative B-literals in Horn subset input chains are never resolved against in extension operations, and (iii) no A-reductions against Horn subset A- or B-literals are performed.

The proof of part (i) is by contradiction. If a positive Horn subset A- or B-literal occurs in a centre chain then the A-literal immediately to its left must be a Horn subset A-literal, as its complement originates from the same input chain as the first literal. Further, the Horn subset A-literal to the left must be positive, for otherwise the first literal occurs positively in a non-Horn input chain. Iteratively, all the A-literals to the left of a positive Horn subset A- or B-literal must be positive. However, the leftmost A-literal in the centre chain must be negative as the top chain is negative. Contradiction. Hence (i) no positive Horn subset A- or B-literal occurs in a centre chain, (ii) as there can be only negative Horn subset B-literals in a centre chain, negative B-literals in Horn subset input chains can never be resolved against in extension operations, and (iii) as complementary Horn subset A- and B-literals cannot occur in any centre chain, no A-reductions against such literals are performed. **QED**

Lemma 2.4
In a GLD deduction from a negative top chain (i) every A- and B-literal to the right of a Horn subset A-literal in a centre chain, is also a Horn subset literal, (ii) no A-reductions against A- and B-literals rightwards from a Horn subset A-literal are performed, and (iii) once a Horn subset B-literal has been selected, no A-reductions against literals rightwards from the selected B-literal are performed until that B-literal (in the guise of an A-literal) is truncated.

By lemma 2.3, once a Horn subset B-literal in a centre chain has been selected, it is necessarily extended against. From the definition of the Horn subset, the B-literals added to the centre chain in the extension are Horn subset B-literals. Therefore (i) iteratively, every A- and B-literal to the right of the original Horn subset B-literal (now an A-literal) is a Horn subset literal, (ii) by lemma 2.3, no reductions against A- and B-literals rightwards from a Horn subset A-literal are performed, and (iii) the structure of GLD deductions and (ii) ensure that no A-reductions against literals rightwards from the selected B-literal are performed until that B-literal (in the guise of an A-literal) is truncated. **QED**

Lemma 2.5
In a GLD deduction from a negative top chain (i) no Horn subset C-literal ever occurs in a centre chain, and (ii) no C-reductions against Horn subset literals are performed.

Whenever a (Horn subset) A-literal is created, it is not within the scope of any A-literal to its left. By lemma 2.4 no A-reduction can occur against a B-literal to the right of any Horn subset A-literal. Thus no Horn subset A-literal can ever come to be within the scope of another A-literal. The A-truncation of a Horn subset A-literal therefore leads to the creation of a unit lemma and not to the insertion of a C-literal. Hence (i) no Horn subset C-literal ever occurs in a centre chain, and (ii) no C-reductions against Horn subset literals are performed. **QED**

Theorem 2.6 - Horn Subset Analysis
In an GLD deduction from a negative top chain (i) no reductions against Horn subset literals are performed, and (ii) once a Horn subset B-literal has been selected, no reductions against literals rightwards from the selected B-literal are performed until that B-literal (in the guise of an A-literal) is truncated.

Directly from lemmas 2.3, 2.4 and 2.5. **QED**

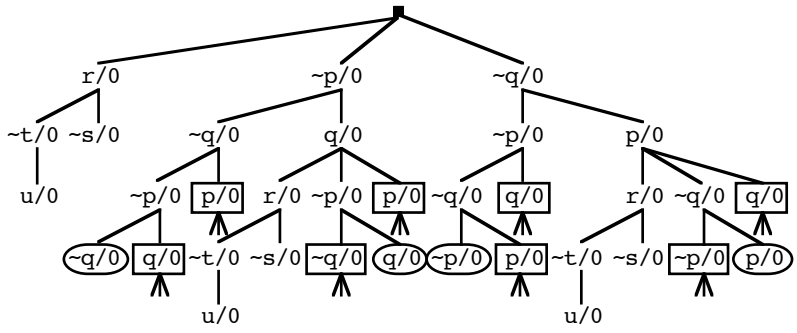
2.5.2 Linear-Input Subsets for literal Structures

Horn subset analysis focuses on Horn input chains. It does not provide adequate analysis for input chains which are non-Horn but are Horn in a renaming of the input set. Many results based on the polarity of literals can be generalised to be based on a division of the literal structures that appear in the input set, e.g., P_1 resolution [Robinson J.A., 1965b] generalises to P_P resolution [Meltzer, 1966], hyper-resolution [Robinson J.A., 1965b] generalises to AM-clashes [Slagle, 1967]. Similarly, Horn subset analysis generalises to results for non-Horn chains, in the form of Linear-Input Subset for literal Structures (LISS) analysis. The generalisation from Horn subsets to LISSs comes at the cost of a more complex analysis. Rather than a direct examination of the input set, LISS analysis requires examination of an abstraction of GLD's search tree. Intuitively, LISS analysis determines a superset of the possible sequences of A- and B-literals that can appear in a centre chain. It then extracts those literals that cannot be involved in a reduction operation, and also cannot appear to the right of a literal that can be involved in a reduction operation. Then extension against such a literal can only lead to further extension operations. This notion is now formalized.

For an input chain, the corresponding *chain structure set* contains the literal structures that occur in the input chain. To detect situations in which reduction does not occur in a deduction from a chosen top chain, the linear-input subset of the literal structures that occur in the input set is extracted. This is done by building an *extension tree* whose nodes are literal structures. The extension tree has a mythical root whose offspring are the elements of the chain structure set corresponding to the top chain. A literal structure in an extension tree has no offspring if it has itself as an ancestor in the extension tree unless, between itself and the ancestor, there exists a literal structure which does not have itself as an ancestor above the first ancestor. If a literal structure does have offspring then its offspring are those literal structures that (i) are in chain structure sets that contain a literal structure complementary to the parent literal structure, and (ii) are not the complementary literal structure. A literal structure is in the LISS iff for every occurrence in the extension tree (i) it is not complementary to an ancestor and (ii) all of its descendants are in the LISS.

Example

The first few levels of the LISS tree for {r~p~q, ~pq, p~q, pq, ~r~t~s, tu, ~u, s}, with r~p~q as the top chain, are :



Circled nodes are those that have no offspring due to the identical ancestor restriction. Boxes nodes are complementary to an ancestor. The lower levels of the tree reveal no new information. The LISS is thus {r/0, ~t/0, u/0, ~s/0}. No Horn subset exists for this top chain, as it is non-negative. With ~r~t~s as the top chain, the Horn subset is {s/0} and the LISS is {~t/0, u/0, ~s/0}. Note that this input set is simply a renaming of that given in the example in section 2.5.1, with r renamed to ~r and u renamed to ~u. The LISS obtained here is a corresponding renaming of the Horn subset in that example, given that negation signs implicitly prefix elements of a Horn subset. This illustrates the generalisation from Horn subset analysis to LISS analysis.

Any literal structures or literals which contain only literal structures that are in the LISS, are called *LISS objects*, e.g., r is a LISS literal.

LISS analysis reveals two structural properties of GLD deductions from a chosen top chain. Firstly, no A- or C-reductions against LISS B-literals are performed. Secondly, once a LISS B-literal has been selected, no reductions against literals rightwards from the selected B-literal are performed until that literal (in the guise of an A-literal) is truncated. These properties are now proved.

Lemma 2.7
In a GLD deduction from a chosen top chain, no A-reductions against LISS A- or B-literals are performed.

The root to tip sequence of literal structures in a branch of the extension tree corresponds to possible left to right sequences of A- and B-literal structures in centre chains of a deduction from the chosen top chain. Each node corresponds to a possible A-literal in a centre chain and literal structures further down the branch correspond to possible B-literals to the left of that A-literal in the centre chain. (Lemma 2.9 shows that no C-reductions against LISS literals are performed, but at this point nodes corresponding to LISS B-literals removed by C-reduction can simply be ignored.) Therefore (i) no LISS B-literal in a centre chain has a structure complementary to an A-literal to its left (LISS definition part (i)) and no A-reductions against LISS B-literals are performed, and (ii) no LISS A-literal has a B-literal with a complementary structure to its right (LISS definition part (ii)) and no A-reductions against LISS A-literals are performed. **QED**

Lemma 2.8
In a GLD deduction from a chosen top chain (i) every A- and B-literal to the right of a LISS A-literal in a centre chain is also a LISS literal, (ii) no A-reductions against A- or B-literals which are to the right of a LISS A-literal are performed, and (iii) once a LISS B-literal has been selected, no A-reductions against A- and B-literals rightwards from the selected B-literal are performed until that B-literal (in the guise of an A-literal) is truncated.

The proof is analogous to that of lemma 2.4.

Lemma 2.9
In a GLD deduction from a chosen top chain (i) no LISS C-literal ever occurs in a centre chain, and (ii) no C-reductions against LISS literals are performed.

The proof is analogous to that of lemma 2.5.

Theorem 2.10 - LISS Analysis

In a GLD deduction from a chosen top chain (i) no reductions against LISS literals are performed, and (ii) once a LISS B-literal has been selected, no reductions against literals rightwards from the selected B-literal are performed until that B-literal (in the guise of an A-literal) is truncated.

Directly from lemmas 2.7, 2.8 and 2.9. **QED**

LISS Extraction

LISS analysis has been employed in the implementation of SGLD, as described in chapter 5. Algorithm 2.11, below, has been used to extract the LISS from input sets. The algorithm implements a traversal of the LISS extension tree, for the given `TopChain`. The offspring of the root node are determined at line M4, and the offspring of other nodes are determined at line L2. For each literal structure in the tree (referred to as `Applicants` in the algorithm, because they "apply" to be in the LISS), a check is first made to determine if it has previously been added to the LISS, by virtue of its occurrence in another branch of the tree (line C2). If this is so, then it rechecked (lines C3-C4). This is necessary because the current occurrence may violate the conditions of membership, even if other occurrences do not. If the literal structure has previously been noted as not in the LISS, then this is acknowledged (lines C5-C6). The subtree rooted at such a node need not be re-examined. If a literal structure is complementary to an ancestor (line C7), then it is not in the LISS (line C9). However, its offspring are still created and checked, so as to determine their status (line C8). If a node meets the identical ancestor restriction (lines C10, R1-R6), then no offspring are created, and the repetition in the tree is noted (line C11). If none of the above conditions hold, then the literal structure is a potential LISS object. The offspring of the structure are created and checked (line C12). Provided that each offspring may be in the LISS (an offspring that is repeated may still be in the LISS - the determination is made at the identical ancestor node), then the current literal structure is in the LISS (line C14). If any offspring is not in the LISS then the current literal structure is not in the LISS, and this is noted (line C13).

Algorithm 2.11 - LISS Extraction

```
M1  Procedure Main(TopChain)
M2  LISS:={}
M3  NonLISS:={}
M4  Applicants:=The set of literal structures in TopChain
M5  CheckEachApplicant(Applicants,[])
```

```
E1  Function CheckEachApplicant(Applicants,HigherInTree)
E2  If there exists AnApplicant ∈ Applicants then
E3      Status = CheckAnApplicant(AnApplicant,HigherInTree)
E4      UpdateSets(AnApplicant,Status)
E5      Return({Status} ∪ CheckEachApplicant(
    Applicants - {AnApplicant},HigherInTree))
E6  Else Return({})

C1  Function CheckAnApplicant(TheApplicant,HigherInTree)
C2  If TheApplicant ∈ LISS then
C3      LISS:=LISS - {TheApplicant}
C4      Return(CheckAnApplicant(TheApplicant,HigherInTree))
C5  Else If TheApplicant ∈ NonLISS then
C6      Return(failed)
C7  Else If ~TheApplicant ∈ HigherInTree then
C8      CheckLowerApplicants([TheApplicant|HigherIntTree])
C9      Return(failed)
C10 Else If Repeated(TheApplicant,HigherInTree) then
C11     Return(repeated)
C12 Else If failed ∈ CheckLowerApplicants(
    [TheApplicant|HigherIntTree]) then
C13     Return(failed)
C14 Else Return(succeeded)

L1  Function CheckLowerApplicants([Parent|HigherInTree])
L2  Return(CheckEachApplicant({Applicant | Applicant ∈ a
    chain structure set containing ~Parent &
    Applicant ≠ ~Parent}, [Parent|HigherInTree]))

U1  Procedure UpdateSets(Applicant,Status)
U2  If Status = succeeded then
U3      LISS:=LISS ∪ {Applicant}
U4  If Status = failed then
U5      LISS:=LISS - {Applicant}
U6      NonLISS:=NonLISS ∪ {Applicant}

R1  Function Repeated(Applicant,[Parent|HigherInTree])
R2  If Applicant = Parent then
R3      Return(TRUE)
R4  Else If Parent ∈ HigherInTree then
R5      Return(Repeated(Applicant,HigherInTree))
R6  Else Return(FALSE)
```

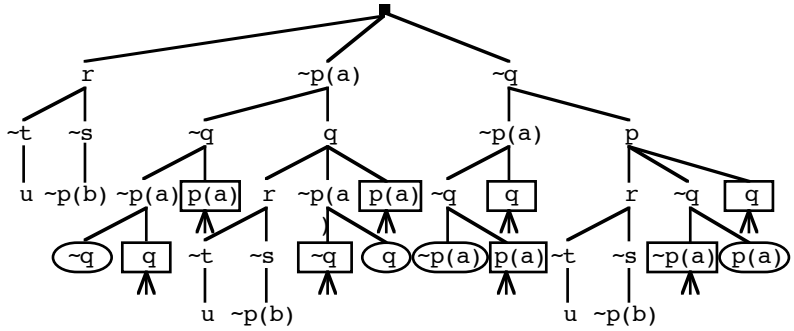
2.5.3 Linear-Input Subsets for Literals

In building the extension tree, LISS analysis makes the assumption that every pair of literals with complementary literal structures can unify. A more accurate analysis is possible by working directly with the literals in the input set. Linear-Input Subset for Literals (LISL) analysis does this. Intuitively, LISL (like LISS) analysis determines a superset of the possible sequences of A- and B-literals that can appear in a centre chain. However, LISL analysis does not assume that every pair of literals with complementary literal structures can unify. Rather, when building the extension tree, LISL analysis tests whether or not the parent node can unify with a literal in an input chain. This notion is now formalised.

To detect situations in which reduction does not occur in a deduction from a chosen top chain, the linear-input subset of the literals in the input set is extracted. This is done by building an extension tree whose nodes are literals from the input set. The method used is similar to that for LISS analysis. The extension tree has a mythical root whose offspring are the literals of the top chain. A literal in a LISL extension tree has no offspring if it has itself as an ancestor in the extension tree unless, between itself and the ancestor, there exists a literal which does not have itself as an ancestor above the first ancestor. If a literal does have offspring then its offspring are those literals that (i) are in chains that contain a literal complementarily unifiable with the parent literal, and (ii) are not the complementarily unifiable literal. A literal structure is in the LISL iff for every occurrence in the extension tree (i) it is not complementarily unifiable with an ancestor and (ii) all of its descendants are in the LISL. Note that although the extension tree uses unifiability, unification is never consummated.

Example

The first few levels of the LISL tree for {`r~p(a)~q, ~p(a)q, p(a)~q, p(a)q, ~r~t~s, tu, ~u, s~p(b), p(b)`}, with `r~p(a)~q` as the top chain, are :



Circled literal structures are leaves of the tree, as dictated by item (iii) in the definition of these trees. Boxes literal structures are complementarily unifiable with an ancestor. The lower levels of the tree produce no new information. The LISL is

thus {r[1], ~t[5], u[6], ~s[5], ~p(b)[8]} (where the superscripts indicate the chain number that the literal is in). The LISS is {~t/0, u/0}.

The results and theorem proofs for LISL analysis are analogous to those for LISS analysis. The results are simply stated here.

Lemma 2.12
In a GLD deduction from a chosen top chain no A-reductions against LISL A- or B-literals are performed.

Lemma 2.13
In a GLD deduction from a chosen top chain (i) every A- and B-literal to the right of a LISL A-literal in a centre chain is also a LISL literal, (ii) no A-reductions against A- or B-literals rightwards from a LISL A-literal are performed, and (iii) once a LISL B-literal has been selected, no A-reductions against A- and B-literals rightwards from the selected B-literal are performed until that B-literal (in the guise of an A-literal) is truncated.

Lemma 2.14
In a GLD deduction from a chosen top chain (i) no LISL C-literal ever occurs in a centre chain, and (ii) no C-reductions against LISL literals are performed.

Theorem 2.15 - LISL Analysis
In a GLD deduction from a chosen top chain (i) no A- or C-reductions against LISL literals are performed, and (ii) once a LISL B-literal has been selected, no reductions against literals rightwards from the selected B-literal are performed until that B-literal (in the guise of an A-literal) is truncated.

2.5.4 Discussion

The above results show that once a Horn subset/LISS/LISL B-literal (henceforth, Horn subset/LISS/LISL objects will be referred to generically as *linear-input objects*) has been selected, a linear deduction system goes into a linear-input configuration. The deduction system remains in linear-input configuration until that B-literal (in the guise of an A-literal) is truncated.

Definition 2.16 - Linear-input subdeductions
In a linear deduction, a subdeduction from the point when a linear-input B-literal is selected up to the point when it (in the guise of an A-literal) is truncated, is called a

linear-input subdeduction. The selected B-literal is called the *top literal* of the subdeduction.

In linear-input subdeductions the reduction operations can be explicitly ignored, so that no effort is spent trying to find A- and C-literals to reduce against. If Horn subset analysis is used then only the positive literals of Horn subset input chains need ever be considered when searching for suitable input chains in extension operations. A more significant benefit that may be derived from linear-input subset analysis is the completeness of a truth value deletion strategy in linear-input subdeductions. This strategy is described in section 3.4. In summary : In a linear-input subdeduction, every subchain consisting of the top literal and all literals to its right, must be FALSE in certain truth value interpretations.

The initial generation of the Horn subset/LISS/LISL is a simple iterative task, and may be done before deductions are built. In the course of a GLD deduction, existing input chains may be removed and lemma chains may be added to the input set. The effect of this on the linear-input subsets needs to be considered. The addition of unit lemma chains to the input set has no effect on the subsets. In the case of non-unit lemma chains, the subsumption requirement of the lemma/C-literal mechanism ensures that the set of literal structures in a non-unit lemma chain is a subset of that of a previously existing input chain. Thus the addition of new non-unit lemma chains to the input set neither reduces nor expands Horn subsets or LISSs. LISsL may, however, be reduced by the addition of non-unit lemma chains to the input set. This is because the added chains may cause the LISL extension tree to grow. In this situation the LISL must be updated immediately. The removal of chains from the input set could expand any of the subsets, as the removed chains may have caused the exclusion of potential linear-input objects. In this situation the subsets can be updated when it is convenient.

This presentation of linear-input subset analysis relates directly to GLD. These results are readily transferred to other linear deduction systems [Sutcliffe, 1992]. If the deduction system does not employ a lemma or C-literal mechanism, the transfer is trivial. If a lemma or C-literal mechanism is used, then the results of lemmas 2.5, 2.9 and 2.14 need to be re-established for the particular mechanism used.

2.6. Embedding Equality into GLD

As defined, GLD has no specific provisions for implementing deductions involving the axioms of equality. Many problems use include these axioms, so consideration has been given to embedding (the axioms of) equality into GLD's inference system. Although not

central to this research, the proposed embedding warrants mention, as the effects of semantic guidance on this form of embedding are of interest.

Probably the most well known method of embedding equality into a deduction system is the use of the paramodulation operation [Robinson G.A. & Wos, 1969]. Many other methods have been proposed, e.g., E-resolution [Morris, 1969], the Knuth-Bendix method [Knuth & Bendix, 1970], equational unification [Plotkin, 1972], narrowing [Slagle, 1974], the Resolution by Unification and Equality (RUE) and Negative Reflexive Function (NRF) inference operations [Digricoli, 1979], Prolog-with-Equality [Kornfield, 1983], lazy paramodulation [Gallier & Snyder, 1989] and relaxed paramodulation [Dougherty & Johann, 1990]. The method proposed for embedding equality into GLD is based on the RUE and NRF inference operations. The original RUE and NRF have been restructured to facilitate smooth integration with GLD.

The core of the embedding is a modified form of unification called *ED-unification* . Given a pair of expressions that need to be unified as part of a deduction operation, ED-unification may instantiate some variables (in a similar manner to standard unification) and also returns an *equality-demand chain*. An equality-demand chain contains negative equality B-literals called *equality-demand literals*. The arguments of equality-demand literals are pairs of non-identical, equivalently positioned, subexpressions of the expressions being ED-unified. The equality-demand literals are included in the deduced chain of the deduction operation.

Example

In extending the centre chain `q(Y) ~p(Y,c,f(e,g))` against the input chain `p(b,d,f(X,h)) ~r(X)`, the equality-demand chain created is `~equal(c,d) ~equal(f(e,g),f(X,h))` and the deduced chain is `q(b) ~p(b,c,f(e,g)) ~r(X) ~equal(c,d) ~equal(f(e,g), f(X,h))`

ED-unification also permits the arguments of equality literals to be used reversibly, so that ED-unification embeds the equality axioms of symmetry, transitivity and predicate substitutivity.

To complete the embedding of equality, a modified version of Digricoli's NRF operation is used. NRF is applied to a negative equality B-literal (possibly created as an equality-demand literal) in a centre chain and removes it from the centre chain. If either of the equality literal's arguments is a variable then the variable is instantiated to the other argument. Otherwise the two arguments must have the same principal symbol and the arguments' arguments are passed pairwise to ED-unification. Any equality-demand literals created are included in the deduced chain.

Example

An NRF operation applied to the last literal of

```
q(b),[~p(b,c,f(e,g))] ,~r(X),~equal(c,d),
                      ~equal(f(e,g),f(X,h))
```

produces

```
q(b),[~p(b,c,f(e,g))] ,~r(e),~equal(c,d),
                      [~equal(f(e,g),f(e,h))] ,~equal(g,h).
```

NRF embeds the equality axioms of reflexivity, and functional substitutivity. Within NRF, ED-unification is refined to implement standard unification whenever the functors of the terms being unified are Skolem functors. The motivation for this refinement is given in [McCune, 1990].

The extent to which this method of embedding equality instantiates variables and dismantles atoms is necessarily controlled. ED-unification instantiates variables and dismantles atoms to a minimal extent, with further instantiation and dismantling occurring in and controlled by the overall deduction structure. This contrasts with Digricoli's formulation, in which the RUE and NRF operations are controlled by a suite of restrictions.

ED-unification and NRF could be used to embed equality into GLD. ED-unification would be used in extension and reduction operations and NRF would be added as a non-compulsory operation. This embedding of equality is especially appropriate in GLD, as any equality-demand literals created will influence the search guidance. If this embedding is viewed as a demand driven implementation of paramodulation, the effects that the equality-demand literals have on the search provide a partial solution to problem 3 in [Wos, 1988]. The implicit existence of equality axiom input chains in such an embedding of equality would also affect subsumption and the generation of linear-input subsets.

2.7. Conclusion

GLD is a chain format linear deduction system that incorporates features of existing chain format systems. It also adds new features that were perceived to be missing from previous systems. The most important development associated with GLD is linear-input subset analysis. Linear-input subset analysis provides important information about the structure of GLD deductions. The information provided is used to gain some efficiency in GLD, but

more importantly to admit a semantic deletion system in (the linear-input) parts of GLD deductions.

Within GLD, there are some interesting features. The main points are listed below.

- GLD has explicit mechanisms for guiding its search, at both ordering and choice points. The ordering guidance is implemented via a selection rule on all non-compulsory inference operations. The choice guidance is implemented via ordering of alternative successor centre chains. Both mechanisms look ahead in the deduction to make the required decisions, using a heuristic function to evaluate alternative successor centre chains. The heuristic function provides an explicit entry point for the incorporation of guidance systems.
- GLD's reuse of deduced information, via the combined lemma/C-literal mechanism, improves on previous mechanisms. The lemma/C-literal mechanism interacts productively with GLD's unit orientated deduction operations, admissibility restrictions and overall search strategy.
- GLD is the first linear deduction system that explicitly exploits deduction chunking. The deduction chunking is possible due to five deduction operations for which alternative successor centre chains need not be considered, once the operation is completed. These compulsory operations are performed within deduction chunks. The deduction chunks are of maximal size and form coarse grain deduction steps. Choice points arise only between deduction chunks.
- GLD's operationally imposed admissibility restrictions have a high level of deduction faithfulness, due their retrospective and prospective effects.

The amalgamation of the features in GLD was guided by sound design maxims. Deciding on and adhering to the design maxims facilitated the development of a coherent system. As a result the various features of GLD complement, rather than conflict with, each other.

The definition of GLD has been given in a dynamic style, thus defining not only the nature of GLD deductions, but also the manner in which they are built. This approach is arguably more appropriate than the static approach used in some previous systems, as it permits GLD deduction systems to be implemented in a reproducible fashion. This kind of rigour in artificial intelligence research, automated deduction in particular, has been argued for by Pollack in her Computers and Thought award lecture [1991, pp. 22-23].

The design criteria of GLD have been successfully fulfilled. The dominant contributions to the research area of chain format linear deduction systems are linear-input subset analysis, the search guidance facilities and the lemma/C-literal mechanism. In the context of this research, linear-input subset analysis and the search guidance facilities are of paramount

importance, in that semantic information can now be used to directly guide GLD's search. The semantic guidance systems are developed in the next chapter.

Chapter Three

Semantic Guidance

This chapter investigates and describes ways of using semantic guidance in deduction systems, particularly in linear deduction systems. As a first step, the underlying structure of truth value (semantic) deletion in linear-input deduction systems has been investigated. Understanding this structure has facilitated the development of (i) effective implementations of truth value deletion for linear-input deduction systems, (ii) a truth value deletion system for linear deduction systems and (iii) a truth value guidance strategy that can be used in a wide range of deduction systems. Sort value (semantic) deletion has also been seen to be effective in guiding deduction systems. This observation has motivated a reformulation of sort value deletion so that it has the same format as truth value deletion. In turn, this reformulation has facilitated the development of combined sort and truth value guidance systems.

This chapter contains :

1. A generic description of the semantic information required for semantic guidance.
2. A survey of semantic guidance in automated deduction systems.
3. An exposé of truth value deletion in linear-input deduction systems.
4. The description of a truth value deletion system for linear deduction systems.
5. The description of a broadly applicable truth value guidance strategy.
6. A useful reformulation of sort value deletion.
7. The description of combined semantic guidance systems.
8 A brief investigation of the relationship between semantic guidance and theory resolution.
9. Concluding comments.

3.1. Basic Semantic Information

The semantic guidance systems described in this chapter make use of truth value and sort value information. There are many ways that such information can be stored (see chapter 4), but for present purposes it is necessary to define only the nature of the information. It is

noteworthy that the information is described here in a generic form and that specific types of semantic information (e.g., truth value information) are presented as specialisations of the generic form. It is for this reason that words such as "semantic guidance" and "interpretation" have been used in a generic manner.

Definition 3.1 - Basic semantic information
An interpretation of a 1st order language is a system that supplies :

- A *domain*, whose elements are constants.
- A set of truth values, whose elements are constants.
- A *universe-relation* from the universe to the domain[6].
- A *base-relation* from the base to the set of truth values.

If a unibase element is related to a certain domain element/truth value, the unibase element is *interpreted* as that domain element/truth value. The domain element/truth value is (one of) the unibase element's *interpretation value(s)*. Interpretations are categorised according to the type of semantic information supplied. Membership of a category is indicated by using the type of semantic information as a prefix; e.g., truth value interpretation.

Because the universe- and base-relations are relations (as opposed to mappings) it is possible that some unibase elements may not be interpretable. If a specific interpretation requires that all universe and/or base elements be interpretable, this must be specified explicitly for that interpretation.

A *truth value interpretation* is an interpretation in which the set of truth values is {TRUE, FALSE} and both relations are total functions. Truth value interpretation is the standard way of interpreting 1st order languages. Three simple truth value interpretations are :

- The positive interpretation, which maps all base elements to TRUE.
- The negative interpretation, which maps all base elements to FALSE.
- A predicate partition, which divides the predicate symbols of the 1st order language into two partitions P_1 and P_2. Base elements whose predicate symbols are in P_1, are mapped to TRUE. Base elements whose predicate symbols are in P_2, are mapped to FALSE.

A *sort value interpretation* is an interpretation where the domain contains the sorts required and the set of truth values is {TRUE, FALSE, UNKNOWN_TRUTH_VALUE}.

[6]For the semantic guidance systems described in this chapter, the domain and the universe-relation are surplus to requirement. However, in chapter 4 these components are used.

The relations may be partial, but the base-relation must be a partial function. A unibase element can relate to a domain element/truth value only if all its arguments relate to domain elements (Cohn [1987, p. 129] calls this restriction "strictness".) Sort-base elements, if they do map to any truth value, are restricted to map to one of TRUE or FALSE. A universe element that is related to a given domain element is said to be *of the sort* of the domain element. Sort value interpretations provide the notion of *sort legality*, which requires expressions to be of certain sorts (see section 3.2.3).

3.2. Semantically Guided Deduction Systems

Definition 3.2 - Semantic guidance and Semantic deletion
Semantic guidance is the use of semantic information to guide the search of a deduction system. *Semantic deletion* is the form of semantic guidance that guides the search by preventing certain chains from existing or being used. Semantic guidance and deletion are categorised according to the type of semantic information used. Membership of a category is indicated by using the type of semantic information as a prefix; e.g., truth value guidance and sort value deletion.

Background
The importance of taking advantage of semantic information in deduction systems has been expounded in chapter 1. The use of semantic information has the potential to improve the performance of deduction systems, as has been illustrated by existing deduction systems that use semantic information. The earliest work done in this area [Gelerneter, 1963; Gelerneter, Hansen, & Loveland, 1963] used semantic information to guide deductions in the domain of elementary geometry. Since then several (but as a proportion of deduction systems developed, relatively few) deduction systems that employ semantic guidance have been developed. There are three main approaches to using semantic information in deduction systems, as follows. (i) Performing reasoning in the problem domain and then reasoning analogously with the deduction system. (ii) Using semantic information directly within deduction operations. (iii) Using semantic information to direct and prune the search of the deduction system, i.e., semantic guidance. The linear deduction system described in this thesis is semantically guided and this background information will concentrate on that area. However, it is instructive to examine briefly the other two approaches.

- Reasoning analogously to reasoning in the problem domain is an under utilised technique in deduction systems. Only a few deduction systems (e.g., [Plaisted, 1981, 1984]) have used it. The principle is to transform the input set by replacing terms in the input set by their interpretation values, and to then build

deductions using the transformed input set. Deductions built using the transformed input set are then used to guide the building of analogous deductions using the original input set. If semantic guidance can be used in the deduction system, then it may often be used while building the deduction using the transformed input set.

- The use of semantic information directly within the deduction operations of a deduction system is a highly restrictive but effective way of solving problems in the associated problem domain. Deduction systems that utilise this approach are necessarily restricted in application to the associated problem domain. There is also some evidence that this kind of direct use of semantic information is unnecessary. See, for example, [Ginsberg & Geddis, 1991] for further discussion of this issue. Examples of problem domains that have been tackled using this approach are topology [Ballantyne, 1973], plane geometry [Nevins, 1975], set theory [Pastre, 1978; Ballantyne & Bledsoe 1982] and Boolean algebra [Winker, 1982]. As well as being of use in their own right, such systems could also find use within a more general deduction system, being used to implement and manipulate the interpretations required for semantic guidance in the general system.

An issue closely associated with the use of semantic information in deduction systems is the automatic generation of the required interpretations. Some work has been done in this area, as follows.

- The process of generating truth value interpretations to be used in Hierarchical Deduction [Wang, 1985] can be, at least in part, automated.
- The EGS system [Kim, 1986] automatically generates truth value interpretations for problems expressed in Boyer-Moore theory.
- Model generation systems, such as SATCHMO [Manthey & Bry, 1988] may be useful for this task.
- Some deduction systems that use semantic information directly in their deduction operations, e.g., [Winker, 1982; Ballantyne & Bledsoe, 1982], do so by generating counter examples for the input set, i.e., they generate truth value interpretations.
- The automatic augmentation of existing sort interpretations has been addressed by Schmidt-Schauss [1988]. Irani and Shin [1985] have done related work.
- The automatic generation of type information for Prolog style clauses has been addressed by, e.g., Mycroft and O'Keefe [1984] and Fruhwirth [1989].

The automatic generation of interpretations will inevitably become more important when techniques of using the interpretations become sufficiently developed. The topic is, however, sufficiently divorced from the focus of this research for it to be placed beyond the scope of this discussion.

Attention is now turned to semantic guidance in deduction systems. By far the most common form of semantic guidance is truth value deletion of parent chains in deduction

operations. Truth value deletion has been developed in a variety of formats. One format, which is of particular interest in this research, is truth value deletion in back chaining deduction systems. Other forms of truth value guidance, less common than truth value deletion, are : suggesting universe instances of parent chains to be used in a deduction operation; helping to choose a deduction operation to perform; and guiding a transformation of the input set. Semantic guidance using sort value information is limited to sort value deletion.

3.2.1. Truth Value Deletion

Truth value deletion requires that one or more parent clauses[7] of each deduction operation be interpreted as FALSE (or at least, not necessarily TRUE) in some given truth value interpretation. Many refinements of the basic resolution procedure which appear to be based on syntactic considerations, and at the time of development were not regarded as being semantically guided, are in fact guided by truth value deletion. The ground approach is appropriate for imposing truth value deletion (because establishing that a clause has a FALSE ground universe instance also establishes that the clause itself is FALSE).

The first exploitation of truth value deletion was the Set of Support (SoS) strategy [Wos et al., 1965]. The SoS strategy can be used in many deduction systems. (The compatibility of SoS with linear deduction systems was noted in chapter 2.) In the SoS strategy a subset of the input set, called the *support set*, is chosen so that the difference between the input set and the support set is a satisfiable set. This SoS strategy requires that at least one parent clause of each deduction operation is in the support set, i.e., ensuring that at least one parent is not necessarily interpreted as TRUE in models of the difference set. Deduced clauses are added to the support set. Some possible support sets are : the set of all positive clauses - the model of the difference set is the negative interpretation; the set of all negative clauses - the model is the positive interpretation; and the set of clauses deduced from the negation of the theorem to be proved - this choice assumes that there exists a model of the difference set.

The SoS strategy shows no preference amongst deduced clauses as parent clauses, thus it rapidly loses its effectiveness as deductions proceed. However, as the addition of a

[7] As GLD and SGLD use the chain format for their input sets, chain format terminology has been used in the sections of this chapter that describe the development of semantic guidance systems for linear deduction systems. However, the descriptions of existing semantic guidance systems have been expressed in terms consistent with their original presentations (typically clausal). With a few obvious exceptions, the semantic guidance systems developed in terms of chains are equally applicable to clausal systems, and vice versa.

deduced clause to an input set does not change the satisfiability of the set, the SoS strategy may be reapplied after each deduction operation. Such repeated use of the SoS strategy gives rise to several refinements of the basic resolution procedure, each of which employs truth value deletion. All of these refinements are variants of model resolution [Luckham, 1968, 1970]. Model resolution requires that at least one parent of each resolution operation is interpreted as FALSE, in a given truth value interpretation. Two simple forms of model resolution are P_1 [Robinson, 1965b] and N_1 resolution, which use the negative and positive interpretations respectively. P_p resolution [Meltzer, 1966] is a generalisation of P_1 and N_1 resolution, being based on a predicate partition.

Closely related to P_1 resolution is hyper-resolution [Robinson, 1965b]. Hyper-resolution is an efficient implementation of P_1 resolution. It requires that the electron parent clauses and the hyper-resolvant are positive clauses, and that the nucleus parent clause contains at least one negative literal. These restrictions ensure that the electron parent clauses and the hyper-resolvant are interpreted as FALSE, and that the nucleus parent clause has a ground universe instance that is interpreted as TRUE, in the negative interpretation. Similarly, negative hyper-resolution, AM-clash resolution [Slagle, 1967] and semantic clash resolution [Slagle, 1967] are efficient implementations of N_1, P_p and model resolution, based on the positive interpretation, a predicate partition, and some given interpretation, respectively. (Semantic clash resolution has also been called O_M-resolution [Loveland, 1978] and PI-resolution [Chang & Lee, 1973]. An extension incorporating subsumption is LI-r_c resolution [Slagle, 1972].)

AM- and semantic clash resolution, as well as being more general than the hyper-resolutions, use an ordering strategy in their deduction operations. A more restrictive ordering strategy is imposed by lock resolution [Boyer, 1971]. Variants of lock resolution that impose truth value deletion have been developed. The first of these, O_{IM}-resolution [Loveland, 1978], combines semantic clash resolution with a variant of lock resolution. O_{IM}-resolution places lock numbers only on those literals that are interpreted as FALSE in a given truth value interpretation. Similar to O_{IM}-resolution is Lock Resolution ∩ The Model Strategy (LR∩TMS) [Sandford, 1980], which combines lock resolution with model resolution. LR∩TMS places lock numbers on all literals in a restricted fashion, and is complete only for ground input sets. The completeness proof for LR∩TMS shows that, at the ground level, a variant of lock resolution is in fact a refinement of model resolution. LR∩TMS paves the way to Hereditary Lock Resolution (HLR) [Sandford, 1980] which combines model resolution with an extended form of lock resolution, in which literals are assigned a TRUE and a FALSE lock number.

On a distinct development path from the systems described above are Semantic Resolution for Horn Sets [Henschen, 1976] and Semantic Paramodulation for Horn Sets [McCune & Henschen, 1983]. Semantic Resolution for Horn Sets is a refinement of model resolution, tuned for Horn input sets. The system requires that, in every resolution operation, either one parent clause is a FALSE unit clause or that one parent clause and the resolvant are interpreted as FALSE, in a given truth value interpretation. Semantic Paramodulation for Horn Sets adds paramodulation to the Semantic Resolution for Horn Sets deduction system. It requires that, in each paramodulation operation, either both parents are FALSE unit clauses or that one parent clause and the paramodulant are interpreted as FALSE, in a given truth value interpretation.

The semantically guided deduction systems described thus far are forward chaining deduction systems. In such systems there is typically a large number of potential parent clauses for each deduction operation, and truth value deletion helps to choose which to use. In back chaining deduction systems the choice of 'sub-goal' parent clauses is limited, in some cases to a single clause. For example, in linear-input and linear deduction systems the centre parent clause must be used. This limited choice makes truth value deletion of sub-goal parent clauses particularly effective. The first semantically guided deduction system [Gelerneter, 1963; Gelerneter et al., 1963], in effect, used this format of truth value deletion.

The most direct use of truth value deletion in back chaining deduction systems is in linear-input deduction systems. Here truth value deletion requires that every centre clause is interpreted as FALSE in all models of the side clauses of the deduction. The first explicit description of truth value deletion in a resolution based linear-input deduction system appears to be that by Brown [1973]. Brown describes a general form of truth value deletion for linear-input deductions using input sets that are, or are renamable to, Horn sets. Truth value deletion was, however, also implicitly present in earlier work [Kuehner, 1972], based on the positive truth value interpretation. The applicability and implementation of truth value deletion in linear-input deduction systems are discussed in section 3.3.

As linear-input deduction is incomplete for non-Horn input sets and truth value deletion is incomplete for linear deduction systems, truth value deletion has been rejected as unsuitable for linear deduction systems. One way of circumventing these problems is to employ splitting [Chang, 1972] to decompose a non-Horn input set into multiple Horn input sets, thus making linear-input deduction applicable. This technique is used in the Simplified Problem Reduction Format [Plaisted, 1982] and the Semantic Proof System

[Nie & Plaisted, 1990]. (Note that these systems are natural, as opposed to resolution based, deduction systems.) An alternative approach is introduced in section 3.4.
There are also some less main stream uses of truth value deletion in back chaining deduction systems. They are as follows.

- The Semantically Guided Deductive System [Reiter, 1973] is an incomplete natural deduction system which, being a validation system, requires sub-goals to be interpreted as TRUE in a given truth value interpretation.
- The MECHO system for for solving mechanics problems [Bundy, Byrd, Luger, Mellish, Milne and Palmer, 1979] uses the interval package INT [Bundy, 1984] to reject "nonsensical solutions" [Bundy, 1984, p. 398] that are generated by the algebraic manipulation program PRESS.
- An unsuccessful attempt has been made to use truth value deletion in Boyer-Moore theory [Kim, 1986].
- Hierarchical Deduction [Wang & Bledsoe, 1987] is a deduction system for non-Horn input sets, for which a truth value deletion system has been designed. The deletion system is, however, complete only in limited situations. More general application of the deletion system relies on specially designed truth value interpretations [Wang, 1985].
- Truth value deletion has been used successfully in deduction systems for non-standard logics [McRobbie et al., 1988].

3.2.2. Other Forms of Truth Value Semantic Guidance

Instance Suggestion

Truth value information can be used to suggest universe instances of parent clauses to be used in deduction operations. The Semantically Guided Deductive System [Reiter, 1973] exploits this approach for arbitrary problem domains, and Bledsoe [1983] used this approach in the domain of set theory. The centre chain instance and compile time systems, described in section 3.3, use this approach.

Determining the Deduction Operation

The choice of which deduction operation to use at each step in a deduction can be influenced by the use of truth value information. SLM [Brown, 1974] uses truth value information to control the use of its reduction rule. A simpler version of this control mechanism can be used in the Sequent-Style Model Elimination Strategy [Plaisted, 1990b].

Transforming the Input Set

The Semantic Proof System [Nie & Plaisted, 1990] requires that, before a deduction starts, the input set be extended by the addition of contrapositives. Truth value information is used to help determine which contrapositives of input clauses to use.

3.2.3. Sort Value Deletion

Definition 3.3 - Sort legality An atom is *sort legal* in a given sort value interpretation iff it has a ground universe instance that is interpreted as at least one truth value. In addition, the atom of a positive sort literal is sort legal only if it has a ground instance that is interpreted as TRUE, and the atom of a negative sort literal is sort legal only if it has a ground instance that is interpreted as FALSE. Literals and clauses are sort legal iff their constituent atoms are simultaneously sort legal. Note that if an atom is sort legal, its subterms are necessarily sort legal.

Sort value deletion prevents a deduction from containing sort illegal clauses. Sort value deletion thus imposes a simultaneous deduction restriction. The definition of sort legality indicates that the ground approach must be used to establish that an expression is sort legal. Imposing the requirement of sort legality onto a deduction from an input set is equivalent to ensuring that there is a corresponding deduction of the "relativisation" [Walther, 1983, p. 885] of the input set. In such a corresponding deduction, sort legality is ensured via the sort literals which are added to the original input set chains in the relativisation process. Primary examples of deduction systems that use sort value deletion are described here. A notable feature of these systems is that they redefine satisfiability in terms of the sort value interpretation in use. Thus the systems are not necessarily complete in the conventional sense.

The Many-Sorted Calculus [Walther, 1983] brought sort value deletion into the spotlight by finding an automatic solution to Schubert's Steamroller problem [Walther, 1984]. The Many-Sorted Calculus uses a sort value interpretation with a monomorphic interpretation of functors. The domain of sorts used is partially ordered and a variable may be substituted only by a term of equal or lesser sort. This restriction is effected by the use of many-sorted unification in the resolution and paramodulation operations. An extra deduction operation, called weakening, is used to permit the unification of variables which are of unrelated sorts but whose sorts have a common lesser sort.

The limiting feature of the Many-Sorted Calculus is its monomorphic interpretation of functors. The Many-Sorted Calculus with Polymorphic Functions [Schmidt-Schauss, 1985] extends the Many-Sorted Calculus to use a sort value

interpretation with a polymorphic interpretation of functors whose arity is greater than 0. The polymorphism is achieved by associating multiple sort signatures with each such functor.

The Many-Sorted Resolution system [Irani & Shin, 1985] also extends the Many-Sorted Calculus, by introducing the notion of aggregate variables. This permits the dynamic alteration of the sort value interpretation by adding new sorts formed from the intersection of existing sorts.

A more expressive 'sort legal' system is the LLAMA many sorted logic [Cohn, 1987]. Variables are non-sorted in this system, but the range of terms to which a variable can be instantiated is restricted by the argument position that the variable holds[8]. LLAMA introduced the notion of having three meaningful truth values (there called TT, FF and UU, corresponding to the TRUE, FALSE and UNKNOWN_TRUTH_VALUE truth values mentioned in section 3.1) in a sort value interpretation. As well as implementing sort value deletion, LLAMA also uses sort value information directly in new deduction operations which manipulate sort literals in clauses. For example, the evaluation operation 'resolves' sort literals against sort value information.

Sort value information has also been used in logical systems other than 1st order deduction systems, e.g., a Logic of Actions [Hayes, 1971], higher order reasoning [Robinson J.A., 1969; Henschen, 1972] and knowledge representation [Shin & Irani, 1984].

3.3. Truth Value Deletion in Linear-Input Deduction Systems

As is indicated in section 3.2.1, truth value deletion is a known semantic guidance system for resolution based linear-input deduction systems. However, the details of implementing this system are often ignored. It is also possible to generalise this system to be used with other deduction operations. In this section various properties of deduction operations and deduction systems are defined, and truth value deletion for linear-input deduction systems is developed in terms of these properties. This clarifies the implementational issues and broadens the field of applicability.

---

[8] The version of sort legality defined in definition 3.3 is actually more restrictive than that that implemented by the sort arrays in the LLAMA logic. The sort array approach may find an expression to be sort legal by virtue of the existence of an appropriate sort with which to instantiate a variable, even if there are no universe elements of that sort. In definition 3.3 the existence of an appropriate universe element is required.

Definition 3.4 - Truth value soundness
A deduction operation is *truth value sound* if it can deduce a chain that is interpreted as FALSE only if one of the parent chains of the operation is interpreted as FALSE, in a given truth value interpretation.

Resolution (i.e., including extension and reduction in chain format deduction systems), paramodulation and factoring are examples of truth value sound deduction operations. The semantic guidance systems developed in this chapter are designed for deduction systems that use only truth value sound deduction operations.

Definition 3.5 - Side chain models
A *side chain model* of a linear or linear-input deduction is a truth value interpretation that is a model of the side chains of the deduction.

Bundy [1983, p. 147] proves the completeness of an independent deduction restriction on linear-input deductions, that requires all centre chains to be interpreted as FALSE in all side chain models of the deduction. This restriction may be refined to a simultaneous deduction restriction, as is shown in theorem 3.6. The proof of theorem 3.6 is analogous to that of Bundy's theorem.

Theorem 3.6 - Truth Values in Linear-Input Deductions
In a linear-input refutation, all centre chains are simultaneously interpreted as FALSE in all side chain models of the refutation.

If the deduced centre chain of a deduction operation is interpreted as FALSE in a given truth value interpretation, then one of its parent chains is necessarily interpreted as FALSE in the truth value interpretation. If a deduced centre chain in a linear-input refutation is interpreted as FALSE in a side chain model of the refutation, then it must be that its centre parent chain is interpreted as FALSE in the side chain model. Let D be such a deduced chain and let its centre parent chain be C. If D is interpreted as FALSE in a side chain model, then it has a FALSE ground universe instance $D\theta$. Then $C\theta$ must be FALSE in the side chain model and hence C has a FALSE ground universe instance $C\theta\sigma$. The last centre chain in a linear-input refutation, the empty chain, is ground and is interpreted as FALSE in all truth value interpretations. Inductively, in a linear-input refutation, all centre chains simultaneously have ground universe instances that are interpreted as FALSE in all side chain models of the refutation. Therefore, in a linear-input refutation, all centre chains are simultaneously interpreted as FALSE in all side chain models of the refutation. **QED**

Theorem 3.6 establishes the completeness of a truth value deletion system for linear-input deduction systems. The deletion system requires all centre chains to simultaneously be interpreted as FALSE in all side chain models of the refutation. This system is called the *simultaneous centre chain* truth value deletion system. The ground approach is evidently appropriate for establishing the satisfaction of the restriction imposed by this system.

In order to use the simultaneous centre chain system in a deduction system, it is necessary to be able to determine in advance which input chains can be used as side chains. This is so that side chain models can be supplied. An extremely common solution to this proviso is to adopt the system only for deductions from Horn input sets, and to use a negative input chain as the top chain. In this environment only the non-negative input chains can be used as side chains, and a model of these chains is supplied. It is, however, unnecessary to so limit the use of these systems. In general, to use this system it is necessary merely to be able to determine which input chains may be needed as side chains in order to obtain a refutation. Having determined which are the potential side chains of a deduction, the side chain models can be constructed.

Definition 3.7 - Side chain predictability
A linear or linear-input deduction system is *side chain predictable* if it is possible to determine in advance which input chains may be needed as side chains in a refutation.

Example

A linear-input deduction system that builds deductions from sets of Horn chains, using a negative chain as the top chain, is side chain predictable because it can be determined that the non-negative input chains may be needed as side chains in a refutation.

A side chain predictable linear-input deduction system can employ the simultaneous centre chain system. In the deductions built, the top chain is subject to the truth value check like all other centre chains; i.e., the top chain must be interpreted as FALSE in all side chain models of the deduction.

3.3.1. Maintaining deduction faithfulness

The problems of imposing deduction restrictions, highlighted in section 1.5, apply to the restriction of the simultaneous centre chain system. If the restriction is imposed operationally, then the techniques of ensuring deduction faithfulness suggested in section 1.5 can be used. The proof of theorem 3.6 also suggests a deduction faithful operational version of the restriction.

Theorem 3.6 shows that every centre chain in a refutation has a ground universe instance that is interpreted as FALSE in all side chain models of the refutation. If, at deduction time, those instances are used instead of the centre chains themselves, a refutation isomorphic to the original would be found. By using the FALSE ground universe instances it is assured that the centre chains will remain FALSE throughout the deduction. Thus the simultaneous centre chain system may be implemented by using, at each deduction operation, a ground universe instance of the centre parent chain that is interpreted as FALSE, in all side chain models of the deduction. This is called the *centre chain instance* truth value deletion system. The restriction of the centre chain instance system is a deduction faithful operational implementation of that of the simultaneous centre chain system. Two benefits are gained. Firstly, the FALSE interpretation value of each centre chain is established only once, at the point of instantiation. Secondly, redundant deduction operations do not occur and any associated overheads are avoided. This contrasts with the simultaneous centre chain system which may detect the redundancy of a deduction operation only long after it is performed. The use of truth value information to suggest instances of parent clauses has been used in other deduction systems; e.g., the Semantically Guided Deductive System [Reiter, 1973]. As noted by Reiter [1973, p. 43], "the first [instantiated] goal will, in general, be much easier to prove than the second [uninstantiated]". The disadvantage of this type of system is the enlargement of the host deduction system's search space.

Although the centre chain instance system is effective, truth values still have to be established at deduction time. It is possible to reformulate the centre chain instance system so that the work required to maintain deduction faithfulness is performed prior to the start of the deduction, i.e., at compile time. The principle of the reformulation is to identify and reject at compile time those ground universe instances of input chains that, if used as side chains, would cause the deduced centre chain to be rejected. This is possible if one can determine a priori which literals of an input chain will be discarded when the input chain is used as a side chain.

Definition 3.8 - Discard predictability
A deduction system is *discard predictable* if it is possible to determine which literals of an input chain will be discarded when that input chain is used as a parent chain in a deduction operation. Deduction systems that are both side chain predictable and discard predictable are *side chain discard predictable*.

Example

A linear-input deduction system that builds deductions from sets of Horn chains, using a negative chain as the top chain, discards the positive literal of the side chain used in each resolution operation.

The centre chain instance system uses only ground universe instance of centre chains that are interpreted as FALSE in all side chain models of the deduction. The literals of such ground universe instances are inherited from either the top chain of the deduction or from non-discarded literals of side chains of the deduction. Thus, to obtain a refutation, it is necessary to use only (i) ground universe instances of the top chain that are interpreted as FALSE in all side chain models of the deduction and (ii) ground universe instances of side chains in which literals that will not be discarded in a deduction operation, are interpreted as FALSE in all side chain models of the deduction. Thus the centre chain instance system may be implemented by using only certain ground universe instances of input chains. The generation of the ground universe instances may be done before a deduction begins. This system is called the *compile time* truth value deletion system. As indicated, this system requires that the deduction system be discard predictable. If the choice of literals to be discarded depends on the deduction operation used or other contextual information, then ground universe instances of input chains need to be generated for each possible scenario.

The compile time system has the same advantages and disadvantages as the centre chain instance system. That is, its restriction is a deduction faithful version of that of the simultaneous centre chain system, but the host deduction system's search space is enlarged. The compile time system does, however, have the advantage that the work required to impose truth value deletion is done prior to the start of the search for a refutation. Further, the FALSE interpretation value of non-discarded literals is established only once, at compile time. This is of benefit if a ground universe instance of a chain is used multiple times in a deduction. In the centre chain instance system the FALSE interpretation value must be established at each usage.

Discussion

The centre chain instance and compile time systems both enlarge the host deduction system's search space. In general there are infinitely many ground universe instances of centre and input chains to be considered. The centre chain instance system can sequentially create and use FALSE ground universe instances of each centre chain. This may, however, seriously degrade the performance of the deduction system, depending on how its search is arranged. The compile time system will not be able to generate the required instances of the input set if the universe in infinite. The root of these difficulties is that the prime

benefit of resolution, that instantiation is delayed for as long as possible, is revoked by these two systems.

For most of the truth value guidance systems described in this chapter, an added demand on the supply of semantic information is that it must be possible to check that the truth value interpretation in use is a side chain model of the deduction. With semantic information supplied in the format described in section 3.1, this is typically an undecidable problem. However, the task is vastly simplified if an appropriate interpretive structure is provided to store the semantic information. This issue is addressed in chapter 4.

The utility of the centre chain instance and compile time systems has not been established, and is not a focus of this research. The simultaneous centre chain system is, however, of more interest and in fact leads to a more general formulation which can be used in GLD (and other linear deduction systems).

3.4. Truth Value Deletion in GLD (and Other Linear Deduction Systems)

In section 2.5 linear-input subset analysis was introduced as a method for detecting situations in which GLD builds linear-input subdeductions. The truth value deletion systems for linear-input deduction systems, described in section 3.3, can be transferred to GLD linear-input subdeductions. This is a significant new idea, as truth value deletion has previously been considered incompatible with linear deduction.

Definition 3.9 - Rightwards subchains
The *rightwards subchain* of a literal in a centre chain is the subchain that consists of the literal and all literals to its right.

A truth value deletion system that requires all rightwards subchains of the top literal in a GLD linear-input subdeduction to be interpreted as FALSE, in all side chain models of the subdeduction, is complete. The system arises from the following results.

Lemma 3.10
In a GLD refutation in which only extension and truncation operations are performed, all centre chains are simultaneously interpreted as FALSE in all side chain models of the refutation. (In determining the interpretation value of a centre chain, only B-literals are considered.)

In this situation a GLD refutation is reduced to a linear-input refutation. The lemma then follows directly from theorem 3.6. **QED**

Lemma 3.11
In a linear-input subdeduction of a GLD deduction, all rightwards subchains of the top literal are simultaneously interpreted as FALSE in all side chain models of the subdeduction. (In determining the interpretation value of a rightwards subchain, only B-literals are considered.)

Let C_1R_1, ... ,$C_{n-1}R_{n-1}$,C_n be the centre chains of a linear-input subdeduction, so that R_1 is the top literal, each R_i is a rightwards subchain of the top literal and R_{n-1} is the top literal in the guise of an A-literal. Theorems 2.6, 2.10 and 2.15 show that no reductions against literals in any R_i are performed, so no literal in any C_i is used in deduction operations that affect the R_i. Therefore there is a refutation from the top chain R_1 using only extension and truncation operations. By lemma 3.10, all the R_i are simultaneously interpreted as FALSE in all side chain models of that refutation. Thus in the linear-input subdeduction all the R_i are simultaneously interpreted as FALSE in all side chain models of the subdeduction. **QED**

Theorem 3.12 - Truth Values in Linear Deductions
In a linear-input subdeduction of a GLD deduction, all rightwards subchains of the top literal are simultaneously interpreted as FALSE in all side chain models of the subdeduction. (In determining the interpretation value of a rightwards subchain, all literals, including A-literals, are considered.)

In a linear-input subdeduction, A-literals in rightwards subchains of the top literal also occur as B-literals in ancestor rightwards subchains. By lemma 3.11 the ancestor rightwards subchains are simultaneously subject to the truth value restriction. Therefore the A-literals can be included when determining the interpretation value of a rightwards subchain. **QED**

Theorem 3.12 establishes the completeness of a truth value deletion system for GLD when in linear-input configuration. The system is analogous to the simultaneous centre chain system and is called the *rightwards subchain* truth value deletion system. The restriction of the rightwards subchain system is a special case of the simultaneous deduction restriction, in that it is imposed only at some deduction operations. Again, the ground approach to establishing the satisfaction of the restriction is appropriate. If the restriction of the rightwards subchain system is imposed operationally, the checking of A-literals effects a retrospective check of B-literals. This helps to (locally) maintain deduction faithfulness of

the restriction. The maintenance of deduction faithfulness is local in the sense that it is maintained only within each linear-input subdeduction.

Example

An example, illustrating the effects of the rightwards subchain system, is to be found in appendix 1, section A1.3. See section 5.4.3 for further details.

As with truth value deletion in linear-input deduction systems, the imposition of the rightwards subchain system relies on side chain predictability. Here side chain predictability is required in each linear-input subdeduction. If Horn subset analysis is used to detect the linear-input subdeductions then GLD is trivially side chain predictable in each linear-input subdeduction. Only non-negative Horn subset input chains are used as side chains. For LISS analysis, side chain predictability is obtained by inspection of the extension tree. The input chains that may be used as side chains in a linear-input subdeduction are those that were used in building extension subtrees rooted at the LISS element corresponding to the top literal. Thus a set of possible side chains is associated with each LISS element. There are then two options for building side chain models, as follows.

- Different side chain models may be built for each LISS element, based upon the associated input chains. Although this may require significant effort, there may be some benefit in constructing models that are local to each possible linear-input subdeduction, as the models need reflect only truth value information relevant to the subdeduction. As is noted by Plaisted [1982, p. 238], "This is interesting because it corresponds to the fact that in the human theorem proving process attention is given to various specialised models at various stages of the proof."
- Side chain models of the union of the sets associated with the LISS elements may be built. This approach is possible only if the union is a proper subset of the input set. If all input chains are possible side chains, the LISS may be made smaller by adding a third condition for membership of the LISS - (iii) the top chain of the deduction is not used in forming any descendant of the literal structure. If this condition is added, then the top chain of the deduction cannot be a side chain in any linear-input subdeduction and models of the union are possible. If this latter approach excludes only the top chain from being a side chain then an additional truth value restriction, that requires the top chain of the deduction to be interpreted as FALSE in the side chain models, may be imposed. This restriction may be added because at least one input chain in a refutation must be interpreted as FALSE, in every truth value interpretation.

Side chain predictability for LISL analysis is analogous to that for LISS analysis.

As with linear-input subset analysis, the rightwards subchain system is readily transferred to linear deduction systems other than GLD. A noteworthy instance is that it is transferable

to chain format systems that use the GC procedure's C-literal mechanism. In these systems C-reduction can occur in, what would otherwise be, linear-input subdeductions. This added possibility does not affect the rightwards subchain system : All linear-input C-literals are inserted at the left most end of centre chains, indicating that they are logical consequences of the side chains that participated in their production. Therefore the C-literals are TRUE in all models of such side chains. Lemma 3.10 is easily extended to cover reduction against such C-literals, and the rightwards subchain system remains complete.

Truth value deletion systems analogous to the centre chain instance and compile time systems have not been formalized for GLD, although there do not appear to be any immediate difficulties with such systems. As is described in chapter 5, the rightwards subchain system has been employed in SGLD. The success of the rightwards subchain system is analogous to that achieved in the Simplified Problem Reduction Format in which the success of truth value deletion "seems to have something to do with the fact that Horn clauses are common in typical problems." [Plaisted, 1982, p. 238] However, the rightwards subchain system has a potential for greater success as a more general notion than Horn-ness is used to determine when truth value deletion can be used.

3.5. The FALSE-Preference Strategy

The imposition of any form of truth value deletion is limited by the requirement of completeness; i.e., a host deduction system must remain complete under the restrictions imposed by its deletion system. If a variant of such a host deduction system is designed, it may be necessary to drop its truth value deletion system in order to maintain completeness. For example, truth value deletion may be imposed on pure resolution but must be dropped when lock numbering is added. In other deduction systems, e.g., hierarchical deduction [Wang & Bledsoe, 1987], the imposition of truth value deletion destroys completeness in only some circumstances. A truth value guidance strategy that can never destroy completeness is described here. This is a new approach to semantic guidance and it broadens the range of deduction systems in which truth value guidance can be used fruitfully.

Truth value deletion systems rigidly expect one or more parent chains to be interpreted as FALSE, in a given truth value interpretation. By softening this expectation to a preference for FALSE parent chains, a new truth value guidance strategy is formed. The new strategy is called the *FALSE-preference* strategy. The FALSE-preference strategy guides a deduction system's search by showing a preference for parent chains that are interpreted as FALSE, in a given truth value interpretation. Clearly the use of the FALSE-preference strategy cannot destroy completeness. The FALSE-preference strategy can also be used in

conjunction with a truth value deletion system. In this scenario truth value deletion is imposed where ever it does not destroy completeness, and the FALSE-preference strategy is used where truth value deletion is 'appropriate' but cannot be imposed because completeness would be lost.

A FALSE-preference strategy that simply compares the interpretation values of chains does not differentiate between chains which are interpreted as the same truth value. To make a finer distinction between chains, and hence differentiate amongst chains with the same interpretation value, a *FALSEness level* can be assigned to each chain. The FALSEness of a chain is determined by looking at ground instances of the chain's literals, as follows.

- The FALSEness of a chain, in a given truth value interpretation, is a function of the FALSEness levels of its ground universe instances. The function must be optimistic, in the sense that it may return a poor FALSEness level only if all of its arguments are poor FALSEness levels.
- The FALSEness of a ground chain is a function of the FALSEness levels of its literals. The function must return a value roughly proportional to its arguments, i.e., so that if the arguments are mostly good FALSEness levels then the function value is a good FALSEness level, and vice versa.
- The FALSEness of a ground literal is GoodScore if the literal is interpreted as FALSE and BadScore otherwise. GoodScore and BadScore are parameters to the FALSE-preference system, with GoodScore being a better FALSEness than BadScore.

FALSEness thus measures the numbers of literals that are interpreted as FALSE in ground instances of a chain. A deduction system is guided by giving preference to parent chains with a better FALSEness level.

In chain format linear deduction systems the FALSE-preference strategy guides the search away from refutations that require reduction operations, as follows. If a centre chain B-literal is interpreted as TRUE in a model of all the forthcoming side chains in the deduction, then that B-literal, or some descendant through extension operations, must be reduced against. By guiding the search away from centre chains that contain TRUE literals, the FALSE-preference strategy guides the search away from refutations that contain reduction operations. In general, however, it is not possible to determine that a truth value interpretation is a model of "all the forthcoming side chains in a deduction". This is because linear deduction systems are not, in general, side chain predictable. The most that can be achieved is to use a truth value interpretation that is a model of as many input chains as possible. If the input set is minimally unsatisfiable, then a truth value interpretation that is a model of all but one of the input chains can be found. If a centre chain literal is interpreted as TRUE in such a truth value interpretation then it, or some

descendant through extension operations, must either be reduced against or extend against the input chain which is FALSE in the truth value interpretation.

There are other possible ways of determining a FALSEness level for a chain, besides that described here. This method corresponds directly to the ground approach of establishing that a chain is interpreted as FALSE, as is typically used by truth value deletion systems. The method is thus suitable for use in a truth value guidance system that combines truth value deletion with the FALSE-preference strategy.

3.6. Reformulating Sort Value Deletion

A common feature of the 'sort legal' deduction systems described in section 3.2.3 is that their deduction operations never deduce sort illegal clauses. This is in contrast to a deduction system which employs truth value deletion, in which the deduction operations independently deduce clauses and the truth value deletion system later rejects those which are unacceptable. It is evident from section 3.1 that truth value interpretations and sort value interpretations supply analogous information. This observation prompts a different formulation of sort value deletion, that is similar to truth value deletion. In this formulation, sort illegal chains may be deduced. Each deduced chain is subsequently checked, and rejected if not sort legal. This formulation parallels the approach taken in truth value deletion systems. The usual considerations concerning deduction faithfulness apply and the previously discussed techniques are again applicable. This approach to sort value deletion is extremely flexible as it does not affect the deduction operations of the host deduction system. It is therefore not tied to a specific deduction system.

The proposed use of this formulation of sort value deletion is to impose the sort legality restriction in deduction systems which are designed independently of the deletion system. As a consequence, any deduction built subject to this sort legality restriction can also be built if the restriction is dropped. The imposition of this formulation of sort value deletion is thus sound. It is, however, possible for the imposition of this formulation of sort value deletion to prune refutations from the search space of the host deduction system. This is evident from the fact that a deduction constructed subject to sort value deletion corresponds to a deduction from the relativisation of the input set. This means that a refutation of the input set, subject to sort value deletion, can be found only if there is a corresponding refutation of the relativised input set. In existing deduction systems that employ sort deletion this problem has been overcome by redefining satisfiability to be in terms of the sort value interpretation in use. The completeness of such deduction systems is then relative to the sort value interpretation in use. Here, where the deletion system is independent of the host deduction system, such redefinition is not appropriate. Thus this

formulation of sort value deletion is used under the cloud of possibly introducing incompleteness into the host deduction system. However, experiences using this deletion system have been positive and loss of completeness has not been a problem.

Finer Granularity

Every atom structure in a deduction necessarily appears at least twice in the input set. In some problems it is appropriate to provide a different sort value interpretation for each distinct occurrence of an atom structure. One way of implementing this is to rename the predicate symbols of the atoms to unique names, and to add clauses which specify the equivalence of the atoms. The specification of a sort value interpretation can then proceed as normal. An alternative, and more direct, approach is to specify the base-relation separately for each occurrence of an atom structure. The interpretation values of ground instances of an atom are then defined in terms of the base-relation for that particular occurrence.

3.7. Combined Truth Value and Sort Value Guidance

3.7.1. Sort&Truth Value Deletion Systems

The formulation of sort value deletion in section 3.6 parallels the approach taken in truth value deletion systems. As a result, the combination of these two systems becomes quite natural. Although it is known that truth value deletion and sort value deletion can be used in parallel, the novelty of the approach described here is that the two are combined into a single system. This makes the incorporation of the two into a deduction system much easier. The combined systems, called *sort&truth value deletion systems,* use a sort value interpretation. Specific sort&truth value deletion systems are formed by selecting a truth value deletion system to combine with sort value deletion. For a seamless combination it is necessary for the selected truth value deletion system to, like sort value deletion, impose a simultaneous deduction restriction. The rightwards subchain system is, for example, suitable. Given an appropriate truth value deletion system, the resultant sort&truth value deletion system simultaneously expects (i) all atoms in a deduction to be sort legal, i.e., all atoms must have ground universe instances that are interpreted as some truth value (any one of TRUE, FALSE or UNKNOWN_TRUTH_VALUE), (ii) the restrictions of the truth value deletion component to be met, i.e. some parent chains are expected to be interpreted as FALSE and (iii) the sort legality restrictions on sort literals to be met, i.e. sort literals are expected to have ground universe instances that are interpreted as TRUE or FALSE, as described in section 3.6. (The possibility of a conflict between the latter two requirements is discussed below.)

The sort value deletion component of a sort&truth value deletion system dictates that the ground approach must be used to establish the satisfaction of the system's restrictions. This is completely convenient in terms of the truth value deletion component. Issues of deduction faithfulness are addressed as before.

The use of a sort value interpretation in a sort&truth value deletion system has some pragmatic advantages. When building a truth value interpretation it is realistic that the interpretation value of a given base element may not be known independently of its actual truth value in the domain of discourse [Delgrande & Mylopolous, 1986, p. 10]. Nevertheless it is necessary to make a commitment to one of the truth values TRUE or FALSE. In contrast, if a sort value interpretation is used then such base elements can be mapped to UNKNOWN_TRUTH_VALUE. An atom that is expected to have a ground universe instance that is interpreted as one of TRUE or FALSE, can be accepted if it has a ground universe instance that is interpreted as UNKNOWN_TRUTH_VALUE. This approach can also be used in truth value deletion systems.

As mentioned above, it is possible for the expectations of the truth value component of a sort&truth value deletion system to conflict with those of the sort value component. This situation occurs when a positive sort literal is expected to be interpreted as FALSE by the truth value component. If the sort value interpretation in use correctly reflects the intended interpretation of the input set, then the expectation of the truth value deletion component is unlikely to be met in a refutation. However, no formal results in this area have been established. As with sort value deletion systems, experiences using sort&truth value deletion systems have been positive.

3.7.2. Combined Semantic Guidance Systems

Section 3.5 introduced the softening of truth value deletion to the FALSE-preference strategy. The notion of combining truth value deletion and the FALSE-preference strategy into a single semantic guidance system was also suggested. An analogous combination of sort&truth value deletion with the FALSE-preference strategy is equally desirable. These *combined semantic guidance* systems encompass a rich spectrum of semantic guidance ideas. A combined semantic guidance system uses a sort value interpretation. The restrictions of sort&truth value deletion are imposed and, in addition, the FALSE-preference strategy is used to guide the search of the host deduction system. As in sort&truth value deletion systems, the ground approach is used to establish the satisfaction of the deletion system's restrictions.

In a combined semantic guidance system, the sort&truth value deletion system expects every literal in a deduction to simultaneously have a ground universe instance that is interpreted as some truth value. There are three possible expected truth values, TRUE, FALSE or 'any truth value' (any one of TRUE, FALSE or UNKNOWN_TRUTH_VALUE). A sort value interpretation may interpret a literal as one of TRUE, FALSE or UNKNOWN_TRUTH_VALUE, or may not interpret the literal as any value. Thus there are twelve possible scenarios. Given that the expectations of the sort&truth value deletion system are met, the FALSE-preference strategy must indicate its preference. These requirements are embodied in a quality measure called *expected-truth-value-compatibility* (ETV-compatibility).

The ETV-compatibility of a chain measures two things. Firstly it checks that the chain has a ground universe instance in which the interpretation values of the literals are acceptable to the sort&truth value deletion system. Secondly it biases the first measure in favour of chains whose literals are interpreted as FALSE. The measure is calculated as follows.

- The ETV-compatibility of a chain, in a given sort value interpretation, is a function of the ETV-compatibilities of its ground universe instances. The function must be optimistic, in the sense that it may return a poor ETV-compatibility only if all of its arguments are poor ETV-compatibilities. For a chain to be acceptable (to the sort&truth value deletion system) it must have at least one ground instance whose ETV-compatibility is not "deletion".
- The ETV-compatibility of a ground chain is a function of the ETV-compatibilities of its literals. The function must return a value roughly proportional to its arguments, i.e., so that if the arguments are mostly good ETV-compatibilities then the function value is a good ETV-compatibility, and vice versa. If the ETV-compatibility of any literal is "deletion", then the ETV-compatibility of the ground chain is also "deletion".
- The ETV-compatibility of a ground literal is is based on its expected and actual interpretation values, according to Table 3.13.

Actual Value	Expected Truth Value		
	FALSE	Any truth value	TRUE
FALSE	CorrectScore	GoodScore	deletion
UNKNOWN	OKScore	OKScore	OKScore
TRUE	deletion	BadScore	CorrectScore
No interp'n	deletion	deletion	deletion

Table 3.13 - ETV-compatibility

CorrectScore, GoodScore, OKScore and BadScore are parameters to the system. CorrectScore indicates better ETV-compatibility than GoodScore, which indicates a better ETV-compatibility than OKScore, which indicates better ETV-compatibility than BadScore. The "deletion" value is a special value, worse than any of the others.

The decreasing ETV-compatibility going down the first two columns of Table 3.13 implements the FALSE-preference strategy. Here it is extended to support the third possible interpretation value for a literal. The "FALSE" and "TRUE" columns in Table 3.13 deal with sort&truth value deletion. In the case where the truth value deletion component expects a FALSE interpretation value, an extension to the third possible interpretation value is supported.

The ETV-compatibility levels given above can clearly be split into finer groupings. The benefits of finer grouping have not been investigated.

3.8. Theory Resolution

Theory resolution [Stickel, 1985] provides a method of removing literals from clauses by virtue of their inconsistency with a given theory. This is in contrast to using deduction operations to prove their inconsistency with the input set. The manner in which the theory is specified is open. Theory resolution is total if no new literals are added to the theory resolvant.

An interpretation may be used to specify the theory to be used in total theory resolution operations which have a single parent chain. If a group of literals in the parent chain have an instance that is interpreted as FALSE in the interpretation, then those literals may be theory resolved away. The substitution that is used to form the instance is adopted by the theory resolution operation. The use of semantic information in theory resolution was also briefly mentioned by Plaisted [1982, p. 259]. The evaluation operation of the LLAMA logic is an instance of this form of theory resolution.

Theory resolution may be viewed as an extreme case of suggesting instances of parent chains to use in deduction operations, as described in section 3.2.2. The instance suggestion is made extreme by removing literals directly rather than using deduction operations.

3.9. Conclusion

The semantic guidance systems developed in this chapter contribute to the field of semantic guidance. They provide new ways of using semantic information to guide the searches of deduction systems. Of particular significance are :

- The non-specific manner in which the truth value deletion systems have been formulated.
- Implementational issues have been considered and addressed. This means that the systems developed can be implemented and used without further inquiry being necessary.
- The rightwards subchain system, which breaks the deadlock between truth value deletion and linear deduction systems.
- The FALSE-preference strategy, which open a whole new range of possibilities for using truth value guidance in deduction systems.
- The smooth integration of sort value deletion, truth value deletion and the FALSE-preference strategy.

For an efficient implementation of a semantic guidance system it is necessary to provide an efficient interpretive structure for storing and supplying the required semantic information. This is the focus of the next chapter.

Chapter Four

Designations

This chapter describes a new interpretive structure suitable for storing and supplying the semantic information used by semantic guidance systems. The new structures are called designations. The difficulty of storing and supplying semantic information is one of the factors that has discouraged the use of semantic guidance systems. There is a need for an interpretive structure that is expressive, space efficient, effective in supplying semantic information and also user friendly. A common approach is to store the semantic information as semantic functions. Interpretation of ground expressions is then performed using recursive descent. Designations generalise this approach, inheriting its good properties and remedying some of its faults. The domains of designations are limited to be finite.

This chapter contains :

1. A survey of existing interpretive structures.
2. An investigation of the inherent properties of the semantic relation + recursive descent approach.
3. Discussion of the underlying design ideas of designations.
4. The formal definition of designations.
5. A description of how 1st order languages are interpreted using designations.
6. Discussion of the use of designations in semantic guidance systems.
7. An algorithm for building designations.
8. Concluding comments.

4.1. Interpretive Structures

"It appears to be quite challenging both to represent and access the large variety of examples the human has available". [Bledsoe & Hodges, 1988, p. 517].

Introduction

In section 3.1 a generic form of the semantic information that is used by semantic guidance systems was given. An interpretive structure is a data structure which stores such semantic information. The primary requirement of an interpretive structure is that it must be expressive enough to store the semantic information. Given unlimited storage capacity, such a structure can always be constructed. However, from a pragmatic view point, an interpretive structure must also be reasonably space efficient. Besides these expressiveness and space requirements, two further criteria can be used to measure the quality of an interpretive structure, as follows. (i) For semantic guidance systems that use the ground approach to establishing the satisfaction of their restrictions (i.e., the majority semantic guidance systems, including almost all those described in chapter 3), the semantic information is used to determine if an atom has a ground universe instance that is interpreted as a given truth value. Thus a fundamental requirement for an interpretive structure is that it should be possible to make this determination in effective manner. This is referred to as the structure's *semantic checking* capability. Several researchers have commented on this point - "... deduction in a model M must be able to be performed extremely efficiently ..." [Brown, 1974, p. 31], "... determining whether or not a clause containing variables is falsified may be a formidable job." [Henschen, 1976, p. 820] and "... testing if a clause is false in I can be expensive or impossible if I is non-trivial" [Plaisted, 1990a, p. 296]. (ii) The original source of the semantic information stored in an interpretive structure is typically human. The task of specifying the semantic information must be sufficiently easy for the user, so that real benefits are obtained from using the information in a semantic guidance system. It is unfortunate that, for many interpretive structures, the specification of the semantic information is prohibitively complex and/or time consuming.

Thus, in summary, there are four axes along which the quality of an interpretive structure will be measured :

- Is the structure expressive enough to store the semantic information?
- Does the structure store semantic information in a space efficient manner?
- Does the structure provide an effective semantic checking mechanism?
- Is the specification of the semantic information an acceptably easy task from a user's point of view?

Finding an interpretive structure which satisfies these criteria is important if semantic guidance systems are to be of utility.

Existing interpretive structures have been designed to supply either truth value or sort value information. A wide variety of interpretive structures have been proposed and used, with varying levels of success along each of the four criterion axes. A common approach is

to store the semantic information as semantic functions and to interpret ground expressions using recursive descent. See, for example, the truth value interpretations in [Lloyd, 1984]. Designations, as described in this chapter, generalise on this format by using semantic relations rather than semantic functions. As well as those based on a semantic functions, other formats of interpretive structures have been used for storing truth value semantic information (to date, interpretive structures used for storing sort value semantic information appear to be exclusively semantic function based). Before defining and examining the properties of the semantic relation format, it is instructive to examine briefly the other formats.

Existing Interpretive Structures

The simplest interpretive structures for fully supplying semantic information are those that explicitly store the universe- and base-relations. A common approach is to partition the universe and base according to their images. If, as in many applications of truth value interpretation, only a base-relation to {TRUE, FALSE} is required, a simplification of the explicit storage approach is to store explicitly only the base-relation mappings to TRUE/FALSE and to use a closed/open world assumption to implicitly store the mappings to FALSE/TRUE. Latent models [Slagle, 1967] take this approach. This approach is workable if the set of base elements that map to TRUE/FALSE is finite. However, the partitioning approach is almost always impractical, due to the infinite sizes of the partitions.

Examination of the syntax of expressions can be used to implement the universe- and base-relations of an interpretation. For example, the positive and negative truth value interpretations examine the sign of a literal, and the predicate partition interpretation determines the interpretation values of base elements and atoms from their predicate symbols. Syntactic approaches such as these have the advantage of being able to directly interpret non-ground expressions. Their weakness is not taking into account the arguments of the expressions being interpreted. A syntactic approach to implementing a base-relation for truth value interpretation, which takes expression arguments into account, is provided by partition settings [Loveland, 1978]. The interpretive structure of partition settings is (in essence) a set of special literals called generators. If a literal is specified as a generator then it and all its universe instances, are mapped to TRUE. A closed world assumption is then taken to store implicitly the mappings to FALSE.

Syntactic approaches to implementing interpretations, although readily implemented, are non-intuitive and are incapable of representing non-trivial semantic information. Interpretive structures that associate more directly with the problem domain are easier for a user to work with, and higher level concepts may be used in the implementation. The

interpretive structure that appears to be closest associated with problem domains is that suggested by the examples in [Brown, 1974, p. 30]. In the examples, a truth value interpretation is stored as a set of clauses. Some of the interpretive clauses are taken from the input set and some are specific to an instance of the problem domain. A feature of this approach is that it allows the image of Skolem constants to range across multiple domain elements, as suggested by Reiter [1973]. Details of how the interpretive clause sets should be specified are not, however, given. A more common format of interpretive structure, which is also closely associated with a problem domain, is that which uses some data structure to represent an instance of the problem domain. The data structure is then interrogated. Examples of this approach are the diagrams used in the geometry proving machine [Gelerneter, 1963; Gelerneter et al., 1963], the piecewise continuous functions used by Bledsoe [1983] and the intervals used in the INT package [Bundy et al., 1984].

At the opposite end of the 'problem association' scale, is the approach of mapping the 1st order language in use to an "amicable" [Sandford, 1980, p. 120] interpretation language. The interpretation of an expression is achieved by interpreting the image expression in the amicable language. The amicable language may express concepts in a domain completely divorced from that of the original 1st order language, but in which interpretation is an extremely efficient process. One possible amicable language is that of simultaneous linear equations [Sandford, 1977].

Interpretive structures that closely associate with the problem domain have the advantage of being easily able to represent subtle features of the problem domain. Their disadvantage is that their applicability is limited to the problem domain. There is a compromise between the degree to which an interpretive structure associates with the problem domain and the generality of the interpretive structure. An intermediate form of interpretive structure, which is not entirely syntactic, but is also flexible enough to be used in a wide range of domains, is preferred. Further, the interpretive structures described above are all dedicated to one of truth value or sort value interpretation. Semantic guidance systems that combine sort value deletion and truth value guidance were presented in section 3.7. Thus there is a demand for an interpretive structure that can be used to store multiple types of semantic information, at a minimum both truth value and sort value information.

Excursus

Two problems that arise when storing semantic information are that (i) the amount of data that needs to be stored may be large and (ii) the universe- or base-relation for a given universe/base element may not be known independently of its actual value in the problem domain (see section 3.7.1). Two existing results that respectively tackle these difficulties have been presented, as follows.

- For truth value interpretation of Horn clauses, a complex interpretation may be formed by taking the cross product of two simpler interpretations [Henschen, 1976]. This approach has the potential for considerable savings in space and computational effort.
- Many semantically guided deduction systems are "false permissive complete" [Sandford, 1980, p. 83]. In the environment of a false permissive complete system, the interpretive structure can use a default that causes literals, which should be interpreted as TRUE, to be interpreted as FALSE. The translation of model schemes to sound semantic functions, described in by Sandford [1980], uses this approach. Note that sort checking is not false permissive complete because literals can be expected to be interpreted as TRUE. Thus this approach cannot be taken in the systems described in section 3.7.

4.2. Semantic Relation based Interpretive Structures

As has been noted, a common form of interpretive structure is that which stores semantic information as semantic functions, and interprets ground expressions using recursive descent. This section introduces a generalisation of this format. Semantic relations, rather than semantic functions, are used.

Definition 4.1 - Semantic relation based interpretive structures
A semantic relation based interpretive structure is a structure that consists of :
- A finite *domain*[9], the elements of which are constants.
- A set of truth values, the elements of which are constants.
- A *functor-relation* from d-functions to the domain.
- A *predicate-relation* from d-predicates to the set of truth values.

Definition 4.2 - SRI structures
A survey of mathematical logic literature indicates that there is no existing name that specifically identifies the form of interpretive structure defined above. Thus, in this thesis, semantic relation based interpretive structures are called *SRI structures*.

An SRI structure can supply the semantic information required by semantic guidance systems, as described in section 3.1.

[9]At this point only unordered domains (i.e. those in which there is no relationship between elements) are considered. Domains which support relationships between elements, as often used in interpretive structures for sort value interpretation, are considered in section 4.2.4.

Definition 4.3 - Interpretation using SRI Structures

For an SRI structure, consisting of domain D, set of truth values T, functor-relation F and predicate-relation P, the semantic information supplied is :

- The domain of the interpretive structure.
- The set of truth values of the interpretive structure.
- A universe-relation, which for a universe element $f(t_1, ... ,t_n)$, produces all possible interpretation values $F(f(d_1, ... ,d_n))$, where each d_i is an interpretation value of t_i.
- A base-relation, which for a base element $p(t_1, ... ,t_n)$, produces all possible interpretation values $P(p(d_1, ... ,d_n))$, where each d_i is an interpretation value of t_i.

The universe- and base-relations extend naturally to extended-universe- and extended-base-relations, for the language being considered extended by the domain of the SRI structure :

- An extended-universe-relation which, for an extended-universe element $f(t_1, ... ,t_n)$, produces the interpretation value f if f is a domain element, otherwise all the interpretation values $F(f(d_1, ... ,d_n))$, where each d_i is an interpretation value of t_i.
- The extended-base-relation, for an extended-base element $p(t_1, ... ,t_n)$, produces all the interpretation values $P(p(d_1, ... ,d_n))$, where each d_i is an interpretation value of t_i.

If a universe/base element has been partially interpreted using the universe- and base-relations, then the interpretation process may be completed using the extended-universe- and extended-base-relations, with the same result as if the universe- and base-relations themselves had been used.

Interpretation using an SRI structure is thus a process of recursive descent.

Many existing interpretive structures are based on semantic relations. Standard truth value interpretations (see [Lloyd, 1984], for example) are based on semantic relations :

- The set of truth values is {TRUE, FALSE}.
- The functor- and predicate-relations are complete functions.

Semantic relations are also a common basis for sort value interpretations, either monomorphic (e.g., [Enderton, 1972]) or polymorphic (e.g., [Hayes, 1971]) :

- A typical set of truth values is {TRUE, FALSE, UNKNOWN_TRUTH_VALUE}.
- The functor- and predicate-relations are typically partial functions.
- For sort-d-predicates, the predicate-relation is to {TRUE, FALSE}.

4.2.1. Expressiveness and Space Efficiency

SRI structures store semantic information in a very fine grained manner, thus enabling them to capture the nuances of domain specific information. They thus satisfy the first criterion for measuring the quality of interpretive structures. That is, they are expressive

enough to represent semantic information. Unfortunately, the direct representation of the finely grained information is inefficient in terms of the information content. SRI structures have no immediate mechanisms for employing concepts such as property inheritance to reduce storage requirements. Each element of the semantic relation has to be stored separately and explicitly. Thus, in an unmodified format, SRI structures do not satisfy the second criterion for measuring the quality of interpretive structures. That is, they do not store semantic information in a space efficient manner

4.2.2. The Interpretation Process using SRI Structures

One approach to semantic checking (using any form of interpretive structure), is to generate and interpret ground universe instances of the atom under consideration. However, if the universe is infinite this approach is undecidable, as it is impossible to create all the ground universe instances. Semantic guidance systems which need to consider all the ground universe instances of an atom (e.g., systems that need to establish that an atom is interpreted as TRUE) are rendered completely inoperable in this situation. The universe is infinite whenever the 1st order language in use contains functors of arity greater than 0, which is the rule rather than the exception. Thus any semantic guidance system that generates and interprets ground universe instances of atoms, is incomplete for practical purposes.

When using an SRI structure to interpret a ground universe instance of an atom (a base element), the universe elements used to instantiate the variables in the atom are, in the interpretation process, replaced by their interpretation values (domain elements). The ground domain instance of the atom thus formed (an extended-base element), has the same interpretation values as the original base element. The extended-base element *represents* the base element in terms of interpretation. The instantiation and interpretation process can therefore be short circuited by directly instantiating variables in the atom with domain elements, and interpreting the resultant extended-base element using the extended-base-relation. From the point of view of a semantic guidance system, if an atom has a ground domain instance that is interpreted as a given truth value, then the atom has at least one (possibly infinitely many) ground universe instances that are interpreted as that truth value. This approach to semantic checking is called the *domain based* approach. The domain based approach has been used in existing semantic guidance systems [Kowalski & Hayes, 1969; Reiter, 1973; Wang, 1985], and could certainly be used in other semantic guidance systems which currently take other approaches, e.g., the False Substitution List system [Sandford, 1980].

The domain based approach has two advantages over instantiating with universe elements. (i) The process is finite whenever the domain is finite. (ii) Some computation is saved. The approach is sound provided that every domain element is the interpretation value of at least one universe element (an important proviso not made explicit in other systems that use this approach).

Single Level Expansion

If the domain in use is finite, the domain based approach solves the undecidability problem in the 'generate ground instance and interpret' approach to semantic checking. However, the effort of searching for a ground domain instance that is interpreted as the required truth value is still, as Kowalski and Hayes [1969, p. 97] noted, "likely to be prohibitive". It is, however, possible to refine the domain based approach to avoid the interpretation process. Rather than generating ground domain instances of an atom and then interpreting them, the inverses of the predicate- and functor-relations are used to expand first the required interpretation value and then recursively the resulting d-expressions' arguments, to find the required domain instance of the atom. This process is described below.

Definition 4.4 - Single level expansion

A *single level expansion* of a truth value/domain element is a d-expression which is related to the truth value/domain element by the predicate-relation/functor-relation of the SRI structure in use.

Definition 4.5 - The SLE Process

The *single level expansion (SLE) process* is a process that converts truth values/domain elements to extended-base/extended-universe elements by :
(i) single level expanding the truth value/domain element to a d-expression.
(ii) applying the SLE process to the arguments of the d-expression.

Given an initial truth value/domain element, applying the SLE process, until no domain elements remain, generates base/universe elements (possibly infinitely many) which are interpreted as the truth value/domain element. More usefully, the SLE process may be restricted to determine if an atom has a ground domain instance that is interpreted as a given truth value. In each step of the restricted SLE process, either (i) the domain element under consideration is identical to the corresponding subexpression in the atom (the subexpression is the same domain element), in which case that branch of the expansion is stopped, (ii) the domain element under consideration single level expands to a d-expression whose principal symbol and arity match those of the corresponding subexpression in the atom, or (iii) the subexpression in the atom is an uninstantiated variable, in which case the variable is instantiated to the domain element under

consideration, and that branch of the expansion is stopped. Option (ii) may have several possibilities, each of which needs to be considered. If all branches of the process stop, then the resultant ground domain instance of the atom is interpreted as the given truth value. The ground domain instance represents ground universe instances of the atom, that are also interpreted as the given truth value.

This approach to finding appropriate ground domain instance of atoms is more direct than generating ground instances. The SLE process allows SRI structures to satisfy the third criterion for measuring the quality of interpretive structures. That is, they can provide an effective semantic checking mechanism.

4.2.3. Specifying SRI Structures

In [Sutcliffe, 1987] a truth value semantic guidance system, employing an SRI structure, is presented. Experience using this system highlighted the difficulties of supplying the required truth value semantic information correctly. It is difficult to ensure that all the necessary mappings have been supplied and stored. Once stored, the large number of mappings makes the result difficult to comprehend. As a result, it is also difficult to make consistent modifications to such a truth value interpretation. Attempts to expand this system to take advantage of sort value information, which can be more complex than truth value information, further indicated the difficulties of supplying semantic information correctly. These difficulties would be exacerbated when storing semantic information in which a d-expression may be related to multiple values. These experiences are not unique - "... to design a suitable example for helping prove a hard theorem is not an easy matter." [Wang, 1985, p. 1201]. Thus, SRI structures do not satisfy the fourth criterion for measuring the quality of interpretive structures. That is, the specification of semantic information is not an acceptably easy task from a user's point of view.

4.2.4. SRI Structures and Ordered Domains

SRI structures, with unordered domains, are suitable for storing both truth value and sort value information. However, the inherent hierarchical nature of sort value information has suggested the use of ordered domains in SRI structures. This modification allows many functor-relation elements to be stored implicitly rather than explicitly. A d-function that is explicitly related to a given domain element, is also implicitly related to all domain elements higher in the order.

Ordering is imposed on a domain by structuring the domain in some fashion. Various structures have been used to date, e.g., unconstrained partial ordering [Walther, 1983;

Schmidt-Schauss, 1985], tree structures [Walther, 1985] and boolean lattices [Cohn, 1987]. It is desirable that the structuring imposed on a domain should support two important features, as follows. (i) If U_1, ... ,U_n are the sets of universe elements that are interpreted as the domain elements d_1, ... ,d_n respectively, then it must be possible to have a domain element $d_\cup$ such that the union of U_1, ... ,U_n is the set of universe elements that are interpreted as $d_\cup$. (ii) Similarly, it must be possible to have a domain element $d_\cap$ such that the intersection of U_1, ... ,U_n is the set of universe elements that are interpreted as $d_\cap$. If these two features are supported, then it is possible to express compactly that a universe/base element is related to multiple domain elements/truth values and that a universe/base element is not related to a given domain element/truth value [Cohn, 1987, p. 119].

The use of ordered domains in SRI structures has two advantages. (i) It allows the natural structure of sort value information to be encoded into the interpretation. (ii) It is more space efficient than using an unordered domain. Designations use ordered domains and comply with the union and intersection requirements.

4.3. The Design of Designations

Designations[10] are SRI structures, designed to retain the expressiveness and semantic checking capability (via the SLE process), but using ordered domains to make them more space efficient and easier to specify. Designations are thus a new form of SRI structure that improve upon existing SRI structures. The important new feature in designations is a relation between domain elements. This feature enables the notion of property inheritance to be utilised.

Designations are designed to be extremely flexible interpretive structures, which can be used to store a large range of types of semantic information. To this end designations make no distinction between d-functions and d-predicates, nor between domain elements and truth values. This generality may be undesirable for some applications, e.g., it violates the standard truth value semantics of 1st order languages. Where necessary, restrictions are imposed so that the semantic information stored in a designation conforms to necessary restrictions on the nature of that semantic information. A result of this homogeneity is that the basic semantic information supplied by a designation differs from that described in section 3.1.

---

[10] The name "designation" has been adopted from Newell and Simon's Turing award lecture [Newell & Simon, 1976, p 116].

Definition 4.6 - Basic semantic information in designations
A designation of a 1st order language is an interpretive structure that supplies :

- A *domain*, the elements of which are constants.
- A *unibase-relation* from the unibase to the domain.

A corner stone of SRI structures is the use of relations from d-expressions to truth values and domain elements. It has been noted previously that it is the fine grained nature of these relations that causes SRI structures to be space inefficient and hard to specify. To solve these problems, designations represent the individual relationships in a new way. The approach taken extrapolates from a feature inherent in the semantic relation approach, that multiple d-expressions can relate to the same truth value/domain element. In particular, d-functions that relate to the same domain element form an equivalence class in terms of interpretation. They are interpretationally equivalent when they appear in arguments of other d-expressions. In designations this feature is extended to form equivalence classes of domain elements that are interpretationally equivalent when they appear as arguments of d-expressions. The equivalence classes are associated with individual argument positions of principal symbols, so that a domain element may belong to different equivalence classes in different situations. This association with argument positions provides more finely grained equivalence classes than those of d-functions, which are global to the interpretive structure.

Designations' equivalence classes of domain elements are represented by relating all the domain elements in a class to a single 'class' domain element. Domain elements that belong to multiple equivalence classes are related to multiple 'class' domain elements. Any d-expression with a given principal symbol and with equivalence class elements in appropriate argument positions (i.e., positions associated with the corresponding equivalence classes), is not explicitly related to a domain element. Rather, the d-expression formed by replacing the equivalence class elements by their 'class' domain elements, is related to the desired domain element. The former d-expressions inherit their image in the relation from the latter. Equivalence classes of 'class' domain elements can also be formed, with the inheritance of images being transitive. In this manner designations can be significantly more space efficient than standard SRI structures.

If the feature of relating domain elements to multiple domain elements is used in an unrestricted fashion then semantic inconsistencies can arise, as follows. Two domain elements are *semantically disjoint* if neither is in the 'class' of the other and there is no other domain element which is in both their 'classes'. If an argument of a unibase element is interpreted as multiple domain elements, then the unibase element may be interpreted as multiple, semantically disjoint, domain elements. This is undesirable, as it contradicts the

nature of functions and predicates. This semantic inconsistency also arises if d-expressions are related to multiple, semantically disjoint, domain elements. To avoid these problems, restrictions are imposed to ensure that all d-expressions relate to at most one domain element, i.e., their relationship to the domain is a partial function. As part of these restrictions, property inheritance in designations is constrained so that a relation element may be defined only if the relationship cannot be inherited, and vice versa. That is, defined and inherited relationships are mutually exclusive. This is different from standard property inheritance in which a property may be defined even if it can be inherited, and the possible inheritance is over-ridden.

4.4. The Formal Definition of Designations

Definition 4.7 - Designations

A *designation* of a 1st order language is an interpretive structure that consists of :

- A finite *domain*, the elements of which are of arity 0.
- An *expression-relation* which
  - (i) maps d-expressions to domain elements (i.e. the expression-relation is a partial function in this case);
  - (ii) relates, in an acyclic manner, domain elements to domain elements.

Example

A designation of the 1st order language L described in section 1.4, consisting of domain D and expression-relation R, is :

D = {`mr_s`, `mrs_s`, `person`, `FALSE`, `TRUE`}

R = {`homer` $\varnothing_R$ `mr_s`,`heart_ok(person)` $\varnothing_R$ `TRUE`,
`spouse_of(mr_s)` $\varnothing_R$ `mrs_s`,`lungs_ok(mr_s)` $\varnothing_R$ `TRUE`,
`spouse_of(mrs_s)` $\varnothing_R$ `mr_s`,`lungs_ok(mrs_s)` $\varnothing_R$ `FALSE`,
`mr_s` $\varnothing_R$ `person`,`alive(person)` $\varnothing_R$ `TRUE`,
`mrs_s` $\varnothing_R$ `person`,`person(person)` $\varnothing_R$ `TRUE`}

An expression-relation determines a partial order on the extended-unibase of the 1st order language extended by the domain.

Definition 4.8 - The partial order <For a designation with expression-relation R, the partial order $<_R$ is the transitive closure of the immediate ordering $<'_R$ between d-expressions :

- $E_1 <'_R E_2$ if $E_2 \in$ expression-relation(E_1)
- $r(t_1, \ldots, t_n) <'_R r(s_1, \ldots, s_n)$ if there exists j such that $t_j <'_R s_j$, and for all $i \neq j$ $s_i = t_i$.

The partial order is used to determine that one d-expression is larger or smaller than another.

Example

In the example designation above :

`alive(spouse_of(homer))` $<'_R$
`alive(spouse_of(mr_s))` $<'_R$
`alive(mrs_s)` $<'_R$
`alive(person)` $<'_R$
`TRUE`

Thus `alive(spouse_of(homer))` $<_R$ `TRUE`.

A domain element is *maximal* if there is no domain element larger than it and *minimal* if there is no domain element smaller than it. Similarly, an extended-unibase element is maximal if all the domain elements in it are maximal and minimal if all the domain elements in it are minimal.

Example

In the designation above :

`mr_s` is a minimal domain element,
`person` is a maximal domain element,
`alive(spouse_of(mr_s))` is a minimal extended-unibase element, and
`alive(spouse_of(person))` is a maximal extended-unibase element.

Two properties of an expression-relation are also defined so that appropriate restrictions on the nature of a designation can be enforced.

Definition 4.9 - Redundancy

An expression-relation R is *non-redundant* if for every extended-unibase element E_1 there exists at most one d-expression E_2 such that $E_1 \leq_R E_2$ and expression-relation(E_2) is defined.

Definition 4.10 - Consistency

A redundant expression-relation R is *consistent* if for each extended-unibase element E_1 and every d-expression E_2 such that $E_1 \leq_R E_2$, every E_2 is mapped to the same domain element.

A non-redundant expression-relation is always consistent and an inconsistent expression-relation is necessarily redundant. A redundant but consistent expression-relation wastes space and computational effort. An inconsistent expression-relation leads to semantic inconsistencies. Consistency, in conjunction with the

mapping (as opposed to the more general relation between domain elements) of d-expressions to the domain, ensures that no unibase element is interpreted as multiple, semantically disjoint, domain elements. Henceforth, unless otherwise noted, all expression-relations are non-redundant.

A designation with a non-redundant expression-relation supplies semantic information. As the basic semantic information supplied by designations is different from that supplied by standard SRI structures, interpretation using designations is equally different.

Definition 4.11 - Interpretation using designations
For a designation consisting of domain D and expression-relation R, the semantic information supplied is :

- The domain of the designation.
- A unibase-relation, which for a unibase-element U produces all possible interpretation values d: $U <_R d$, $d \in D$.

The unibase-relation extends naturally to an extended-unibase-relation, for the language being considered extended by the domain of the designation :

- The extended-unibase-relation, which for an extended-unibase-element E produces all the interpretation values d: $E <_R d$, $d \in D$.

In a designation with a non-redundant expression-relation, a unibase element may be interpreted as multiple domain elements. The set of interpretation values contains totally ordered sequences of domain elements. Some domain elements may appear in more than one sequence. If the designation is specified in an appropriate manner (see section 4.7.2), it is semantically acceptable for a unibase element to be related to multiple domain elements in this manner. If the expression-relation is inconsistent, however, the total ordering may be lost as a result of an argument of a unibase element being interpreted as multiple, semantically disjoint, domain elements.

4.4.1. Expressiveness and Space Efficiency

As designations are SRI structures, they inherit the representational adequacy of such interpretive structures. Designations are more space efficient than standard SRI structures, due to the use of property inheritance. Although the domain of a designation is typically larger than an unordered domain, the typical number of relation elements that have to be stored is significantly smaller.

The union and intersection properties required of interpretive structures with ordered domains, described in section 4.2.4, hold for designations. Here the requirement is extended to be in terms of unibase elements, rather than universe elements.

Theorem 4.12 - The Union and Intersection Properties for Designations
For a given designation, let U_1, ... ,U_n be the sets of unibase elements that are interpreted as the domain elements d_1, ... ,d_n respectively. Then it is possible to have a domain element $d_\cup$ such that the the union of U_1, ... ,U_n is the set of unibase elements that are interpreted as $d_\cup$. Similarly, it is possible to have a domain element $d_\cap$ such that the intersection of U_1, ... ,U_n is the set of unibase elements that are interpreted as $d_\cap$.

- Add $d_\cup$ as a new domain element. Define expression-relation to relate each of d_1, ... ,d_n to $d_\cup$. The set of unibase elements interpreted as $d_\cup$ is the union of U_1, ... ,U_n. As $d_\cup$ relates to no domain elements, expression-relation remains acyclic. As no expression-relations for d-expressions containing $d_\cup$ are defined, non-redundancy is maintained.
- For a unibase element that is interpreted as at least one domain element, there is a single smallest domain element that is an interpretation value of the unibase element. The unibase element can only be interpreted as more than one domain element by virtue of that smallest domain element being smaller than other domain elements. Thus if the intersection of U_1, ... ,U_n is non-empty, there exists a smallest domain element $d_\cap$, smaller than each of d_1, ... ,d_n, that is the interpretation value of the intersection of U_1, ... ,U_n.

QED

4.5. The Interpretation Process using Designations

Definition 4.11 does not immediately indicate that interpretation using designations is a recursive descent process. An alternative definition of the unibase-relation, that makes this aspect more clear and facilitates algorithmic implementation, is as follows. Here expression-relation$^+$ denotes one or more applications of expression-relation.

- A unibase-relation, which for a unibase-element $p(e_1, ... ,e_n)$ produces all possible interpretation values expression-relation$^+(p(d_1, ... ,d_n))$, where each d_i is an interpretation value of e_i. Note that because the expression-relation is non-redundant, there will be at most one combination of d_is such that expression-relation$(p(d_1, ... ,d_n))$ is defined.

This definition of the unibase-relation also extends naturally to an extended-unibase-relation, exactly the same as the unibase-relation. If a unibase element has been partially interpreted using the unibase-relation, then the interpretation process

may be completed using the extended-unibase-relation, with the same result as if the unibase-relation itself had been used.

Example

An example of interpreting a unibase element using the designation in section 4.4, but following this definition, is as follows. This process should be compared with the example illustrating the partial order $<_R$.

To interpret `alive(spouse_of(homer))`,

Interpret `spouse_of(homer)`. To do this,

Interpret `homer`.

`homer` $\not\varnothing_R$ `mr_s`, and `mr_s` $\not\varnothing_R$ `person`, so

`homer` is interpreted as `mr_s` and `person`.

`spouse_of(mr_s))` $\not\varnothing_R$ `mrs_s`, and `mrs_s` $\not\varnothing_R$ `person`, so

`spouse_of(homer)` is interpreted as `mrs_s` and `person`.

`alive(person)` $\not\varnothing_R$ `TRUE`, so

`alive(spouse_of(homer))` is interpreted as `TRUE`.

Note that R, being non-redundant, is not defined for `spouse_of(person)` or `alive(mrs_s)`.

The Domain Based Approach and Single Level Expansion

The domain based approach to semantic checking generalises naturally to the designation scenario. Here the aim is to determine if an expression has a ground universe instance that is interpreted as a given domain element. When using a designation to interpret a ground universe instance of an expression (a unibase element), the universe elements used to instantiate the variables in the expression are recursively replaced by one or more of their interpretation values (domain elements), to form ground domain instances of the expression (extended-unibase elements). The replacement has two phases. Firstly each universe element is replaced by its smallest interpretation value (a domain element) and then, in the process of relating the encompassing d-expression to its smallest interpretation value, the domain element may be replaced by a larger domain element. The extended-unibase elements thus formed all have the same interpretation values as the original unibase element. The extended-unibase elements *represent* the unibase element in terms of interpretation. The instantiation and interpretation process can be short circuited by directly instantiating variables in the expression with domain elements, and interpreting the resultant extended-unibase element using the extended-unibase-relation. From the point of view of a semantic guidance system, if an expression has a ground domain instance that is interpreted as a given domain element, then the expression has at least one (possibly infinitely many) ground universe instances that are interpreted as that domain element. Soundness is again predicated on every domain element being the interpretation value of at least one universe element.

The restricted SLE process for semantic checking can also be transferred to the designation scenario. Here the SLE process is restricted so as to determine if an expression has a ground domain instance that is interpreted as a given domain element. To deal with the totally ordered sequences of interpretation values, the restricted SLE process is slightly modified. In each step, either (i) the domain element under consideration is identical to the corresponding subexpression in the unibase element (the subexpression is the same domain element), in which case that branch of the expansion is stopped, (ii) the non-minimal domain element under consideration single level expands to another smaller domain element, (iii) the domain element under consideration expands to a d-expression whose principal symbol and arity match those of the corresponding subexpression in the original expression, or (iv) the domain element under consideration is minimal and the subexpression in the original expression is an uninstantiated variable, in which case the variable is instantiated to the domain element and that branch of the expansion is stopped. Options (ii) and (iii) may have several possibilities, each of which needs to be considered. If each branch of the process stops, then the resultant ground domain instance of the expression is interpreted as the given domain element. The ground domain instance represents ground universe instances of the expression, that are also interpreted as the given domain element.

Restrictions (ii) and (iv) of this restricted SLE process prevent a variable from being instantiated to a non-minimal domain element. This restriction is necessary only if the variable occurs more than once in the expression. If a variable that occurs more than once is instantiated to a non-minimal domain element, the instantiation may prevent a match between another occurrence of the variable (now instantiated to the non-minimal domain element) and a smaller domain element. For this reason non-minimal domain elements may not generally be used to instantiate variables. (Walther [1983, p. 886] handles this situation by the use of a "weakening" rule, which is also discussed by Cohn [1987, p. 135], but is there unnamed.)

4.6. Using Designations

4.6.1 Implementing Interpretations

Designations provide an extremely flexible interpretive structure in which to store semantic information. As previously noted, the nature of designations may be restricted so that the stored semantic information conforms to predetermined standards. This is the case when designations are used for truth value or sort value interpretations.

Truth Value Interpretations

A designation with domain D and expression-relation R may be used for truth value interpretation :

- TRUE, FALSE $\in$ D
- R(P) $\in$ {TRUE,FALSE} if P is a d-predicate
- R(F) $\notin$ {TRUE,FALSE} if F is a d-function
- R(r(d_1, ... ,d_n)) is undefined if any $d_i \in$ {TRUE,FALSE}
- R must be defined so that every unibase element is interpreted as exactly one domain element.

Sort Value Interpretation

A designation with domain D and expression-relation R may be used for polymorphic sort value interpretation :

- TRUE, FALSE, UNKNOWN_TRUTH_VALUE $\in$ D
- R(P) $\in$ {TRUE,FALSE,UNKNOWN_TRUTH_VALUE} or is undefined, if P is a d-predicate
- R(F) $\notin$ {TRUE,FALSE,UNKNOWN_TRUTH_VALUE} if F is a d-function
- R(r(d_1, ... ,d_n)) is undefined if any $d_i \in$ {TRUE, FALSE, UNKNOWN_TRUTH_VALUE}

4.6.2. Semantic Guidance

Given a designation, the SLE process may be used for any semantic checking required. Thus almost all of the semantic guidance systems described in chapter 3 can use designations.

The two semantic guidance systems that cannot use designations directly are the centre chain instance and compile time systems. This is because the SLE process does not immediately generate ground universe instances of atoms, but rather indicates their existence. There are two possible solutions to this problem (but both complicate implementation), as follows. (i) Modify the SLE process so that domain elements, which are in a position to instantiate variables, are expanded to universe elements before instantiation takes place. All possible expansions to universe elements would have to be considered. (ii) Permit the instantiation of variables with domain elements and modify the unification algorithm used by the host deduction system so that the domain elements are further expanded in unification [Sutcliffe, 1987]. In the latter approach special care must be taken to ensure that common occurrences of domain elements, which arise from the instantiation of a variable that has multiple occurrences in a clause, are expanded equivalently in unification. Expanding common occurrences of a domain element

differently corresponds to one or more equality deduction operations, which must be justified later. Such variations may prove to be useful, similar to relation-matching [Manna & Waldinger, 1986] and the loose matching ideas of Bledsoe [1986]. This final complication does not arise if no variables have multiple occurrences, e.g., if clauses are linearised. As domain elements may represent infinite universe elements, a stronger form of 1st order reasoning is implemented by instantiating variables with domain elements.

4.6.3. Theory Resolution

The form of theory resolution described in section 3.8 may be implemented using a designation to express the theory. To obtain appropriate ground universe instances of the literals being theory resolved upon, the SLE process has to be modified as described in section 4.6.2. The alternative of instantiating variables with domain elements and further expanding in unification can also be adopted in theory resolution. Again both approaches lead to complications in implementation. However, in theory resolution there is a third option - (iii) Consider the domain elements to be functors of the 1st order language in use and allow them to remain in theory resolvants. This approach has been adopted in other theory resolution based systems, e.g., CLP(R) [Jaffar & Lassez, 1987], in which new numeric functors are introduced as required. Literals containing domain elements (from the instantiation of variables in literals that have previously been theory resolved upon) can only be resolved upon in a normal manner if they match with a variable in unification. Otherwise such literals can only be theory resolved upon.

4.7. Building Designations

The correct specification and maintenance of semantic information stored in SRI structures has been noted to be a hard task. The smaller size of designations makes completed designations easier to comprehend and consistently modify, but the difficulty of ensuring that all the necessary relation elements have been supplied and stored, remains. In this research, this difficulty has been overcome by providing the user with a mechanical specification interface. This interface examines the 1st order language which is to be interpreted, and queries the user for semantic information as required. The consideration of this pragmatic aspect of semantic guidance is useful.

When building a designation for the semantic guidance of a deduction from an input set, it is the language implicit in the input set that needs to be interpreted. If the designation is built for the implicit language then each domain element is the interpretation value of elements of the Herbrand universe or Herbrand base of the clauses. A designation based on the implicit language is usually appropriate for semantic guidance.

Algorithm 4.13, below, builds a designation with domain `D` and expression-relation `R`, for a given 1st order language. The general approach of the algorithm is to generate maximal d-expressions (line M4) and to obtain the images for such d-expressions (line B9). It is important that the d-expressions generated at line M4 remain maximal, even in the completed designation. This permits the expression-relation to be defined for such d-expressions without fear of a larger d-expression coming into existence. If the expression-relation is defined for a given d-expression and it is subsequently also defined for a larger d-expression, then the designation will be redundant. To ensure that the d-expressions generated remain maximal, the expression-relation is defined for new domain elements immediately after their addition to the domain (lines B12 to B14). This makes it impossible for non-maximal d-expressions to be generated at line M4, at any iteration of the repeat loop. The expression-relation is also not defined for d-expressions that can inherit their expression-relationships (line B3). The implementation of `GetValuesFromUser`, called at line B9, must ensure that (i) d-expressions map to a single value and (ii) the expression-relation image values obtained for a domain element do not cause the expression-relation to be cyclic.

The expression-relation may be undefined for a given d-expression. A d-expression may also be expanded to a set of smaller d-expressions so that the expression-relation can be defined for these smaller d-expressions. The choice of smaller d-expressions is such that every d-expression that would have inherited its expression-relationship from the first larger d-expression, now is one of, or inherits from one of, the smaller d-expressions. To implement the expansion of a d-expression into a set of smaller d-expressions, the larger d-expression is mapped to the special value `expand(`p_1`, ... ,`p_n`)`. The arguments $p_1, \ldots, p_n$ are integers which indicate which arguments of the larger d-expression should be replaced by smaller domain elements. The arguments in the specified positions are single level expanded in all possible ways (line C3), to form the set of smaller d-expressions. The expression-relation is then defined for these smaller d-expressions (line C3). The expression-relation is not defined for the larger d-expression and hence none of the smaller d-expressions is in danger of inheriting another expression-relationship from it. Whenever smaller d-expressions are generated by the expansion process, it is possible that one or more of them may inherit an expression-relationship. The expression-relation is not redefined for such d-expressions (line B3).

Algorithm 4.13 - Building Designations

```
M1  Procedure Main
M2  D:={}
M3  Repeat
M4      Build({r(e1, ... ,en) | r is a functor or predicate
    symbol and each ei is a maximal domain element})
M5      Until no new domain elements are found
M6  Remove expression-relation elements to 'expand' values

B1  Procedure Build(Instances) :
B2  For each r(e1, ... ,en) ∈ Instances do
B3      If there is no d-expression larger than
    r(e1, ... ,en) for which R is defined then
B4          If R(r(e1, ... ,en)) is known then
B5              CheckExpand(r(e1, ... ,en),R(r(e1, ... ,en)))
B6          Else If there exists a d-expression smaller than
    r(e1, ... ,en) for which either (i) R is defined or (ii)
    there exists a d-expression larger than it for which R
    is defined then
B7
        Store(r(e1, ... ,en) ∅R expand(GetPositionsFromUser))
B8                  CheckExpand(r(e1, ... ,en),
        R(r(e1, ... ,en)))
B9              Else V:=GetValuesFromUser(r(e1, ... ,en))
B10                 For each d ∈ V
B11                     Store(r(e1, ... ,en) ∅R d)
B12                     If d is a new domain element then
B13                         D := D ∪ {d}
B14                         Build({d})
B15                     Else CheckExpand(r(e1, ... ,en),d)

C1  Procedure CheckExpand(r(e1, ... ,en),Mapping)
C2  If Mapping = expand(Expansion_positions) then
C3      Build({r(d1, ... ,dn) | if i ∈ Expansion_positions
    and ei is not minimal then di is a single level expansion
    of ei, else di is ei})
```

Notes

- `GetPositionsFromUser` gets a non-empty set of integers from the user.
- `GetValuesFromUser` gets a set (possibly empty) of domain elements from the user.

There are two situations in which a d-expression may be expanded. Firstly, the user may decide to do this (line B9). Secondly, expansion may be necessary to maintain the non-redundancy of the designation. The latter situation arises when there exists a d-expression, smaller than the current target d-expression (selected at line B2), which already has or inherits an expression-relationship (line B6). If such a smaller d-expression exists it is necessary to expand the target d-expression (line B7) to prevent the smaller d-expression having or inheriting multiple expression-relationships. In this situation the `GetPositionsFromUser` routine prompts the user for argument positions to be expanded. `GetPositionsFromUser` uses the target d-expression to suggest to the user which argument positions need to be single level expanded. To do this, all d-expressions that are smaller than the target d-expression, and which have or inherit an expression-relationship, are examined. If the Nth argument in any such smaller d-expression is smaller than the corresponding argument in the target d-expression, then the Nth position is suggested for expansion. The rationale for this recommendation is that if none of those positions are single level expanded, only d-expressions that must be further expanded will be produced.

In each iteration of the repeat loop new domain elements, smaller than existing ones, may be added to the domain. Depending on the point at which they are added it is possible for d-expressions, that have such domain elements as arguments, to neither have nor inherit an expression-relationship. Thus the repeat loop continues until no new domain elements are added (line M5). At each iteration of the repeat loop it is necessary to restart the examination of d-expressions from maximal ones so that existing expression-relation elements, in particular those for which the image value is `expand( ... )`, are accounted for.

Example

A trace of algorithm 4.13, building a designation of the 1st order language L[S] described in section 1.4.1, is listed in appendix 1, section A1.1. The line numbers from algorithm 4.13 are shown and the trace is indented to the recursion level.

4.7.1. Non-redundancy, Soundness and Completeness

It is important that the designations built using algorithm 4.13 should be suitable for use in a semantic guidance system, using the SLE process for semantic checking. The properties required are as follows. (i) The designation must be non-redundant. (ii) Every domain element must be an interpretation value of at least one universe element. (iii) No unibase element should be precluded from having interpretation values. These properties are assured by the following theorems.

Theorem 4.14 - Building Non-Redundant Designations
Designations built by algorithm 4.13 are non-redundant.

The proof is covered in the discussion above, the critical points being :

- All d-expressions generated at line M4 remain maximal.
- The expression-relation is not defined for d-expressions that already have or inherit an expression-relationship (line B3).
- Target d-expressions are broken down whenever a smaller d-expression already has or inherits an expression-relationship (line B6).

QED

Theorem 4.15 - Building Sound Designations Every element of a domain built by algorithm 4.13 is an interpretation value of at least one unibase element.

The proof is by induction on the size of the domain. If the domain contains the single element `d`, it must have been obtained at line B9 as the image of a 0 arity functor `r`. Then `d` is an interpretation value of `r`. This establishes a base case. Assume that if the size of the domain is less than n then every domain element is the interpretation value at least one unibase element. Then if the size of the domain is n, let `c` be the latest element added to the domain. The element `c` could have been obtained in one of three ways. (i) `c` was obtained as in the base case and the same argument applies. (ii) `c` was obtained at line B9 as the image of a d-expression $r(e_1, \ldots, e_n)$. Each e_i is an element of the domain as it existed at the time of the creation of $r(e_1, \ldots, e_n)$ at line M4. At that time the size of the domain was less than n. So by the induction hypothesis each e_i is an interpretation value at least one universe element, say t_i. Then `c` is an interpretation value of $r(t_1, \ldots, t_n)$. (iii) `c` was obtained at line B9 as an image element of a domain element `d`. The element `d` was added to the domain at line B13. At the time when `d` was added to the domain, the size of the domain was less than n. By the induction hypothesis `d` is an interpretation value of at least one universe element, say `t`. Then `c` is an interpretation value of `t`. **QED**

Theorem 4.16 - Building Complete Designations
No unibase element is precluded from having interpretation values in a designation built using algorithm 4.13.

To show that no unibase element is precluded from having interpretation values, it is necessary to show that for every unibase element, a d-expression equal to or larger than it is at some point an element of the argument to `Build`. The d-expression represents the

unibase element in terms of interpretation. The proof is by induction on the depth of principal symbol nesting in unibase elements.

At the first iteration of the repeat loop, constants and propositional symbols, which are unibase elements (and also d-expressions) of depth 0, are elements of the argument to `Build`. This establishes a base case. Assume that for unibase elements of depth less than n, d-expressions equal to or larger than them have have been elements of the argument to `Build`. Let $r(t_1, \ldots, t_n)$ be a unibase element of depth n. Each t_i is of depth less than n, so by the induction hypothesis a d-expression equal to or larger than each t_i has been an element of the argument to `Build`. Let $d_1, \ldots, d_n$ be the maximal interpretation values of the t_i, if they exist. If any d_i does not exist then $r(t_1, \ldots, t_n)$ is necessarily not interpreted and the theorem is complete. Otherwise, in the iteration of the repeat loop after the last d_i is added to the domain, the d-expression $r(d_1, \ldots, d_n)$ is an element of the argument to `Build`. The d-expression $r(d_1, \ldots, d_n)$ is equal to or larger than $r(t_1, \ldots, t_n)$. Thus for every unibase element, a d-expression equal to or larger than it is at some point an element of the argument to `Build`. Therefore no unibase element is precluded from having interpretation values in a designation built using algorithm 4.13. **QED**

4.7.2. Pragmatics of building designations

The specification of SRI structures has been seen to be a complex and for the user, tiresome task. The interactive algorithm given above significantly simplifies the task, absolving the user of the responsibility of keeping track of domain elements and expression-relation elements. Experience using an implementation of the algorithm has lead to some refinements which further simplify the task of building designations.

General Issues

There are a few general observations relevant to the task of building a designation, as follows.

1. Mapping a d-expression to a non-minimal domain element, i.e., one that represents an equivalence class, may prevent its effective use. It will probably be necessary to define or inherit expression-relationships for d-expressions containing such a non-minimal domain element. (These expression-relationships are required for interpreting unibase elements that contain subexpressions whose smallest interpretation value is the non-minimal domain element.) Then the expression-relationships for smaller d-expressions, formed by replacing the non-minimal domain element by smaller domain elements, must be inherited.

Therefore d-expressions that map to the smaller domain elements could map directly to the non-minimal domain element and have the same effect.

2. In truth value and sort value interpretations, the range of the expression-relation for d-predicates is disjoint from that for d-functions. Further, the expression-relation is undefined for d-expressions with truth value arguments. It is therefore possible to phase the building of a designation for truth value or sort value interpretation. The first phase defines the expression-relation for d-functions, at the same time creating non-truth value domain elements. The second phase defines the expression-relation for d-predicates, creating truth value domain elements. This phased building process allows the user to deal with the two issues separately.
3. It has been noted that algorithm 4.13 builds non-redundant expression-relations, hence preventing a unibase element from being interpreted as multiple, semantically disjoint, domain elements. This does not, however, prevent the user from relating a domain element to multiple domain elements which are unrelated in the 'real world' semantics modelled by the designation. If this is done then unibase elements that are interpreted as the first domain element are also interpreted as those multiple, semantically unrelated, domain elements. This is undesirable. It is the user's responsibility to ensure that a domain element relates only to other domain elements which, in a semantic sense, encompass it.

Standard Domain Elements and Automatic Expression-relation Definition

In the building of designations for truth value or sort value interpretation, several domain elements regularly arise, either out of necessity or convenience. These standard domain elements have standard uses. It is possible to tune the designation building process to take this into account. From the users point of view, a particularly useful modification is the possibility of automatically defining the expression-relation. Automatic definition is also possible in cases besides those relating to standard domain elements.

Three standard domain elements are the truth values TRUE, FALSE and UNKNOWN_TRUTH_VALUE, typically mapped to by d-predicates. UNKNOWN_TRUTH_VALUE is a truth value that means one of TRUE or FALSE (but not both). UNKNOWN_TRUTH_VALUE should not be confused with the equivalence class of truth values, which would be represented by a 'class' domain element, such as TRUTH_VALUES, to which all of TRUE, FALSE and UNKNOWN_TRUTH_VALUE would be related. In both truth value and sort value interpretations, the expression-relation is not defined for d-expressions which have any of these domain elements as arguments. Such domain elements can therefore be ignored when building the maximal d-expressions in line M4 of algorithm 4.13.

As well as the UNKNOWN_TRUTH_VALUE, it is sometimes necessary to define an UNKNOWN_DOMAIN_ELEMENT that is mapped to by d-functions. The UNKNOWN_DOMAIN_ELEMENT represents a domain element whose exact identity is not known. The definition of an expression-relation may be automated for d-expressions which have the UNKNOWN_DOMAIN_ELEMENT as an argument. Any d-functions containing UNKNOWN_DOMAIN_ELEMENT map to UNKNOWN_DOMAIN_ELEMENT and d-predicates containing UNKNOWN_DOMAIN_ELEMENT map to UNKNOWN_TRUTH_VALUE.

Equality is often used in the 1st order formulation of problems. The definition of an expression-relation for equality d-predicates may be automated, under the assumption that all universe elements that are interpreted as the same minimal domain element are equal. To achieve this (i) equality d-predicates with the same non-minimal domain element in both argument positions are automatically mapped to the value `expand(1,2)`. This causes the domain-element to expand to smaller, and eventually minimal, domain elements, (ii) equality d-predicates with the same minimal domain element in both argument positions are automatically mapped to TRUE, (iii) all other equality d-predicates are automatically mapped to FALSE.

As indicated in section 3.1, sort-base elements are interpreted as one of TRUE or FALSE. The definition of an expression-relation may be automated for sort-d-predicates. If the predicate symbol and argument of a sort-d-predicate are the same domain element, then the sort-d-predicate is mapped to TRUE. If the the predicate symbol is smaller than the argument, or there is a domain element which is smaller than both the predicate symbol and the argument, then the sort-d-predicate is mapped to `expand(1)`. All other sort-d-predicates are mapped to FALSE. Note that the case where the argument is smaller than the predicate symbol does not arise.

In some interpretive structures, e.g., the truth value interpretations defined by Wang [1985], a standard domain element is the 'meaningless' domain element/truth value ("unknown" in [Wang, 1985, p. 1203]). Such a domain element/truth value is used as the image of d-expressions which are semantically meaningless. It is necessary to have such a domain element in some interpretive structures because the universe- and base-relations have to be complete functions. In designations there is no need for such a domain element/truth value. Rather the expression-relation is simply undefined for meaningless d-expressions. If such a domain element were to be used, then the definition of the expression-relation could be automated for d-expressions that have the 'meaningless' value as an argument. Such d-expressions would be automatically be mapped to the 'meaningless' domain element.

Rules of Thumb

This last section relates some handy rules of thumb which have been found to be effective when using algorithm 4.13.

- If a d-expression requires multiple argument positions to be expanded, then the positions should be expanded one at a time. If all the positions are expanded together, then all the d-expressions formed by using all combinations of expanded arguments must be considered. If the argument positions are expanded one at a time, this gives the opportunity to map larger d-expressions to domain elements.
- To obtain the maximum effect from a semantic guidance system, the designation being used should store as much semantic information as possible. This is achieved by specifying finely grained equivalence classes and by defining the expression-relation for smaller rather than larger d-expressions. This leads to a larger designation.
- In a truth value deletion system, a model that maximises the number of TRUE ground instances of non-discarded literals in input clauses, will be the most effective. This conclusion is also reached for HLR : "the best models for a given clause set are those which make as many clauses true as possible" [Sandford, 1980, p. 203].
- To obtain maximum space saving from using a designation, coarsely grained equivalence classes should be specified and the expression-relation should be defined for larger rather than smaller d-expressions.
- If the 1st order language in use has no constants, dummy domain elements have to be introduced to be used in the SLE process.

4.7.3. Flattening Designations

A designation, in which no d-expression maps to a non-minimal domain element, may be mechanically converted to designation with an unordered domain. The domain of the *flattened* designation has the minimal domain elements of the original designation as its domain elements. Its expression-relation is obtained by (i) selecting from the original designation those expression-relation elements that map d-expressions to (minimal) domain elements and then (ii) replacing the d-expressions by minimal d-expressions, that are smaller or equal in $<_R$, in all possible ways. The relationships between unibase elements and minimal domain elements are preserved by this process.

Theorem 4.17 - The Soundness and Completeness of Flattening

A minimal domain element is the interpretation value of a unibase element in a flattened designation iff it is an interpretation value of the unibase element in the original designation (in which no d-expression maps to a non-minimal domain element).

Each unibase element is interpreted as a minimal domain element in the original designation. There are three possible cases. (i) The unibase element is a functor of arity 0, in which case the expression-relation maps the unibase element to the minimal domain element. In the flattened designation this expression-relation element is retained. (ii) The unibase element is interpreted as the minimal domain element because it is smaller than a minimal d-expression that maps to the minimal domain element. In the flattened designation this expression-relation element is also retained. (iii) The unibase element is interpreted as the minimal domain element because it is smaller than a non-minimal d-expression that maps to the minimal domain element. In this case there exists a minimal d-expression that is larger than the unibase element and smaller than the non-minimal d-expression. In the flattening process an expression-relation element, mapping the minimal d-expression to the minimal domain element, forms part of the replacement for the expression-relation element mapping the non-minimal d-expression to the minimal domain element. Then the unibase element is still interpreted as the minimal domain element, now because the unibase element is smaller than the minimal d-expression. **QED**

The SLE process does semantic checking slightly faster in designations with unordered domains, as an ordered domain sometimes requires the expansion of a domain element to smaller domain elements. Flattening, however, negates two of the basic advantages of designations. (i) Space efficiency is lost. Although a flattened designation may have a smaller domain than the original, it typically will have many more expression-relation elements. (ii) The flattened designation is more complex than the original, thus making it more difficult to comprehend and consistently modify. There is thus a compromise between computational efficiency on one hand and space efficiency and comprehensibility on the other. Experience with using designations indicates that the increased computational efficiency does not warrant the losses incurred.

4.8. Conclusion

Designations are an improved SRI structure for interpreting 1st order languages. They provide a pragmatic solution to problems which are often finessed in the context of semantic guidance systems. Designations are specific to neither problem domain nor semantic information type. Designations thus have a broad range of applicability. In particular, designations may be used to store both truth value and sort value information for any problem domain. The supply of semantic information from designations is well suited to use by semantic guidance systems, and may also be used in a form of theory resolution.

Along the four criterion axes for measuring the quality of interpretive structures, designations provide the following results.

- Expressiveness : As designations are SRI structures, they inherit the expressiveness of such structures. Semantic information is stored precisely in designations, with no defaults taken (as, for example, in partition settings) or loss of information in the interpretation process (e.g., use of false permissiveness). The use of a relation between domain elements enhances the expressiveness of designations. It permits unibase elements to be interpreted as multiple, totally ordered sequences of, domain elements. This in turn allows direct specification of interpretations whose domains are naturally ordered. The domain ordering imposed on the domains of designations has the desirable union and intersection properties.

  Like most implementable interpretive structures, designations can store semantic information only for interpretations with finite domains. Both Plaisted [1984] and Wang [1985] have noted the advantages of an interpretive structure with a finite domain, and both have proposed mechanisms for dealing with infinite domains. No mechanism for dealing with infinite domains has been specifically designed for designations, but both Plaisted's and Wang's mechanisms could be used .
- Space efficiency : Designations store semantic information in a space efficient manner. This is achieved by the use of property inheritance. Designations thus overcome the major weakness of SRI structures with unordered domains. The smaller size of designations makes them easy to comprehend and consistently modify.
- Semantic checking capability : The SLE process, which can be used with designations, is a computationally effective semantic checking mechanism. Its formalisation in this thesis is beneficial. The SLE process avoids the search required by instantiation and interpretation approaches, the size of which is proportional to both the number of distinct variables in the unibase element being interpreted and the size of the set of instantiating elements (typically infinite when instantiating from the universe). Increased computational effectiveness may be obtained from the SLE process, at the expense of increased space requirements and complexity, by flattening designations.
- Specifiability : The specification of semantic information to be stored in a designation has been made into a mechanical process by supplying an interactive algorithm. The well organised 'question and answer' format of the algorithm is very effective for obtaining the required semantic information from users. Some semantic information can be specified automatically in the algorithm. The algorithm ensures that the designation built is sound and complete. It appears that this is the first time that this pragmatic aspect of specifying semantic information has been this thoroughly considered.

The work reported in this chapter means that an appropriate interpretive structure is now available for use in a semantic guidance system, used to guide a host deduction system. The next chapter describes such a combination.

Chapter Five

Semantically Guided Linear Deduction

This chapter describes the Semantically Guided Linear Deduction system (SGLD). SGLD is a semantically guided implementation of GLD. The implementation, in Prolog, combines GLD with a semantic guidance system. Designations are used to store the semantic information used. SGLD has some features that are not specified in GLD. These features improve the real time performance of the implemented system without changing the structure of the deductions or the search space. The performance of SGLD has been investigated.

This chapter contains :

1. A description of the overall structure of SGLD.
2. A description of SGLD's semantic guidance system and related features.
3. A table illustrating the effects of using different combinations of search strategy and designation in SGLD.
4. Performance results for a range of test problems and discussion thereof.
5. Concluding comments.

5.1. The Overall Structure

SGLD is a semantically guided implementation of GLD. Its semantic guidance system is a combined system, incorporating the rightwards subchain system, sort value deletion and a FALSE-preference strategy for centre chains. Designations are used to store the semantic information used. The significance of SGLD is that it is (apparently) the first implemented linear deduction system to employ semantic guidance. The only other linear system that can use semantic information is SLM [Brown, 1974]. The only known implementation of SLM [Tabada, 1992] does not exploit that facility.

SGLD has been implemented in Prolog. The implementation avoids the use of language features that are specific to a particular Prolog interpreter, thus making it readily portable.

The original implementation in Arity Prolog [Arity, 1988] has been easily ported to both muProlog [Naish, 1985] and SICStus Prolog [Carlsson & Widen, 1990].

5.1.1. Deduction Data Structures & Code

A central issue in the implementation of SGLD is its representation of input and centre chains. The representation is a list of Prolog terms, each of which has `a`, `b` or `c` as its functor. Each such term is called a *link* of the chain and, depending on its functor, is called an *A-*, *B-* or *C-link*. The first argument of each link is a literal of the chain, and the functor of the link indicates the class of the literal. A literal is represented by a unary Prolog term whose argument is the atom of the literal. The functor of such a term is either `++` or `--`, indicating the sign of the literal. These two functors are defined as prefix Prolog operators. Atoms are simply Prolog terms. Information associated with each literal is stored in its link. A-links contain an expected truth value and a scope value. B-links contain an expected truth value. C-links contain an expected truth value and a list of scope A-literals. This encapsulation of information in links permits easy manipulation of a literal and its associated information.

Example

The literal `~lungs_ok(P)` is represented by the Prolog term `--lungs_ok(P)`. An example of an A-link is `a(--lungs_ok(P),any,0)`.

A problem is presented to SGLD as set of input clauses. The input clauses are supplied as Prolog facts in a text file. The arguments of an input clause fact are (i) the clause's name, (ii) its status, one of `theorem`, `axiom`, or `hypothesis` and (iii) a list of the constituent literals. SGLD converts input clauses into input chains, which are stored as facts in the Prolog database. The arguments of an input chain fact are (i) the number of links in the chain (this improves the performance of unit extension and subsumption checking), (ii) the chain's name, (iii) a list of the constituent links, (iv) the chain's status, one of `theorem`, `axiom`, `hypothesis`, or `lemma` and (v) the origin of the chain. If an input chain is converted from an input clause then the origin is `input_clause`. If an input chain is formed in an A-truncation then its origin is a list of the deduction steps that led to its formation, and its status is `lemma`. To convert an input clause to an input chain, each literal of the input clause is placed into a B-link. The expected truth value in each such B-link is FALSE if the B-literal is a negative sort-literal, TRUE if the B-literal is a positive sort-literal, otherwise "`any`". The `any` value means any one of TRUE, FALSE or UNKNOWN_TRUTH_VALUE. These expected truth values are updated after linear-input subset analysis. The chain name and status are copied to the input chain from the corresponding fields in the input clause. No reordering of literals or clauses is performed in the conversion from input clauses to input chains.

Example

The input clause :

```
input_clause(life,hypothesis,
    [++heart_ok(P),
     ++lungs_ok(P),
     --alive(spouse_of(P))]).
```

is used to build the input chain :

```
input_chain__(3,life,
    [b(++heart_ok(P),any),
     b(++lungs_ok(P),any),
     b(--alive(spouse_of(P)),any)],
    hypothesis,input_clause).
```

Centre chains in SGLD deductions are stored with the rightmost literal first in the chain. This ordering provides easy access to the rightmost cell, to select a B-literal for the base deduction operation of a deduction chunk.

Example

The centre chain :

~heart_ok(P) [~lungs_ok(P)] 0 (~heart_ok(P))

~alive(spouse_of(P))

in which the A-literal is the only scope literal of the C-literal, is represented by the Prolog list :

```
[b(--alive(spouse_of(P)),false),
 c(--heart_ok(P),any,[--lungs_ok(P)]),
 a(--lungs_ok(P),any,0),
 b(--heart_ok(P),any)]
```

5.1.2. Designation Data Structures & Code

SGLD uses a named designation to store a sort value interpretation of the set of input chains. SGLD's semantic checking mechanism uses the restricted SLE process to interrogate the designation. Designations can be represented in one of two ways. The first way is direct, consisting of the domain and the expression-relation elements of the designation. This information is stored as facts in the Prolog database. These facts are used as input data to an implementation of the restricted SLE process.

Example

The name of the example designation in section 4.4 (now named `simpsons`) is stored as the fact :

```
designation(simpsons).
```

The domain of `simpsons` is stored as the facts :

```
domain_element(mr_s,simpsons).
domain_element(mrs_s,simpsons).
domain_element(person,simpsons).
domain_element(false,simpsons).
domain_element(true,simpsons).
```

The expression-relation of `simpsons` is stored as the single level expansion facts :

```
sle(mr_s,homer,simpsons).
sle(mrs_s,spouse_of(mr_s),simpsons).
sle(mr_s,spouse_of(mrs_s),simpsons).
sle(person,mr_s,simpsons).
sle(person,mrs_s,simpsons).
sle(true,heart_ok(person)).
sle(true,lungs_ok(mr_s)).
sle(false,lungs_ok(mrs_s)).
sle(true,alive(person),simpsons).
sle(true,person(person),simpsons).
```

The second representation compiles the domain and expression-relation elements into Prolog code that implements the restricted SLE process directly. Expansions from non-minimal domain elements to minimal domain elements are explicitly recorded, thus immediately supplying the minimal domain elements which instantiate variables (restricted SLE process, item (iv)). The compiled representation is significantly more efficient than the direct representation.

Example

The designation `simpsons` compiles to the Prolog program listed in appendix 1, section A1.2. This program implements the restricted SLE process for `simpsons`. The entry point is `expand_compiled__`/3.

5.1.3. Starting an SGLD Deduction

Search Style

The search style is user specified. The user selects one of the following search styles : literal-selected, literal-ordered, cell-selected or cell-ordered, as specified in section 2.3.3.

Choice of Top Chain

SGLD uses the entire input set as the support set. The order in which the input chains are used as top chains is thus, in general, determined by the search style of the deduction, as specified in section 2.3.3. In SGLD, however, any input chains with the status `theorem` are used before others. This allows the user to focus attention on certain input chains, e.g., input chains formed from the negation of the theorem to be proved. In the literal-ordered and cell-ordered search styles, heuristic values of deduced chains are calculated to determine the order in which input chains are used as top chains (see section 2.3.3). The heuristic value of a chain is dependent on the expected truth values in its links (see section 5.2.1), which in turn are dependent on the linear-input subset analysis. Linear-input subset analysis cannot take place until after the top chain is chosen. Therefore the heuristic values calculated here are necessarily unaware of the expected truth values determined by linear-input subset analysis.

Linear-Input Subset Analysis

SGLD employs LISS analysis to detect linear-input subdeductions. After a top chain has been chosen the LISS of the input chains is extracted using an implementation of algorithm 2.11. The LISS is recorded so as to be available throughout the deduction. As well as its contribution to the truth value deletion system described below, the LISS is also used to determine when reductions cannot be performed.

Initial Search Bound

SGLD supports two methods for setting the initial bound of its consecutively bounded search. The default method is to use the length of the top chain. This is the minimum value within which a refutation can be obtained - via a sequence of unit extension operations. The user may override the default by explicitly specifying an initial bound.

5.2. Semantic Guidance

5.2.1. The Semantic Guidance System

The three components of SGLD's combined semantic guidance system (the rightwards subchain system, sort value deletion and the FALSE-preference strategy) are implemented directly within the GLD search guidance mechanisms. The expected truth values stored in input chain links reflect the requirements of the deletion systems. The heuristic function used in SGLD determines ETV-compatibility, as described in section 3.7.2. Integers are used to represent ETV-compatibilities, a smaller value meaning a better ETV-compatibility. Thus CorrectScore < GoodScore <OKScore <BadScore. Calculating

the ETV-compatibility of a centre chain requires finding the ground instances of the chain in which no literal's interpretation value is incompatible with its expected truth value. For literals, links and chains, such ground instances are called *acceptable* instances. If a centre chain does not have an acceptable instance then the heuristic function fails to return an ETV-compatibility value. The centre chain under consideration is then (naturally within the Prolog implementation) rejected. The possibility of imposing the restrictions of the deletion systems after the base operation of a deduction chunk, as well as at the end of each chunk, has been tested. This option sometimes reduces the number of derivation operations performed, but it is consistently of negative utility. A noteworthy feature of SGLD's semantic guidance system is the way in which the three components have been integrated into a coherent whole.

Sort Value Deletion

The sort value deletion system rejects centre chains that are not sort legal, as described in section 3.6. The expected truth values for sort-literals are stored in their links when the input chains are created. Sort value deletion occurs only when the designation supplied contains sort value information. This is detected automatically.

Truth Value Deletion

The rightwards subchain system requires all rightwards subchains of the top literal in a linear-input subdeduction to be interpreted as FALSE in all side chain models of the subdeduction. This is as described in section 3.4. In SGLD a single side chain model is used to interpret all rightwards subchains.

To implement the restriction of the rightwards subchain system, the expected truth values in input chains' B-links are updated after the LISS analysis. Input chain B-links that contain linear-input literals (linear-input B-links) have their expected truth values changed to FALSE. This is as expected by the rightwards subchain system. If a linear-input literal is also a positive sort-literal, then a conflict arises. The rightwards subchain system expects the literal to be interpreted as FALSE and the sort value deletion system expects the literal to be interpreted as TRUE. In this situation a warning is issued and the expected truth value is set to FALSE.

The FALSE-Preference Strategy

In SGLD, the ETV-compatibility of a chain is found by estimating the minimum of the chain's ground instances' ETV-compatibilities (see section 5.2.4). The function for calculating the ETV-compatibility of a ground chain sums the ETV-compatibilities of the A- and B-literals in the chain. The sum of the B-literals' ETV-compatibilities is a measure

of the quality of the clause represented by the chain. The addition of the A-literals' ETV-compatibilities moderates the sequence of extension operations performed. This function also gives preference to shorter chains, in line with GLD's fewest-literals maxim.

5.2.2. Checking the Designation

It is important to ensure that the designation used in an SGLD deduction is a side chain model of all possible linear-input subdeductions. Further, the effect of the FALSE-preference strategy should be stronger if the designation is a side chain model of the entire deduction. Thus, before any deduction operations are performed, the designation is examined in these terms. Firstly, no input chain that may be a side chain in a linear-input subdeduction may be interpreted as definitely FALSE in the designation (ground instances that are interpreted as UNKNOWN_TRUTH_VALUE are considered acceptable). Secondly, a warning is issued for every other input chain which is interpreted as definitely FALSE in the designation. Such warnings are useful pointers to parts of the designation that could be modified to improve the semantic guidance.

Example

In the input set S given in section 1.4.1, the input chain :

```
~heart_ok(P) lungs_ok(P)
```

is interpreted as FALSE by the designation `simpsons`. By changing the expression-relation element :

```
lungs_ok(mrs_s) ∅R FALSE
```

to :

```
lungs_ok(mrs_s) ∅R TRUE
```

the input chain is interpreted as TRUE.

5.2.3. Semantic Chains

The deletion systems used in SGLD impose simultaneous deduction restrictions. In the SGLD implementation the restrictions are operationally imposed and extra mechanisms are employed to maintain deduction faithfulness. The difficulties of maintaining deduction faithfulness in semantically guided deduction systems, and the failure of many semantically guided deduction systems to support a satisfactory solution, are discussed by Sandford [1980, p. 41]. SGLD employs two mechanisms for maintaining deduction faithfulness in this context. Both mechanisms keep track of literals in ancestor centre chains.

The first mechanism makes use of the A- and C-literals in centre chains. Every A- or C-literal in a centre chain is (the complement of) a B-literal in an ancestor centre chain.

The A- and C-literals in a centre chain can therefore be used to retrospectively check the expected truth values of their ancestor B-literals. This idea is built into the rightwards subchain system which, by definition, checks A-literals when checking rightwards subchains in linear-input subdeductions. In the SGLD implementation C-literals are also given an expected truth value, opposite to that of their parent A-literals, and the C-literals are also checked. Similarly, the sort value deletion system requires all A- and C-literals to be sort legal. The examination of the A- and C-literals in a centre chain comes at no extra cost, in the sense that those literals have to be maintained in the centre chain anyway. This mechanism is effective while the A- and C-links remain in the centre chain. However, upon their truncation the effect could be lost.

To prevent loss of deduction faithfulness when A- and C-links are truncated from a centre chain, SGLD maintains a *semantic chain* in parallel with the centre chain. A- and C-links which have been completely removed from the centre chain, i.e. truncated A-links that do not lead to the insertion of a C-link and all truncated C-links, are placed into the semantic chain. The semantic chain must remain acceptable throughout a deduction. If it becomes unacceptable, then the corresponding centre chain is rejected. The semantic chain is an auxiliary data structure maintained specifically to enforce deduction faithfulness. As is stated in section 1.5, "The auxiliary data structures need keep only sufficient information to detect violations and do not need to store a complete history of the deduction". In fact, maintaining redundant information in such an auxiliary data structure decreases its utility. This view is noted in the context of False Substitution Lists (FSLs) - "some mechanism for reducing the size of FSL sets must be employed." [Sandford, 1980, p. 216]. The size of SGLD's semantic chain is controlled by removing those links that contain acceptable ground literals. Such literals will always remain acceptable. This principle could be extended to remove links whose literals have an acceptable ground instance and also have no variables in common with the centre chain. This extension has not been implemented due to the high cost of determining if Prolog variables in the semantic chain also appear in the centre chain. Semantic chains are more general than Sandford's False Substitution Lists, in that they allow any truth value as an expected interpretation value for a stored literal, not just FALSE.

5.2.4. Calculating ETV-compatibility

An acceptable instance of a chain is found by sequentially considering the links of the chain and using the restricted SLE process to find an acceptable instance of each link's literal. If at any stage there is no acceptable instance of the literal under consideration, the system backtracks to find alternative acceptable instances of previous literals. All the acceptable instances of a centre chain can be found by backtracking over all the acceptable

instances of the literals in the chain. The ETV-compatibility of the chain itself can then be calculated.

Example

The centre chain (in the SGLD representation) :

```
[b(--alive(spouse_of(P)),false),
 c(--heart_ok(P),any,[--lungs_ok(P)]),
 a(--lungs_ok(P),any,0),
 b(--heart_ok(P),any)]
```

has an acceptable instance when interpreted using `simpsons`. It is :

```
[b(--alive(spouse_of(mr_s)),false),
 c(--heart_ok(mr_s),any,[--lungs_ok(mr_s)]),
 a(--lungs_ok(mr_s),any,0),
 b(--heart_ok(mr_s),any)]
```

The ETV-compatibility value of this ground instance is

CorrectScore + GoodScore + GoodScore

This value is obtained by summing the ETV-compatibilities of the A- and B-literals in the instance. For example, the A-literal `--lungs_ok(mr_s)` in the instance has an expected truth value of `any`. Examining the designation in section 4.4, it is seen that `lungs_ok(mr_s)` is interpreted as TRUE. Thus `--lungs_ok(mr_s)` is interpreted as FALSE. From TABLE 3.13 the literal is assigned the ETV-compatibility value GoodScore. Similarly, the ETV-compatibilities of the two B-literals in the acceptable instance are CorrectScore and GoodScore, respectively.

When designing SGLD, minimisation was chosen for calculating the ETV-compatibility of a centre chain from its ground instances' ETV-compatibilities. In implementing SGLD, however, it was decided that a search through all the acceptable instances of a centre chain would make the semantic guidance system of negative utility. Thus instead of searching for the minimum ETV-compatibility value, an estimate is obtained, as follows. Depending on its expected truth value, a literal can be assigned up to three different ETV-compatibility values as the system backtracks through alternative acceptable instances. In the SGLD implementation the restricted SLE process is controlled so as to look for acceptable instances of literals that are highly ETV-compatible, before less ETV-compatible instances are considered. Once an acceptable instance of a centre chain has been found, the ETV-compatibility value obtained is accepted and no further instances of the centre chain are considered. As the links in the centre chain are considered from right to left, this means that literals to the left in the centre chain may be assigned a non-optimum ETV-compatibility value. Therefore the centre chain may be assigned a non-optimum ETV-compatibility value. This inaccuracy has been considered to be justified by the reduced effort required.

Example

The centre chain in the example above has a second acceptable instance :

```
[b(--alive(spouse_of(mrs_s)),false),
 c(--heart_ok(mrs_s),any,[--lungs_ok(mrs_s)]),
 a(--lungs_ok(mrs_s),any,0),
 b(--heart_ok(mrs_s),any)]
```

with an ETV-compatibility value CorrectScore + BadScore + GoodScore. Thus the ETV-compatibility of the original centre chain is CorrectScore + GoodScore + GoodScore. However, if the second acceptable instance is found before the first, then the non-optimum ETV-compatibility value CorrectScore + BadScore + GoodScore would be assigned to the centre chain.

The possible ETV-compatibility values of a particular literal in a centre chain may change as a deduction progresses, due to the instantiation of variables. For ground literals, however, their ETV-compatibilities are constant. SGLD takes advantage of this by storing the constant ETV-compatibility values within such literals' links. The heuristic function then uses the stored values directly, rather than recalculating them.

5.2.5. Semantic Guidance of Equality Reasoning

In section 2.6 a method of embedding equality into GLD has been presented. The method is based on the generation of equality-demand literals, which form part of the deduced centre chains. If this embedding were to be incorporated into SGLD then the equality-demand literals would contribute to the ETV-compatibility of centre chains. If an equality-demand literal whose arguments are interpreted as unequal were generated, then a bad ETV-compatibility value would be assigned to that literal. The quality of the centre chain would therefore be reduced. Further, equality literals have the potential to be linear-input objects, in which case equality-demand literals would have a FALSE expected truth value. The rightwards subchain system would then reject centre chains containing equality-demand literals whose arguments are interpreted differently.

5.3. The Effects of the Semantic Guidance System

Table 5.1, below, describes the general effects of the semantic guidance system in SGLD, for various combinations of designation and search style. The designations considered are (i) the null designation, which interprets every base element as every truth value, (ii) the positive designation, which interprets all base elements as TRUE and (iii) domain specific designations. The negative designation, which interprets all base elements as FALSE, is

Desig'n	Effects
Literal-selected Search Style	
Null	No effects
Positive	Ensures that the rightwards subchains of top literals in linear-input subdeductions are negative.
Domain specific	Ensures that centre chains are sort legal and that the rightwards subchains of top literals in linear-input subdeductions are FALSE.
Literal-ordered Search Style	
Null	Deduced chains are reordered by length.
Positive	The effects of the positive designation in the literal-selected search style and deduced chains are reordered with a preference for shorter chains with more negative literals.
Domain specific	The effects of a domain specific designation in the literal-selected search style and deduced chains are reordered with a preference for chains which are more ETV-compatible.
Cell-selected Search Style	
Null	A selection rule is used in non-compulsory extension and reduction operations. The selection rule selects the B-literal whose successor set's shortest element is the longest amongst all the successor sets.
Positive	The effects of the positive designation in the literal-selected search style and a selection rule is used in non-compulsory extension and reduction operations. The selection rule selects the B-literal whose successor set's shortest and most negative element is longest and least negative amongst all the successor sets.
Domain specific	The effects of a domain specific designation in the literal-selected search style and a selection rule is used in non-compulsory extension and reduction operations. The selection rule selects the B-literal whose successor set's most ETV-compatible element is the least ETV-compatible amongst all the successor sets.
Cell-ordered Search Style	
Null	The combined effects of the null designation in the literal-ordered search style and the null designation in the cell-selected search style.
Positive	The combined effects of the positive designation in the literal-ordered search style and the positive designation in the cell-selected search style.
Domain specific	The combined effects of a domain specific designation in the literal-ordered search style and the domain specific designation in the cell-selected search style.

Table 5.1 - The Effects of Search Styles and Designationsnot considered in the table. The effects of the negative designation correspond to those of the positive designation, but with truth values and the signs of literals inverted.

The positive, negative and null designations have been used in SGLD's semantic guidance system, to obtain performance figures for SGLD (see section 5.4.1). Table 5.1 shows that the use of any of these designations forms, to all intents and purposes, a syntactic guidance strategy. That is, SGLD runs without semantic guidance. The positive and negative designations produce better results than the null designation. At the same time both the positive and the negative designations can be implemented syntactically, thus very little effort is required to calculate the ETV-compatibility of a centre chain. Therefore, in the performance testing described in section 5.4.1, either the positive or negative designation has been used provided that it is a side chain model of all possible linear-input subdeductions. In the majority of cases the positive designation is acceptable. If the LISS contains both positive and negative elements then neither the positive nor the negative designation is a side chain model of all possible linear-input subdeductions. In these cases the null designation is used. An alternative to using the null designation would be to build a syntactically implementable model which is aware of the linear-input subset. Such a model is possible if the LISS does not contain complementary elements. In such a model LISS atoms are interpreted such that all LISS literals are interpreted as FALSE. All other atoms may be given any interpretation. The implementation can be syntactic because all literals with the same structure are treated equivalently by LISS analysis. The interpretation value of an atom may therefore be specified in terms of its structure.

5.4. Performance

SGLD has been tested on 40 problems. The problems were selected such that they represent a range of problem domains, such that semantic information can be specified for as many problems as possible and such that performance figures from other deduction systems are available for comparison purposes. Many of the problems chosen are from the Wilson and Minker study [1976]. (The problem names used in the Wilson and Minker study are noted in () brackets after the local problem names in Table 5.2.) Appendix 2 supplies statements of the problems and descriptions of the designations used with each. All testing has been performed using the top chain length as the initial bound for the consecutively bounded search.

5.4.1. SGLD Without Semantic Guidance

The first phase of testing illustrates SGLD's performance without the benefit of semantic guidance. A syntactic guidance system is implemented by using the positive, negative or null designation with SGLD's semantic guidance system, as discussed in section 5.3. For most of the problems the positive interpretation has been used so that shorter centre chains with negative literals are preferred. For those problems where the positive designation is not a side chain model of all possible linear-input subdeductions, either the null or negative designation has been used. These cases are indicated with the designation name in {} braces after the problem name. The results of these tests are given in Table 5.2.

Of the 40 test problems, 19 have been solved by SGLD using less than 100 deduction operations (small problems), 11 using between 100 and 1000 deduction operations (medium sized problems) and seven problems have required more than 1000 deduction operations (large problems). There are three problems for which no refutation has been found within the time limit imposed.

The results show that the literal-selected search style is the most effective. This search style produces the best result in 16 small problems, six medium sized problems and three large problems, i.e. it produces the best result in 25 of the 37 solved problems. The literal-ordered search style performs more consistently across problem sizes, producing the best result in three small problems, three medium sized problems and two large problems. The cell-selected and cell-ordered search styles are the poorest performers. They produce the best results in only a few problems. The cell-selected style produces the best result in a single large problem, while the cell-ordered style produces the best result once in each of the medium sized and large categories.

The literal-ordered search style is the most consistent performer. It produces the worst result in only two problems, both of them large. The literal-selected search style produces the worst result in three problems, one small and two medium sized. The cell-selected search style produces the worst results in six small, three medium sized, and three large problems. The cell-ordered search style produces the worst result most often, the distribution being 12 small, six medium sized, and three large problems. It is noteworthy that in most of the problems where the cell-ordered search style produces the worst result, the cell-selected search style uses the same number of deduction operations. However, the cell-ordered search style takes longer to perform those deduction operations. The extra overhead comes from the ordering of alternative successor chains.

Problem	S.	H.	LS		LO		CS		CO		Extremes	
			Size	Tm	Size	Tm	Size	Tm	Size	Tm	Best	W'st
ALGEBRA												
Additive 01 (L&S 28)	13	H	664	23.9	687	26.9	998	36.8	998	37.2	LS	CS
Additive 02 (L&S 29)	13	H	299	10.1	315	11.4	495	16.6	495	16.7	LS	CO
Group 03 (Ch&Lee 3)	5	H	54	1.86	65	2.33	202	6.08	202	6.14	LS	CO
Group 06 (Ch&Lee 6)	9	H	0	0	256	8.10	393	11.9	393	12.0	LO	LS
Group 11 (Wos 8)	18	H	401	10.9	467	13.1	744	20.8	878	25.2	LS	CO
Monoids 01 (Ch&Lee 2)	7	H	2263	73.2	2268	76.2	4554	157	4554	160	LS	CO
Semi-group 01 (Ch&L 1)	4	H	7	0.26	11	0.41	20	0.65	20	0.67	LS	CO
Semi-group 04 (Wos 5)	16	H	0	0	0	0	0	0	0	0	--	--
Sub-group 01 (Wos 12)	21	H	48	1.32	52	1.45	91	2.37	91	2.42	LS	CS
Sub-group 02 (Wos 13)	22	H	88	2.35	101	2.75	206	5.15	206	5.26	LS	CS
Sub-group 03 (Wos 14)	21	H	0	0	0	0	0	0	0	0	--	--
Sub-group 10 (L&S 26)	9	H	544	16.8	417	13.3	728	23.2	728	23.8	LO	CO
ANALYSIS												
IMV Theorem	18	N	2180	72.3	2178	76.0	1024	31.5	750	22.9	CO	LS
NUMBER THEORY												
Primes 01 (Ch&Lee 7)	7	N	27	0.57	35	0.72	44	0.90	44	0.93	LS	CO
Primes 02 (Ch&Lee 8)	9	N	210	5.41	156	4.11	299	7.98	398	11.8	LO	CO
Primes 03 (Ch&Lee 9)	8	N	28	0.91	33	1.02	56	1.44	56	1.45	LS	CO
Primes 04 (L&S 17)	11	N	43	1.35	48	1.49	89	2.22	89	2.24	LS	CO
Rec. func. 01 (L&S 41)	11	H	24	0.56	21	0.48	27	0.55	27	0.58	LO	CO
Rec. func. 05 (L&S 68)	15	H	5	0.17	9	0.22	9	0.22	9	0.23	LS	CO
Rec. func. 10 (L&S 76.1)	16	N	14	0.38	32	0.79	32	0.64	41	0.93	LS	CO
SET THEORY												
Naive Sets 02 (L&S 103)	14	N	66	1.97	81	2.24	136	3.75	135	3.83	LS	CS
Naive Sets 03 (L&S 105)	14	N	31	0.81	45	1.10	52	1.32	56	1.40	LS	CO
Naive Sets 04 (L&S 106)	14	N	31	0.81	45	1.10	52	1.31	56	1.40	LS	CO
Naive Sets 06 (L&S 111)	14	N	35	0.92	45	1.13	69	1.59	65	1.51	LS	CS
Naive Sets 08 (L&S 115)	21	N	82	2.05	91	2.14	200	4.38	170	3.50	LS	CS

Legend

- S. - Number of input clauses in the problem.
- H. - Horn status, either H = Horn or N = non-Horn.
- LS - Literal-selected search style LO - Literal-ordered search style
  CS - Cell-selected search style CO - Cell-ordered search style
- Size - The number of deduction operations performed to find the refutation.
- Tm - The time taken in seconds, to at least three significant digits, to find the refutation. A pair of 0s in the Size and Tm columns means that the system failed to find a refutation within an imposed time limit of 2500 seconds.
- Extremes - Indicates the search styles that produce the best and worst results. The judgement is based firstly on the number of deduction operations performed and secondly on the time taken.

Table 5.2a - SGLD Performance without Semantic Guidance - Maths Problems

Problem	S.	H.	LS		LO		CS		CO		Extremes	
			Size	Tm	Size	Tm	Size	Tm	Size	Tm	Best	W'st
PLANNING												
Getting Bread	16	H	5453	1590	0	0	2562	932	0	0	CS	?O
Going 01	17	N	1383	58.5	1401	61.7	1093	44.4	1085	46.2	CO	LO
Monkey & Banana	11	H	655	19.4	664	20.5	664	20.9	664	21.8	LO	LS
PUZZLES												
Aunt Agatha	12	N	51	1.40	53	1.43	74	1.85	74	1.91	LS	CO
Borders	27	H	36	0.87	47	0.99	170	2.92	170	2.97	LS	CO
Schubert's Steamroller	26	N	10.2k	374	10.2k	387	0	0	30.4k	1111	LS	CS
Truth tellers & Liars	10	N	2141	73.3	1484	52.7	3014	107	2060	72.6	LO	CS
MISCELLANEOUS												
Blind Hand 2 (dbabhp)	14	N	1994	66.3	2031	69.7	2097	73.0	2098	72.5	LS	CO
Blind Hand 3	10	N	0	0	0	0	0	0	0	0	--	--
Compute 2 (burstall)	19	H	111	3.45	133	4.07	177	5.27	177	5.46	LS	CO
Compute 3 {null}	19	N	111	3.53	133	4.09	177	5.32	177	5.45	LS	CO
Has Parts 2	8	N	106	3.32	68	2.13	141	4.84	81	2.61	LO	CS
Latin Squares {negative}	16	N	293k	18.5k	168k	10.8k	298k	18.5k	298k	18.4k	LO	CS
Pigeon 4	22	N	344	11.0	348	11.1	548	17.0	518	16.1	LS	CS
XOR	7	H	80	1.39	71	1.34	77	1.45	71	1.39	LO	LS

Legend

- A value with a k suffix is in thousands of units.
- The limit of 2500 seconds was not imposed in the Latin Squares problem.
- A ? in the Extremes column is a wildcard. For example, ?O means LO and CO achieved the same worst result.

Table 5.2b - SGLD Performance without Semantic Guidance - Other Problems

In summary, for small problems the literal-selected search style is the best, while the literal-ordered style may be preferred for medium sized and large problems. The cell-selected and cell-ordered search styles may be desirable only for (very) large problems. This is a surprising result, as the literal-selected search style provides default search guidance. The reason for its superiority has not been established, but the ordering of the literals in the problems' clauses must be suited to its default search strategy.

In the course of testing SGLD without semantic guidance, an experiment was performed to test the B-literal selection method of the cell-ordered (and hence the cell-selected) search style. The selection method was changed to select the B-literal with the best successor set rather than the worst. This improved the performance in some problems. Notably, the change produced refutations of Semi-group 04 and Sub-group 03. On the other hand, the results for some problems got significantly worse, e.g. the IMV theorem took 18243 operations in a time of 713 seconds. There is thus some potential to affect the performance of GLD by slightly modifying the search strategy. It is unlikely that an optimum configuration can be determined, as the best configuration will be different for different problems. A deduction system that runs several configurations in parallel may produce the

best results. This approach has been taken in the Random Competition system [Ertel, 1991].

5.4.2. A Comparison with Other Deduction Systems

SGLD's results have been compared with results from the Prolog Technology Theorem Prover (PTTP) [Stickel, 1986b], SETHEO [Letz et al., 1992] and the Modified Problem Reduction Format (MPRF) deduction system [Plaisted, 1988]. The comparison is made in Table 5.3. As the implementation environments of the systems vary enormously, a comparison of the times taken by the systems would be of little significance. Thus only the numbers of deduction operations performed have been compared. There are, however, still some factors that may distort the comparison :

- The SETHEO "without preprocessing" results have been used so that the input set sizes are the same as for the other systems. These results are worse than those where the input sets have been preprocessed. On the other hand, the best of SETHEO's six reported results has been used.
- The MPRF results for the default (the better) configuration have been used.
- In many cases, the local versions of the problems list the clauses in a order different from the originals. An examination of the original problems suggests that their clauses have been arranged to the benefit of deduction systems. This suggestion concurs with the comment in the previous section regarding the ordering of literals in clauses. Changing the order of the clauses in the local versions of the problems may have tainted SGLD's results. The PTTP, SETHEO and MPRF have used the original versions of the problems.
- Finally, the measurements taken are in no way standardised. A more accurate comparison of the systems would be possible if a standardised measure were available and each system were run without inter-problem tuning.

SGLD's performance is comparable with that of the other systems. Although SGLD is the best performer in only three of the 24 problems for which a result is available for all systems, it is also the worst performer in only seven of the problems. The three problems in which SGLD performs the best are all non-Horn problems. Of the seven problems in which SGLD performs the worst, five are Horn problems. This suggests that SGLD is, relative to the other three systems, more consistent in non-Horn problems. There are 11 non-Horn problems for which a result is available for all systems. SGLD produces the best result in three of these and the worst in only two. Only the MPRF system performs better in non-Horn problems, producing five best results and only one worst.

Problem	S.	H.	Deduction System				Extremes	
			SGLD	PTTP	Setheo	MPRF	Best	Worst
ALGEBRA								
Additive 01 (L&S 28)	13	H	664	1322	105	1117	Setheo	PTTP
Additive 02 (L&S 29)	13	H	299	1322	105	1117	Setheo	PTTP
Group 03 (Ch&Lee 3)	5	H	54	206	35	29	MPRF	PTTP
Group 06 (Ch&Lee 6)	9	H	256	26	36	157	PTTP	SGLD
Group 11 (Wos 8)	18	H	401	200	21	4120	Setheo	MPRF
Monoids 01 (Ch&Lee 2)	7	H	2236	1589	987	259	MPRF	SGLD
Semi-group 01 (Ch&L 1)	4	H	7	5	5	6	Setheo	SGLD
Semi-group 04 (Wos 5)	16	H	0	795	142	236		
Sub-group 01 (Wos 12)	21	H	48	6	4	36	Setheo	SGLD
Sub-group 02 (Wos 13)	22	H	88	51	53	3969	PTTP	MPRF
Sub-group 03 (Wos 14)	21	H	0	118	34	7208		
Sub-group 10 (L&S 26)	9	H	417	34	45	199	PTTP	SGLD
NUMBER THEORY								
Primes 01 (Ch&Lee 7)	7	N	27	24	8	16	Setheo	SGLD
Primes 02 (Ch&Lee 8)	9	N	156	3104	99	64	MPRF	PTTP
Primes 03 (Ch&Lee 9)	8	N	28	163	138	40	SGLD	PTTP
Primes 04 (L&S 17)	11	N	43	175	1601	76	SGLD	Setheo
Rec. func. 01 (L&S 41)	11	H	21	9	6	80	Setheo	MPRF
Rec. func. 05 (L&S 68)	15	H	5	2	13	291	PTTP	MPRF
Rec. func. 10 (L&S 76.1)	16	N	14	8	4			
SET THEORY								
Naive Sets 02 (L&S 103)	14	N	66	1826	70	131	SGLD	PTTP
Naive Sets 03 (L&S 105)	14	N	31	34	47	5	MPRF	Setheo
Naive Sets 04 (L&S 106)	14	N	31	34	46	5	MPRF	Setheo
Naive Sets 06 (L&S 111)	14	N	35	35	48	5	MPRF	Setheo
Naive Sets 08 (L&S 115)	21	N	82	109	47	207	Setheo	MPRF
PUZZLES								
Schubert's Steamroller	26	N	10.2k		2524	953		
MISCELLANEOUS								
Blind Hand 2 (dbabhp)	14	N	1994	1168	168	190	Setheo	SGLD
Compute 2 (burstall)	19	H	111	690	32	63	Setheo	PTTP
Has Parts 2	8	N	68	87	171	28	MPRF	Setheo

Legend

- The figures record the number of deduction operations performed.
- Blank entries indicate that no results have been reported for that problem.
- The best and worst performer has been noted only when all four systems have reported a result.

Table 5.3 - Performance Comparison : SGLD vs PTTP, SETHEO and MPRF

5.4.3. SGLD With Semantic Guidance

The second phase of testing demonstrates the effects of semantic guidance in SGLD. Domain specific semantic information has been specified for 20 of the problems listed in Table 5.2, using an implementation of algorithm 4.13. Note that semantic information has been specified separately for each problem, i.e., no designation is used for more than one problem. The semantic information has been used by the semantic guidance system in SGLD and the resultant performance figures are given in Table 5.4. No comparison has been made with results produced by other semantically guided deduction systems due to the absence of (results for) such systems. Thus the results that SGLD obtains with semantic guidance can only be compared against those obtained without semantic guidance.

It is not claimed that results like those shown in Table 5.4 can be produced for all problems. Rather, the results illustrate what effects can be produced for problems that are amenable to semantic guidance. It is these results that, in pragmatic terms, establish the thesis of this research.

The results in Table 5.4 show that the literal-selected search style dominates when semantic guidance is used. The literal-selected search style produces the best result in 15 of the 20 problems. The literal-ordered, cell-selected and cell-ordered search styles produce the best results in only one, two and two problems, respectively. The literal-selected search style also performs consistently when semantic guidance is used, producing the worst result in only one problem. The cell selected search style benefits the least from the addition of semantic guidance, producing the worst result in 10 of the 20 problems. The literal-ordered and cell-ordered style produce the worst results in three and six problems respectively.

Of more importance than the relative performances of the search styles, Table 5.4 demonstrates that the addition of semantic guidance significantly improves SGLD's performance. Comparing the best results with and without semantic guidance, there are only two problems in which semantic guidance does not improve SGLD's performance. They are Group 03 and Has Parts 2. In Has Parts 2 the literal-ordered and cell-ordered search styles perform better without semantic guidance. In the other two search styles the semantic guidance has improved performance. Finding a refutation for problem Group 03 has not been affected by the use of semantic guidance. It is noteworthy that Group 03 tries

to establish a general lemma in group theory. As is noted in section 6.6, these types of problems often appear indifferent to the use of semantic guidance. Another problem where the effect of semantic guidance is marginal, is Pigeon 4. Here the literal-selected search style performs the best both with and without semantic guidance. However, the difference in performance is only in the time used. The use of semantic guidance does marginally reduce the number of deduction operations performed in the literal-ordered and cell-ordered search styles, for that problem.

Problem	S.	H.	LS			LO			CS			CO		
			Size	Tm	SDs	Size	Tm	SDs	Size	Tm	SDs	Size	Tm	SDs
Group 03	5	H	54	1.94	0	65	2.43	0	202	6.42	4	202	6.39	4
Group 06	9	H	76	2.55	3	79	2.68	3	70	2.11	2	70	2.09	2
Monoids 01	7	H	1264	47.5	113	1273	48.4	113	2801	107	417	2801	106	417
Rec. Func. 10	16	N	6	0.30	1	10	0.40	2	19	0.52	2	19	0.50	2
Getting Bread D1	16	H	1655	112	683	1901	130	800	1391	58.5	946	1675	71.2	1162
Getting Bread D2	16	H	1655	110	683	1901	129	800	1391	58.2	946	1675	71.5	1162
Going 01	17	N	1383	80.6	0	877	47.2	0	1093	58.3	0	788	40.1	0
Monkey & Banana D1	11	H	628	18.1	38	637	19.1	38	637	19.6	38	637	19.6	38
Monkey & Banana D2	11	H	115	3.09	23	124	3.45	26	124	3.50	26	124	3.53	26
Borders	27	H	9	0.24	4	20	0.37	7	116	1.71	34	116	1.67	34
Schubert's Steamroller D1	26	N	3039	81.5	873	3087	84.4	890	8714	361	3132	8656	358	3110
Schubert's Steamroller D2	26	N	3039	90.4	873	3087	94.1	890	8714	394	3002	8656	383	2981
Blind Hand 3	10	N	239	14.9	84	250	17.5	91	510	119	242	510	119	242
Compute 2 D1	19	H	56	1.82	29	76	2.27	41	109	3.85	56	109	3.80	56
Compute 2 D2	19	H	56	1.84	29	76	2.32	41	109	3.85	56	109	3.90	56
Compute 3	19	N	56	1.90	29	76	2.39	41	109	4.01	56	109	4.06	56
Has Parts 2	8	N	83	4.42	22	83	4.53	22	101	6.24	24	101	6.25	24
Latin Squares	16	N	902k	70.5k	91k	144k	7978	10.5k	265k	14.4k	17.7k	264k	14.3k	17.6k
Pigeon 4	22	N	344	10.2	0	345	10.3	0	548	16.0	0	515	14.5	0
XOR	7	H	40	0.71	15	43	0.77	17	43	0.81	17	43	0.82	17

Legend

- SDs - The number of semantic deletions that occurred within the deduction.
- Best Non-SG - SGLD's best results without semantic guidance.

Table 5.4 - SGLD Performance with Semantic Guidance

A direct comparison between SGLD's performance with and without semantic guidance is made in Table 5.5, below. For each problem the result for the best performing search style with semantic guidance is compared with the corresponding result without semantic guidance. In 15 of the 20 problems listed, the best performing search style with semantic guidance also performs the best without semantic guidance. Thus the comparison is fair in these 15 problems. The five problems that witness a change in the best search style are Group 06, the two versions of Monkey & Banana, Has Parts 2 and XOR. In the two versions of Monkey & Banana and XOR, the literal-selected search style takes over from the literal-ordered search style as the best performer when semantic guidance is added. In these three cases the results without semantic guidance for these two search styles are similar, and thus the comparison made in Table 5.5 is still meaningful. In Group 06 the literal-ordered search style performs the best without semantic guidance, about 35% better than the cell-ordered style. Thus the comparison for this problem is biased by that amount towards semantic guidance. Similarly, in Has Parts 2 the bias is about 36%.

Problem	S.	H.	Best SG			Non-SG		SG/N-SG	
			Size	Tm	By	Size	Tm	Size	Tm
Group 03	5	H	54	1.94	LS	54	1.86	1.00	1.04
Group 06	9	H	70	2.09	CO	393	12.0	0.18	0.17
Monoids 01	7	H	1264	47.5	LS	2263	73.2	0.56	0.65
Rec. Func. 10	16	N	6	0.30	LS	14	0.38	0.43	0.78
Getting Bread D1	16	H	1391	58.5	CS	2562	932	0.54	0.06
Getting Bread D2	16	H	1391	58.2	CS	2562	932	0.54	0.06
Going 01	17	N	788	40.1	CO	1085	46.2	0.73	0.87
Monkey & Banana D1	11	H	628	18.1	LS	655	19.4	0.96	0.93
Monkey & Banana D2	11	H	115	3.09	LS	655	19.4	0.18	0.16
Borders	27	H	9	0.24	LS	36	0.87	0.25	0.28
Schubert's Steamroller D1	26	N	3039	81.5	LS	10.2k	374	0.30	0.22
Schubert's Steamroller D2	26	N	3039	90.4	LS	10.2k	374	0.30	0.24
Blind Hand 3	10	N	239	14.9	LS	0	0	0.00	0.00
Compute 2 D1	19	H	56	1.82	LS	111	3.45	0.50	0.53
Compute 2 D2	19	H	56	1.84	LS	111	3.45	0.50	0.53
Compute 3	19	N	56	1.90	LS	111	3.53	0.50	0.54
Has Parts 2	8	N	83	4.42	LS	106	3.32	1.22	1.33
Latin Squares	16	N	144k	7978	LO	168k	10.8k	0.86	0.74
Pigeon 4	22	N	344	10.2	LS	344	11.0	1.00	0.93
XOR	7	H	40	0.71	LS	80	1.39	0.50	0.51

Legend

- Best SG - The best result for SGLD with semantic guidance. The search style that produced the result is noted in the "By" column.
- Non-SG - The results for SGLD without semantic guidance, for the search style in the "By" column.
- SG/N-SG - The ratio of the results with semantic guidance to the results without semantic guidance. Thus a value less than one indicates improved performance.

Table 5.5. - Summary of the Improved Performance due to Semantic Guidance

The last two columns in Table 5.5 clearly indicate the utility of using semantic guidance. The average size ratio is 0.55 and the average time ratio is 0.53. The better time ratio is not what is intuitively expected. In most applications of semantic guidance the overhead of processing the semantic information would make the time ratio larger than the deduction operations ratio. The cause of the anomaly in SGLD is the relatively large number of lemmas that are created when semantic guidance is absent. Processing lemmas is fairly slow in SGLD. In particular, adding lemmas to the input set means adding them to the Prolog database. This is a relatively slow operation in Prolog.

The Effects of Semantic Deletion

An examination of the designations used in this testing reveals that there are 17 problems that are affected by one or both of the rightwards subchain system and sort value deletion. Of the 17, there are 10 problems which are affected by only the rightwards subchain system. They are Group 06, Monoids 01, Rec. Func. 10, Getting Bread D2, Schubert's Steamroller D2, Compute 2 D2, Compute 3, Has Parts 2, Latin Squares and XOR. It is noteworthy that five of these are non-Horn problems for which truth value semantic deletion has improved performance. This demonstrates the usefulness of linear-input subset analysis and the rightwards subchain system.

Example

An example, illustrating the effects of the rightwards subchain system, is to be found in appendix 1, section A1.3. The example traces the start of an SGLD deduction for the Schubert's Steamroller problem, with the rightwards subchain system using the second designation provided for that problem (see appendix 2). This example corresponds to the lines labelled "Schubert's Steamroller D2" in tables 5.4 and 5.5.

There are six problems which are affected by both the rightwards subchain system and sort value deletion. They are Getting Bread D1, Monkey & Banana D1 and D2, Borders, Blind Hand 3 and Compute 2 D1. Schubert's Steamroller D1 is the only problem that is affected solely by sort value deletion.

The large proportion of problems that are affected by semantic deletion is consistent with the dominance of the literal-selected search style in these tests. The FALSE-preference strategy has no effect in the literal-selected search style, so only semantic deletion can improve its performance. In general, the results show that both the rightwards subchain system and sort value deletion play an important role in improving the performance of SGLD. The effort of combining these two forms of semantic deletion into a sort&truth value deletion system is warranted.

Although semantic deletion has been found to usually improve the performance of SGLD, some pathological cases have been found in which semantic deletion degrades

performance. The cause of this behaviour is the effect of lemmas within the consecutively bounded search. When performing a deduction without semantic deletion, SGLD searches portions of the search space which are not reached when semantic deletion is active. In such searches lemmas may be generated, even though a refutation cannot be found. Then, upon entering a portion of the search space that does contain a refutation, the lemmas are used in finding a refutation within the current search bound. In deductions guided by semantic deletion the lemmas are not created and the equivalent deduction must be done on the path to a refutation. This can cause the search bound to be exceeded, resulting in another iteration of the consecutively bounded search being necessary.

In some problems, particularly simpler ones, semantic deletion of a centre chain only slightly preempts a natural termination of that branch of the search. As a semantically deleted centre chain cannot lead to a refutation this is not surprising. In these cases the search subtree under such a centre chain is small. Depending on the size of the subtree it may be of greater utility to allow the search to terminate naturally. In complex problems such search subtrees are typically large, and semantic deletion is most likely to be of utility.

The Effect of the FALSE-Preference Strategy

It is hard to accurately judge the effect of the FALSE-preference strategy in SGLD from the results, as the semantic deletion effects overawe those of the FALSE-preference strategy. There are two problems in which the effect of the FALSE-preference strategy can be observed. They are Going 01 and Pigeon 4. In Going 01 the FALSE-preference strategy makes a significant improvement. In Pigeon 4 the FALSE-preference strategy marginally reduces the number of derivation operations performed. Some effort has been expended trying to find more problems in which the FALSE-preference strategy has a distinct effect. These efforts have been unsuccessful.

In the light of the above, the question that arises naturally is whether or not the FALSE-preference strategy is in fact appropriate. In an effort to answer this question, a TRUE-preference strategy has been investigated. Limited experimentation with the TRUE-preference strategy has indicated that it degrades the performance of SGLD. Thus the FALSE-preference strategy appears to be 'in the right direction'. However, further development may be necessary. The following observation suggests one possible development. Literals that are TRUE in side chain models of a deduction are likely to be reduced against. The FALSE-preference strategy avoids centre chains that contain TRUE literals. If all refutations of an input set contain a reduction operation (not unlikely in non-Horn problems), then the FALSE-preference strategy may guide the host deduction system away from refutations. Further analysis of the input set, in the style of linear-input

subset analysis, may be possible to determine for which literals a FALSE-preference should be shown.

5.5. Conclusion

This chapter has described the implementation and testing of SGLD - the combination of GLD, a semantic guidance system and designations. The contribution made by this chapter is to illustrate the effects that semantic guidance can have in a linear deduction system.

The main value of implementing SGLD is that the implementation has facilitated evaluation of GLD, the semantic guidance system, designations and their combination. The following are the conclusions :

- GLD is an effective deduction system. Without the aid of semantic guidance its performance is comparable with that of other well regarded deduction systems.
- The rightwards subchain system and sort value deletion can significantly improve the performance of SGLD.
- The FALSE-preference strategy may require further development.
- Designations are well suited to the task of supplying semantic to semantic guidance systems. The (implementation of the) designation building algorithm has been particularly useful.
- The combination of GLD, the semantic guidance system and designations, forms a coherently integrated deduction system.

SGLD's Prolog implementation is believed to be, as Prolog implementations go, fairly efficient. A more efficient implementation of SGLD could be achieved by compiling the input set to a Prolog program which implements SGLD deductions from the input set. This is the approach taken in the MPRF system. An even faster implementation could be achieved by compiling to an executable form, as is done in the PTTP.

All of the features in SGLD have been developed cognisant of each other and this has made their combination into a coherent whole possible. An interesting feature of the implementation is the use of an auxiliary data structure - the semantic chain - to maintain deduction faithfulness. This has facilitated an efficient implementation of the semantic guidance system. The admissibility restrictions in GLD are operation restrictions, so the issue of deduction faithfulness does not arise there. However, if those restrictions were to be extended to be deduction restrictions, a second auxiliary data structure - a 'syntactic chain' - could be used. As the current admissibility restrictions already have retrospective effect, the amount of information that would have to be stored in such a data structure would be fairly small.

Chapter Six

Conclusion

This chapter reviews the outcomes of this research. SGLD has combined GLD, a semantic guidance system and designations, to form a unique deduction system. The components of SGLD are individually of interest and their combination into SGLD has confirmed the thesis of this research. Areas worthy of further investigation have also been noted.

This chapter contains :

1. An overview of the work done.
2. Discussion of GLD.
3. Discussion of the semantic guidance systems developed.
4. Discussion of designations.
5. Discussion of SGLD.
6. Concluding comments.

6.1. Overview

The research described in the preceding chapters started with the thesis given in chapter 1 :

> **Semantic guidance can be used to improve the performance of a linear deduction system.**

It is noted there that four subobjectives would contribute to establishing this thesis. They are as follows. (i) To develop a host deduction system. (ii) To develop a semantic guidance system for the host deduction system. (iii) To develop an interpretive structure for storing the semantic information used by the semantic guidance system. (iv) To combine the deduction system, semantic guidance system and the interpretive structure into a coherent whole. Each of these subobjectives has been satisfied. (i) The host deduction system developed is GLD. (ii) Several semantics guidance systems, most notably combined semantic guidance, have been developed for GLD. (iii) Designations store the semantic

information used. (iv) GLD, a combined semantic guidance system and designations, have been combined into SGLD.

6.2. GLD

Although GLD has been designed as a platform for using and testing semantic guidance in linear deduction systems, it also has features that make it an interesting deduction system in its own right.

The most significant idea associated with GLD is linear-input subset analysis. The important consequence of linear-input subset analysis is that a truth value deletion system - the rightwards subchain system - can be used in (the linear-input) parts of GLD deductions. A secondary payoff of the analysis is that the reduction operation can be explicitly ignored in linear-input subdeductions. A small number of non-Horn problems with non-trivial linear-input subsets have been identified in this research. It would be desirable to find a large number of such problems, as this would further establish the pragmatic importance of linear-input subset analysis.

There are four noteworthy features within GLD. Firstly, GLD has explicit search guidance mechanisms and an explicit entry point for the incorporation of guidance systems. As a result, the potential for search guidance in GLD is higher than in previous comparable systems. Secondly, GLD employs deduction chunks. Thirdly, GLD's combined lemma/C-literal mechanism improves upon previous mechanisms for reusing deduced information. The combined lemma/C-literal mechanism can be used in chain format linear deduction systems other than GLD and is thus a useful, general, development. Finally, the extended admissibility restrictions imposed in GLD are important in terms pruning the search tree. The admissibility restrictions are operational ones, but have a fairly high level of retrospective and prospective effect. As is noted in section 5.5, the extension of the admissibility restrictions to a deductional nature is possible.

The embedding of equality into GLD has been examined superficially. The basic approach appears to have potential, particularly as it is naturally controlled by GLD's search guidance mechanisms. Its full development, evaluation and exploitation, are areas for further research.

The dynamic definition of GLD contrasts with the presentations of many other systems. This form of definition makes exact implementation possible. With an exact implementation, it is possible to examine closely the intrinsic properties of GLD. Such examination could of course lead to further improvements.

6.3. Semantic Guidance

Eight semantic guidance systems have been defined in this research. Three are specifically for linear-input deduction systems. The remaining five have been designed with linear deduction systems in mind, but can also be used with other deduction formats. A useful feature of the definitions of these guidance systems, is that implementational issues have been considered. This means that they can be implemented and used without further inquiry being necessary.

The rightwards subchain truth value deletion system is the most important of the guidance systems developed. The rightwards subchain system is complete for linear deduction systems. As truth value deletion has previously been considered incompatible with linear deduction, the rightwards subchain system is a significant development. Building on the rightwards subchain system, a coherent suite of semantic guidance systems has emerged. With the reformulation of sort value deletion into the same terms as truth value deletion and the tempering of truth value deletion into the FALSE-preference strategy, it has been possible to formulate new semantic guidance systems which are widely applicable. As the use of semantic guidance has been seen to be a neglected area, these developments are of interest and use.

There is potential to further develop combined semantic guidance systems. At least three questions need to be addressed. (i) Is it possible to make the preference for FALSE literals more selective? (This is the most important of these three questions.) The results presented in chapter 5 indicate that further tuning of the FALSE-preference strategy may be necessary. As discussed in section 5.4.3, it may not always be appropriate to avoid centre chains that contain TRUE literals. (ii) How may ETV-compatibility levels should be available to be assigned to ground literals, and what values should be assigned to each of the levels? This research uses four of the eight possible levels and the integer values used in SGLD are based on rough empirical evidence. (iii) What ETV-compatibility functions are appropriate for determining the ETV-compatibility of a chain and of its ground instances? Minimisation and summation have been used in SGLD, but other functions may produce better results.

Plaisted [1990b] has suggested another way of semantically guiding the reduction operation. The positive refinement for Model Elimination style deduction systems [Plaisted 1990b] shows that it is necessary to reduce only against A-literals that are interpreted as FALSE. The effect of imposing such a restriction is worth investigating.

Finally, it is known that multiple truth value interpretations can be used for truth value deletion in linear-input deductions [Brown, 1973]. It should be possible to extend the semantic guidance systems developed in this research to use multiple interpretations. This is also an area for further research.

6.4. Designations

The development of designations has been motivated by the complexity and volume of semantic information used by semantic guidance systems. An important feature of designations, that makes them superior to existing SRI structures, is their use of property inheritance. Designations are also capable of storing more complex semantic information than standard SRI structures. This added capability is due to the relationship (rather than a mapping) between domain elements. This latter facility has, however, been used in only a minority of the designations built in the course of this research. Another important feature of designations is their compatibility with the SLE process.

The designation building algorithm is probably the unsung hero of this research. It has been indispensable for the specification of the designations used with SGLD. The algorithm makes it possible for a user to concentrate on the accurate and appropriate supply of semantic information, without having to be concerned about the completeness and soundness of the interpretations built.

Two topics for further research are evident in this area. (i) This research supplies (in section 4.7.2) only very rough indications of what properties make an interpretation effective in a semantic guidance system. Accurate analysis of interpretations' properties, in terms of semantic guidance, is needed. (ii) Only a low level of automation has been achieved in the specification of designations. Further automation, using techniques such as those mentioned in section 3.2, would be desirable.

6.5. SGLD

SGLD has been developed to test the thesis of this research, and it has been successful in this task. The basic capability of GLD has been illustrated in performance testing without semantic guidance. The positive effects of the rightwards subchain system and sort value deletion have been demonstrated in the performance testing with semantic guidance. This testing also indicated the potential of the FALSE-preference strategy. Designations have shown themselves to be effective for storing semantic information. The prime feature of

SGLD is its combined semantic guidance system. The consistent use of the same semantic information in all of its components has resulted in coherent guidance of SGLD's search.

It is worth considering the quality of SGLD as a whole. Plaisted [1990a] gives four criteria for evaluating deduction systems. They are (in essence) :

1. Does the system support back chaining with reuse of deduced information?
2. Does the system use a genuine support strategy, that concentrates on chains deduced from the negation of the conclusion to be proved?
3. Does the system permit the use of semantic information to reject redundant deduced chains?
4. Is the use of resolution well controlled during back chaining?

To these questions, SGLD provides the following answers :

1. For the first part, linear deduction systems are, by definition, back chaining systems. For the second part, the lemma/C-literal mechanism in SGLD implements the reuse of deduced information.
2. SGLD's preference for input chains whose status is `theorem`, allows the user to ensure that a genuine support strategy is used.
3. The rightwards subchain and sort value deletion systems, in SGLD's semantic guidance system, reject redundant deduced chains.
4. The use of resolution is intrinsically well controlled in linear deduction systems. SGLD improves on this via its compulsory operations and admissibility checking. The four search styles provide different ways of directing SGLD's search for a refutation.

These answers indicate that SGLD is (or at least should be) a high quality deduction system. It is hard to make any overall comparisons between SGLD and other deduction systems, due to the absence of other deduction systems which which make such extensive use of semantic guidance. Possibly the only valid comparison that can be made is with Sandford's Hereditary Lock Resolution (HLR) [1980] (see section 3.2.1). HLR appears to be the best semantically guided deduction system to date, with a strong theoretical basis for its semantic guidance. In terms of Plaisted's four criteria, HLR is not a back chaining system, nor does it use a genuine support strategy. It does use semantic deletion and, as a version of lock resolution, has a well controlled use of resolution. A similarity between SGLD and HLR is the use of an auxiliary data structure to maintain deduction faithfulness of the semantic guidance restrictions.

Beyond the potential for faster implementation, as mentioned in section 5.5, future work on SGLD will certainly arise out of any changes to GLD, the semantic guidance system, or designations. It would also be interesting to investigate to what extent a single designation

can be used to guide deductions for multiple input sets. For a single designation to be used with multiple input sets, the semantic information stored would have to be fairly general. This may make the semantic guidance systems less effective.

6.6. Conclusion

The ideas and results presented in the preceding chapters show that the thesis of this research holds. Semantic guidance can be used to improve the performance of a linear deduction system. Only two issues remain slightly open in this regard. Firstly, it would be desirable to uncover a large group of non-Horn problems which have non-trivial linear-input subsets. This would more firmly establish the pragmatic importance of linear-input subset analysis and the rightwards subchain system. Secondly, the FALSE-preference strategy may need further tuning.

The outcomes of this research have the potential to contribute to other deduction systems. The FALSE-preference strategy, sort&truth value deletion and combined semantic guidance, are generally applicable semantic guidance mechanisms. Designations (and the algorithm for building them) make it far easier to use these and other semantic guidance systems in appropriate host deduction systems.

Beyond the specific areas for further research mentioned in the earlier sections of this chapter, two general issues have stood out as warranting further attention, as follows. (i) Although complete truth value deletion systems now exist for linear and linear-input deduction systems, use of these systems has indicated that they have effect only in some types of problems. For example, truth value deletion appears to be ineffective when proving general lemmas from the axioms of a problem domain. On the other hand, truth value deletion appears particularly effective in problems that have some special hypotheses. Some theoretical analysis should reveal generic problem types for which truth value deletion is (and is not) appropriate. (ii) The technique of proof by analogy to a proof in the semantic domain needs to be developed. Some work has been done in this area [Plaisted, 1981, 1984]. However, when compared to its potential, this approach to semantic guidance appears to have been neglected. This thought has been voiced by others, e.g. "Somehow intelligent machines (including reasoners) must make use of analogy ..." [Bledsoe & Hodges, 1988, p. 517]. The domain based approach described in chapter 3 could form a starting point for research in this area.

References

Anderson R. and Bledsoe W.W. (1970), A Linear Format for Resolution with Merging and a New Technique for Establishing Completeness, In *Journal of the ACM* 17(3), ACM Press, New York, NY, 525-534.

Andrews P.B. (1968), Resolution with Merging, In *Journal of the ACM* 15(3), ACM Press, New York, NY, 367-381.

Arity Corporation (1988), *The Arity/Prolog Language Reference Manual*, Arity Corporation, Concord, MA.

Astrachan O.L., and Stickel M.E. (1992), Caching and Lemmaizing in Model Elimination Theorem Provers, In Kapur, D. (Ed.), *Proceedings of the 11th International Conference on Automated Deduction* (Saratoga Springs, NY, 1992), (Lecture Notes in Artificial Intelligence 607), Springer-Verlag, New York, NY, 224-238.

Ballantyne A.M. and Bennett W. (1973), Graphing Methods for Topological Proofs, Research Report ATP 7, Department of Computer Science, University of Texas at Austin, Austin, TX.

Ballantyne A.M., and Bledsoe W.W. (1977), Automatic Proofs of Theorems in Analysis Using Nonstandard Techniques, In *Journal of the ACM* 24(3), ACM Press, New York, NY, 353-374.

Ballantyne A.M. and Bledsoe W.W. (1982), On generating and using examples in proof discovery, In Hayes J.E., Michie D. (Ed.), *Machine Intelligence* 10, Ellis-Horwood, Chichester, England, 3-39.

Bibel W. (1987), *Automated Theorem Proving*, Vieweg & Sohn, Braunschweig, Germany.

Bledsoe W.W. (1983), Using examples to generate instantiations of set variables, In Bundy A. (Ed.), *Proceedings of the 8th International Joint Conference on Artificial Intelligence* (Karlsruhe, Germany, 1983), International Joint Conferences on Artificial Intelligence Inc, Los Altos, CA, 892-901.

Bledsoe W.W. (1986), Some Thoughts on Proof Discovery, In *Proceedings of the 3rd Symposium on Logic Programming* (Salt Lake City, UT, 1986), IEEE Computer Society Press, Washington, DC, 2-10.

Bledsoe W.W. and Henschen L.J. (1985), An Overview of Automated Reasoning : What is Automated Theorem Proving?, In *Journal of Automated Reasoning* 1(1), Kluwer Academic Publishers, Dordrecht, The Netherlands, 5-48.

Bledsoe W.W. and Hodges R. (1988), A Survey of Automated Deduction, In Schrobe H.E. (Ed.), *Exploring Artificial Intelligence : Survey Talks from the National Conferences on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA, 483-543.

Bledsoe W.W. (1992), Personal Correspondence.

Boyer R.S. (1971), Locking : a restriction of resolution, PhD Thesis, University of Texas at Austin, Austin, TX.

Brown F.M. (1973), The Use of Several Models as a Refinement of Resolution with sets of Horn Clauses, Internal Memo #63, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland.

Brown F.M. (1974), SLM, Internal Memo #72, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland.

Bundy A. (1983), *The Computer Modelling of Mathematical Reasoning*, Academic Press, London, England.

Bundy A. (1984), A Generalized Interval Package and Its Use for Semantic Checking, In *ACM Transactions on Mathematical Systems* 10(4), ACM Press, New York, NY, 397-409.

Bundy A. (1987), Personal Correspondence.

Bundy A., Byrd L., Luger G., Mellish C., Milne R. and Palmer M. (1979), Solving Mechanics Problems Using Meta-level Inference, In *Proceedings of the 6th International Joint Conference on Artificial Intelligence* (Tokyo, Japan, 1979), International Joint Conferences on Artificial Intelligence Inc, Los Altos, CA, 1017-1027.

Carlsson M. and Widen J. (1990), SICStus Prolog User's Manual, R88007C, Swedish Institute of Computer Science, Kista, Sweden.

Chang C-L. (1970), The Unit Proof and the Input Proof in Theorem Proving, In *Journal of the ACM* 17(4), ACM Press, New York, NY, 698-707.

Chang C-L. (1972), The Decomposition Principle for Theorem Proving Systems, In *Proceedings of the 10th Annual Allerton Conference on Circuit and System Theory* (Urbana, IL, 1972), The Conference, Urbana, IL, 20-28.

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

Cohn A.G. (1987), A More Expressive Formulation of Many Sorted Logic, In *Journal of Automated Reasoning* 3(2), Kluwer Academic Publishers, Dordrecht, The Netherlands, 113-200.

Delgrande J.P. and Mylopolous J. (1986), Knowledge Representation : Features of Knowledge, In Bibel W., Jorrand Ph (Eds.), *In Fundamentals of Artificial Intelligence : An Advanced Course*, (Lecture Notes in Computer Science 232), Springer-Verlag, New York, NY, 3-36.

Digricoli V.J. (1979), Automatic Deduction and Equality, In Martin A.L. (Ed.), *Proceedings of the Annual Conference of the ACM* (Detroit, MI, 1979), ACM Press, New York, NY, 240-250.

Dougherty D.J. and Johann P. (1990), An Improved General E-Unification Method, In Stickel M. (Ed.), *Proceedings of the 10th International Conference on Automated Deduction* (Kaiserslautern, Germany, 1990), (Lecture Notes in Artificial Intelligence 449), Springer-Verlag, New York, NY, 261-275.

Enderton H.B. (1972), *A Mathematical Introduction to Logic*, Academic Press, New York, NY.

Ertel W. (1991), Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems, In Kanal L.N., Suttner C. B. (Eds.), *Informal Proceedings of PPAI-91, International Workshop on Parallel Processing for Artificial Intelligence* (Sydney, Australia, 1991), International Joint Conferences on Artificial Intelligence Inc, Sydney, Australia, 36-39.

Fleisig S., Loveland D.W., Smiley A.K. and Yarmush D.L. (1974), An Implementation of the Model Elimination Proof Procedure, In *Journal of the ACM* 21(1), ACM Press, New York, NY, 124-139.

Fruhwirth T.W. (1989), A Type Language for Prolog and its Application to Type Inference, In Martelli A., Valle G. (Eds.), *Computational Intelligence* 1, Elsevier Science Publishers, Amsterdam, The Netherlands, 29-41.

Gallier J. and Snyder W. (1989), Complete Sets of Transformations for General E-Unification, In *Theoretical Computer Science* 67(2,3), North-Holland, Amsterdam, The Netherlands, 203-260.

Gelerneter H. (1963), Realisation of a Geometry-Theorem Proving Machine, In Feigenbaum E.A., Feldman J (Eds.), *Computers and Thought*, McGraw-Hill, New York, NY, 134-152.

Gelerneter H., Hansen J.R. and Loveland D.W. (1963), Empirical Explorations of the Geometry-Theorem Proving Machine, In Feigenbaum E.A., Feldman J (Eds.), *Computers and Thought*, McGraw-Hill, New York, NY, 153-163.

Ginsberg M.L. and Geddis D.F. (1991), Is there any Need for Domain-Dependent Control Information?, In Dean T., McKeown K. (Eds.), *AAAI-91, Proceedings of the 9th National Conference on Artificial Intelligence* (Asilomar, CA, 1991), AAAI Press/MIT Press, Menlo Park, CA, 452-457.

Hayes P.J. (1971), A Logic of Actions, In Meltzer B., Michie D. (Eds.), *Machine Intelligence* 6, Edinburgh University Press, Edinburgh, Scotland, 495-520.

Henschen L.J. (1972), N-Sorted Logic for Automatic Theorem-Proving in Higher Order Logic, In *Proceedings of the Annual Conference of the ACM* (Boston, MA, 1972), ACM Press, New York, NY, 71-81.

Henschen L.J. (1976), Semantic Resolution for Horn Sets, In *IEEE Transactions on Computers* C-25(8), IEEE Computer Society Press, Washington, DC, 816-822.

Henschen L.J. and Wos L. (1974), Unit Refutations and Horn Sets, In *Journal of the ACM* 21(4), ACM Press, New York, NY, 590-605.

Irani K.B. and Shin D.G. (1985), A Many-Sorted Resolution based on an Extension of a First-Order Language, In Joshi A. (Ed.), *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (Los Angeles, CA, 1985), International Joint Conferences on Artificial Intelligence Inc, Los Altos, CA, 1175-1177.

Jaffar J. and Lassez J-L. (1987), Constraint Logic Programming, In *Proceedings of the Annual ACM Symposium on Principles of Programming Languages* (Munich, Germany, 1987), ACM Press, Baltimore, MD, 0-15.

Kim M.W. (1986), On Automatically Generating and Using Examples in a Computational Logic System, The University of Texas at Austin, Austin, TX.

Knuth D.E. and Bendix P.B. (1970), Simple word problems in universal algebras, In Leech J. (Ed.), *Computational Problems in Abstract Algebras*, Pergamon Press, 263-297.

Korf R.E. (1985), Depth-First Iterative Deepening: An Optimal Admissible Tree Search, In *Artificial Intelligence* 27, Elsevier Science, Amsterdam, The Netherlands, 97-109.

Kornfield W.A. (1983), Equality for Prolog, In Joshi A. (Ed.), *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (Los Angeles, CA, 1983), International Joint Conferences on Artificial Intelligence Inc, Los Altos, CA, 514-519.

Kowalski R.A. (1970), Studies in the Completeness and Efficiency of Theorem-Proving by Resolution, PhD Thesis, University of Edinburgh, Edinburgh, Scotland.

Kowalski R.A. and Hayes P.J. (1969), Semantic Trees in Automatic Theorem Proving, In Meltzer B., Michie D. (Eds.), *Machine Intelligence* 4, Edinburgh University Press, Edinburgh, Scotland, 87-101.

Kowalski R.A. and Kuehner D. (1971), Linear Resolution with Selection Function, In *Artificial Intelligence* 2, Elsevier Science, Amsterdam, The Netherlands, 227-260.

Kuehner D. (1972), Some Special Purpose Resolution Systems, In Meltzer B., Michie D (Eds.), *Machine Intelligence* 7, Edinburgh University Press, Edinburgh, Scotland, 117-128.

Lawrence J.D. and Starkey J.D. (1974), Experimental tests of resolution based theorem-proving strategies., Technical Report, Computer Science Department, Washington State University, Pullman, WA.

Letz R., Schumann J., Bayerl S. and Bibel W. (1992), SETHEO: A High-Performance Theorem Prover, In *Journal of Automated Reasoning* 8(2), Kluwer Academic Publishers, Dordrecht, The Netherlands, 183-212.

Lloyd J.W. (1984), *Foundations of logic programming*, Springer-Verlag, New York, NY.

Loveland D.W. (1968), Mechanical Theorem Proving by Model Elimination, In *Journal of the ACM* 15(2), ACM Press, New York, NY, 236-251.

Loveland D.W. (1969a), A Simplified Format for the Model Elimination Theorem-Proving Procedure, In *Journal of the ACM* 16(3), ACM Press, New York, NY, 349-363.

Loveland D.W. (1969b), Theorem-provers Combining Model Elimination and Resolution, In Meltzer B., Michie D. (Eds.), *Machine Intelligence* 4, Edinburgh University Press, Edinburgh, Scotland, 73-86.

Loveland D.W. (1970), A Linear Format for Resolution, In Laudet M. et al. (Eds.), *Proceedings of the IRIA Symposium on Automatic Demonstration* (Versailles, France, 1968), Springer-Verlag, New York, NY, 147-162.

Loveland D.W. (1972), A Unifying View of Some Linear Herbrand Procedures, In *Journal of the ACM* 19(2), ACM Press, New York, NY, 366-384.

Loveland D.W. (1978), *Automated Theorem Proving : a logical basis*, Elsevier Science, Amsterdam, The Netherlands.

Luckham D. (1968), Some Tree-paring Strategies for Theorem Proving, In Michie D. (Ed.), *Machine Intelligence* 3, Edinburgh University Press, Edinburgh, Scotland, 95-112.

Luckham D. (1970), Refinement Theorems in Resolution Theory, In Laudet M. et al. (Eds.), *Proceedings of the Symposium on Automatic Demonstration* (Versailles, France, 1968), Springer-Verlag, New York, NY, 163-190.

Lusk E. and Overbeek R. (1985), Non-Horn Problems, In *Journal of Automated Reasoning* 1(1), Kluwer Academic Publishers, Dordrecht, The Netherlands, 103-114.

Manna Z. and Waldinger R. (1985), Special Relations in Automated Deduction, Internal Report STAN-CS-85-1051, Department of Computer Science, Stanford University, Stanford, CA.

Manthey R. and Bry F. (1988), SATCHMO: a theorem prover implemented in Prolog, In Lusk E., Overbeek R. (Eds.), *Proceedings of the 9th International Conference on Automated Deduction* (Argonne, IL, 1988), (Lecture Notes in Computer Science 310), Springer-Verlag, New York, NY, 415-434.

McCune W.W. (1990), Skolem Functions and Equality in Automated Deduction, In Dietterich T., Swartout W. (Eds.), *Proceedings of the 8th National Conference on Artificial Intelligence* (Boston, MA, 1990), American Association for Artificial Intelligence / MIT Press, Menlo Park, CA, 246-252.

McCune W.W. and Henschen L.J. (1983), Semantic Paramodulation for Horn Sets, In Bundy A. (Ed.), *Proceedings of the 8th International Joint Conference on Artificial Intelligence* (Karlsruhe, Germany, 1983), International Joint Conferences on Artificial Intelligence Inc., Los Altos, CA, 902-908.

McRobbie M.A., Meyer R.K. and Thistlewaite P.B. (1988), Towards Efficient "Knowledge-Based" Automated Theorem Proving for Non-Standard Logics, In Lusk E., Overbeek R. (Eds.), *Proceedings of the 9th International Conference on Automated Deduction* (Argonne, IL, 1988), (Lecture Notes in Computer Science 310), Springer-Verlag, New York, NY, 197-217.

Meltzer B. (1966), Theorem-proving for computers: Some results on resolution and renaming, In *The Computer Journal* 8, The British Computer Society, London, England, 341-343.

Michie D., Ross R. and Shannan G.J. (1972), G-deduction, In Meltzer B., Michie D. (Eds.), *Machine Intelligence* 7, Edinburgh University Press, Edinburgh, Scotland, 141-165.

Minker J. and Zanon G. (1982), An Extension to Linear Resolution with Selection Function, In *Information Processing Letters* 14(4), Elsevier Science, Amsterdam, The Netherlands, 191-194.

Morris J.B. (1969), E-Resolution: Extension of Resolution to include the equality relation, In Walker D.E., Norton L.M. (Eds.), *Proceedings of the 1st International Joint Conference on Artificial Intelligence* (Washington, DC, 1969), Mitre Corp., Bedford, MA, 287-294.

Mycroft A. and O'Keefe R.A. (1984), A Polymorphic Type System for Prolog, In *Artificial Intelligence* 23, Elsevier Science, Amsterdam, The Netherlands, 295-307.

Naish L. (1985), muProlog 3.2 Reference Manual, Technical Report 85/11, Department of Computer Science, University of Melbourne, Melbourne, Australia.

Naish L. (1986), *Negation and Control in Prolog*, (Lecture Notes in Computer Science 238), Springer-Verlag, New York, NY.

Nevins A.J. (1975), Plane Geometry Theorem Proving Using Forward Chaining, In *Artificial Intelligence* 6, Elsevier Science, Amsterdam, The Netherlands, 1-23.

Newell A. and Simon H.A. (1976), Computer Science as Empirical Inquiry : Symbols and Search, In *Communications of the ACM* 19(3), ACM Press, New York, NY, 113-126.

Nie X. and Plaisted D.A. (1990), A Complete Semantic Back Chaining Proof System, In Stickel M. (Ed.), *Proceedings of the 10th International Conference on Automated Deduction* (Kaiserslautern, Germany, 1990), (Lecture Notes in Artificial Intelligence 449), Springer-Verlag, New York, NY, 16-27.

Nilsson N.J. (1971), *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, NY.

Overbeek R., McCharen J. and Wos L. (1976), Complexity and Related Enhancements for Automated Theorem-Proving Programs, In *Computers and Mathematics with Applications* 2, Pergamon Press, England, 1-16.

Pastre D. (1978), Automatic Theorem Proving in Set Theory, In *Artificial Intelligence* 10, Elsevier Science, Amsterdam, The Netherlands, 1-27.

Pelletier F.J. (1986), Seventy-five Problems for Testing Automatic Theorem Provers, In *Journal of Automated Reasoning* 2(2), Kluwer Academic Publishers, Dordrecht, The Netherlands, 191-216.

Plaisted D.A. (1981), Theorem Proving with Abstraction, In *Artificial Intelligence* 16, Elsevier Science, Amsterdam, The Netherlands, 47-108.

Plaisted D.A. (1982), A Simplified Problem Reduction Format, In *Artificial Intelligence* 18, Elsevier Science, Amsterdam, The Netherlands, 227-261.

Plaisted D.A. (1984), Using Examples, Case Analysis and Dependency Graphs in Theorem Proving, In Shostak R.E. (Ed.), *Proceedings of the 7th International Conference on Automated Deduction* (Napa, CA, 1984), (Lecture Notes in Computer Science 170), Springer-Verlag, New York, NY, 356-374.

Plaisted D.A. (1988), Non-Horn Clause Logic Programming Without Contrapositives, In *Journal of Automated Reasoning* 4(3), Kluwer Academic Publishers, Dordrecht, The Netherlands, 287-325.

Plaisted D.A. (1990a), Mechanical Theorem Proving, In Banerji R.B. (Ed.), *Formal Techniques in Artificial Intelligence, A Sourcebook*, Elsevier Science, Amsterdam, The Netherlands, 269-320.

Plaisted D.A. (1990b), A Sequent-Style Model Elimination Strategy and a Positive Refinement, In *Journal of Automated Reasoning* 6(4), Kluwer Academic Publishers, Dordrecht, The Netherlands, 389-402.

Plaisted D.A. (1991), Implementation of the Modified Problem Reduction Format Theorem Prover, Computer Program, Department of Computer Science, University of North Carolina, Chapel Hill, NC.

Plotkin G.D. (1972), Building-in Equational Theories, In Meltzer B., Michie D. (Eds.), *Machine Intelligence* 7, Edinburgh University Press, Edinburgh, Scotland, 73-91.

Pollack M.E. (1991), The Use of Plans, In Mylopolous J., Reiter R. (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (Sydney, Australia, 1991), International Joint Conferences on Artificial Intelligence Inc., Los Altos, CA, Computers and Thought Lecture.

Popplestone R.J. (Unpublished), Freddy, things and sets.

Raphael B. (1969), Some Results about Proof by Resolution, In *SIGART* 14, ACM Press, New York, NY, 22-25.

Reiter R. (1971), Two Results on Ordering for Resolution with Merging and Linear Format, In *Journal of the ACM* 18(4), ACM Press, New York, NY, 630-646.

Reiter R. (1973), A Semantically Guided Deduction System for Automatic Theorem Proving, In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence* (Stanford, CA, 1973), Stanford Research Institute, Menlo Park, CA, 41-46.

Ringwood G.A. (1988), SLD: A Folk Acronym, In Moss C. (Ed.), *Logic Programming Newsletter* 2(1), Association for Logic Programming, London, England, 5-7.

Robinson G.A. and Wos L. (1969), Completeness of Paramodulation, In *Journal of Symbolic Logic* 34, Association for Symbolic Logic Inc., Providence, RI, 159-160.

Robinson J.A. (1963), Theorem Proving on the Computer, In *J. ACM* 10(2), ACM Press, New York, NY, 163-174.

Robinson J.A. (1965a), A Machine-Oriented Logic Based on the Resolution Principle, In *Journal of the ACM* 12(1), ACM Press, New York, NY, 23-41.

Robinson J.A. (1965b), Automatic Deduction with Hyper-resolution, In *International Journal of Computer Mathematics* 1, Gordon and Breach, London, England, 227-234.

Robinson J.A. (1969), Mechanizing Higher-Order Logic, In Meltzer B., Michie D. (Eds.), *Machine Intelligence* 4, Edinburgh University Press, Edinburgh, Scotland, 151-170.

Sandford D.M. (1977), Formal Specification of Models for Semantic Theorem Proving Strategies, SOSAP-TR-32, Department of Computer Science, Rutgers, The State University of New Jersey, New Brunswick, NJ.

Sandford D.M. (1980), *Using Sophisticated Models in Resolution Theorem Proving*, (Lecture Notes in Computer Science 90), Springer-Verlag, New York, NY.

Schmidt-Schauss M. (1985), A Many-Sorted Calculus with Polymorphic Functions Based on Resolution and Paramodulation, In Joshi A. (Ed.), *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (Los Angeles, CA, 1985), International Joint Conferences on Artificial Intelligence Inc, Los Altos, CA, 1162-1168.

Schmidt-Schauss M. (1988), Computational Aspects of an Order-Sorted Logic with Term Declarations, Doctoral Dissertation, University of Kaiserslautern, Kaiserslautern, Germany.

Schumann J., Letz R. and Kurfess F. (1990), High Performance Theorem Provers : Efficient Implementation and Parallelisation, In Stickel M. (Ed.), *Proceedings of the 10th International Conference on Automated Deduction* (Kaiserslautern, Germany, 1990), (Lecture Notes in Artificial Intelligence 449), Springer-Verlag, New York, NY, Tutorial Session.

Shin D.G. and Irani K.B. (1984), Knowledge Representation using an extension of a Many-Sorted Language, In *Proceedings of the 1st Conference on Artificial Intelligence Applications* (Denver, CO, 1984), IEEE Computer Society Press, Silver Spring, MD, 404-409.

Shostak R.E. (1976), Refutation Graphs, In *Artificial Intelligence* 7, Elsevier Science, Amsterdam, The Netherlands, 51-64.

Slagle J.R. (1965), A Proposed Preference Strategy using Sufficiency Resolution for Answering Questions, UCRL-14361, Lawrence Radiation Laboratory, Livermore, CA.

Slagle J.R. (1967), Automatic Theorem Proving with Renamable and Semantic Resolution, In *Journal of the ACM* 14(4), ACM Press, New York, NY, 687-697.

Slagle J.R. (1972), Automatic Theorem Proving with Built-in Theories Including Equality, Partial Ordering and Sets., In *Journal of the ACM* 19(1), ACM Press, New York, NY, 120-135.

Slagle J.R. (1974), Automated theorem-proving for theories with simplifiers, commutativity and associativity, In *Journal of the ACM* 21(4), ACM Press, New York, NY, 622-642.

Stickel M.E. (1985), Automated Deduction by Theory Resolution, In *Journal of Automated Reasoning* 1(4), Kluwer Academic Publishers, Dordrecht, The Netherlands, 333-356.

Stickel M.E. (1986a), An Introduction to Automated Reasoning, In Bibel W., Jorrand Ph. (Eds.), *Fundamentals of Artificial Intelligence*, (Lecture Notes in Computer Science 232), Springer-Verlag, New York, NY, 75-131.

Stickel M.E. (1986b), A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler, In Siekmann J.H. (Ed.), *Proceedings of the 8th International Conference on Automated Deduction* (Oxford, England, 1986), (Lecture Notes in Computer Science 230), Springer-Verlag, New York, NY, 573-587.

Stickel M.E. (1990), A Prolog Technology Theorem Prover, In Stickel M. (Ed.), *Proceedings of the 10th International Conference on Automated Deduction* (Kaiserslautern, Germany, 1990), (Lecture Notes in Artificial Intelligence 449), Springer-Verlag, New York, NY, 673-674.

Stickel M.E. and Tyson W.M. (1985), An Analysis of Consecutively Bounded Depth-First Search with Applications in Automated Deduction, In Joshi A. (Ed.), *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (Los Angeles, CA, 1985), International Joint Conferences on Artificial Intelligence Inc., Los Altos, CA, 1073-1075.

Sutcliffe G. (1987), Single Interpretation, Domain Based, Semantic Checking, Research Report 87/9, Department of Computer Science, The University of Western Australia, Perth, Australia.

Sutcliffe G. (1989), Complete Linear Derivation Systems for General Clauses, In Wos L. (Ed.), *Association for Automated Reasoning Newsletter* (13), Association for Automated Reasoning, Argonne, Il, 3-4.

Sutcliffe G. (1992), Linear-Input Subset Analysis, In Kapur D. (Ed.), *Proceedings of the 11th International Conference on Automated Deduction* (Saratoga Springs, NY, 1992), Springer-Verlag, New York, NY.

Sutcliffe G. and Tabada W. (1991), Compulsory Reduction in Linear Derivation Systems, In Bibel W. (Ed.), *Artificial Intelligence (Letters to the Editor)* 50, Elsevier Science, Amsterdam, The Netherlands, 131-132.

Tabada W. and Sutcliffe G. (1990), An Analysis of the Selective Linear Model Inference System, Research Report 90/2, Department of Computer Studies, Western Australian College of Advanced Education, Perth, Australia.

Tabada W. (1992), An Analysis and Implementation of Linear Derivation Strategies, MSc Thesis, Department of Computer Science, Edith Cowan University, Perth, Australia.

Tarver M. (1990), An Examination of the Prolog Technology Theorem Prover, In Stickel M. (Ed.), *Proceedings of the 10th International Conference on Automated Deduction* (Kaiserslautern, Germany, 1990), (Lecture Notes in Artificial Intelligence 449), Springer-Verlag, New York, NY, 322-335.

Wakayama T. and Payne T.H. (1990), Case-Free Programs: An Abstraction of Definite Horn Programs, In Stickel M. (Ed.), *Proceedings of the 10th International Conference on Automated Deduction* (Kaiserslautern, Germany, 1990), (Lecture Notes in Artificial Intelligence 449), Springer-Verlag, New York, NY, 87-101.

Walther C. (1983), A Many-Sorted Calculus Based on Resolution and Paramodulation, In Bundy A. (Ed.), *Proceedings of the 8th International Joint Conference on Artificial Intelligence* (Karlsruhe, Germany, 1983), International Joint Conferences on Artificial Intelligence Inc, Los Altos, CA, 882-891.

Walther C. (1984), A Mechanical Solution of Schubert's Steamroller by Many-Sorted Resolution, In *Proceedings of the National Conference on Artificial Intelligence* (Austin, TX, 1984), American Association for Artificial Intelligence, Los Altos, CA, 330-334.

Walther C. (1985), Unification in Many-Sorted Theories, In O'Shea T. (Ed.), *Advances in Artificial Intelligence, Proceedings of the European Conference on Artificial Intelligence* (Pisa, Italy, 1984), Elsevier Science, Amsterdam, The Netherlands, 383-393.

Wang T-C. (1985), Designing examples for semantically guided hierarchical deduction, In Joshi A. (Ed.), *Proceedings of the 9th International Joint Conference on Artificial Intelligence* (Los Angeles, CA, 1985), International Joint Conferences on Artificial Intelligence Inc, Los Altos, CA, 1201-1207.

Wang T-C and Bledsoe W.W. (1987), Hierarchical Deduction, In *Journal of Automated Reasoning* 3(1), Kluwer Academic Publishers, Dordrecht, The Netherlands, 35-77.

Wilson G.A. and Minker J. (1976), Resolution, Refinements and Search Strategies: A Comparative Study, In *IEEE Transactions on Computers* C-25(8), IEEE Computer Society Press, Washington, DC, 782-801.

Winker S. (1982), Generation and Verification of Finite Models and Counterexamples Using an Automated Theorem Prover Answering Two Open Questions, In *Journal of the ACM* 29(2), ACM Press, New York, NY, 273-284.

Winston P.H. (1984), *Artificial Intelligence*, Addison-Wesley, Reading, MA.

Wos L. (~1965), Unpublished notes, Argonne National Laboratory, Argonne, IL.

Wos L. (1988), *Automated Reasoning - 33 Basic Research Problems*, Prentice-Hall, Englewood Cliffs, New Jersey.

Wos L., Carson D. and Robinson G.A. (1964), The Unit Preference Strategy in Theorem Proving, In *Proceedings of the AFIPS 1964 Fall Joint Computer Conference* (San Francisco, CA, 1964), Spartan Books, Baltimore, MD, 615-621.

Wos L., Robinson G.A. and Carson D.F. (1965), Efficiency and Completeness of the Set of Support Strategy in Theorem Proving, In *Journal of the ACM* 12(4), ACM Press, New York, NY, 536-541.

Wos L., Verhoff R., Smith B. and McCune W. (1984), The Linked Inference Principle, II: The User's Viewpoint, In Shostak R.E. (Ed.), *Proceedings of the 7th International Conference on Automated Deduction* (Napa, CA, 1984), (Lecture Notes in Computer Science 170), Springer-Verlag, New York, NY, 316-332.

Yates R.A., Raphael B. and Hart T.P. (1970), Resolution Graphs, In *Artificial Intelligence* 1, Elsevier Science, Amsterdam, The Netherlands, 247-256.

Zamov N.K. and Sharonov V.I. (1969), On a class of strategies which can be used to prove theorems by the resolution principle (In Russian), In *Issled, po konstruktivnoye matematikye i matematicheskoie logikye* III(16), National Lending Library Russian Translating Program 5857, Boston Spa, England, 54-64.

Appendix One

Examples

This appendix holds examples that are too bulky to be retained in the main text.

A1.1. Trace of Algorithm 4.13

```
M2  D = {}
M4  Build({homer})
B2  Instance = homer
B9  {mr_s} = GetValuesFromUser(homer)
B11 homer ∅R mr_s
B13 D = {mr_s}
B14 Build({mr_s})
B2      Instance = mr_s
B9      {person} = GetValuesFromUser(mr_s)
B11     mr_s ∅R person
B13     D = {mr_s, person}
B14     Build({person})
B2          Instance = person
B9          {} = GetValuesFromUser(person)
M4  Build({spouse_of(person), heart_ok(person),
    lungs_ok(person), alive(person)})
B2  Instance = spouse_of(person)
B9  {expand(1)} = GetValuesFromUser(spouse_of(person))
B11 spouse_of(person) ∅R expand(1)
B15 CheckExpand(spouse_of(person),expand(1))
C3  Build({spouse_of(mr_s)})
B2      Instance = spouse_of(mr_s)
B9      {mrs_s} = GetValuesFromUser(spouse_of(mr_s))
B11     spouse_of(mr_s) ∅R mrs_s
B13     D = {mr_s, person, mrs_s}
B14     Build({mrs_s})
B2          Instance = mrs_s
B9          {person} = GetValuesFromUser(mrs_s)
```

```
B11         mrs_s ⊘R person
B15         CheckExpand(mrs_s,person)
B2  Instance = heart_ok(person)
B9  {TRUE} = GetValuesFromUser(heart_ok(person))
B11 heart_ok(person) ⊘R TRUE
B13 D = {mr_s, person, mrs_s, TRUE}
B14 Build({TRUE})
B2      Instance = TRUE
B9      {} = GetValuesFromUser(TRUE)
B2  Instance = lungs_ok(person)
B9  {expand(1)} = GetValuesFromUser(lungs_ok(person))
B11 lungs_ok(person) ⊘R expand(1)
B15 CheckExpand(lungs_ok(person),expand(1))
C3  Build({lungs_ok(mr_s), lungs_ok(mrs_s)})
B2      Instance = lungs_ok(mr_s)
B9      {TRUE} = GetValuesFromUser(lungs_ok(mr_s))
B11     lungs_ok(mr_s) ⊘R TRUE
B15     CheckExpand(lungs_ok(mr_s),TRUE)
B2      Instance = lungs_ok(mrs_s)
B9      {FALSE} = GetValuesFromUser(lungs_ok(mrs_s))
B11     lungs_ok(mrs_s) ⊘R FALSE
B13     D = {mr_s, person, mrs_s, TRUE, FALSE}
B14     Build({FALSE})
B2          Instance = FALSE
B9          {} = GetValuesFromUser(FALSE)
B2  Instance = alive(person)
B9  {TRUE} = GetValuesFromUser(alive(person))
B11 alive(person) ⊘R TRUE
B15 CheckExpand(alive(person),TRUE)
M4  Build({spouse_of(person), spouse_of(TRUE),
    spouse_of(FALSE), heart_ok(person), heart_ok(TRUE),
    heart_ok(FALSE), lungs_ok(person), lungs_ok(TRUE),
    lungs_ok(FALSE), alive(person), alive(TRUE),
    alive(FALSE)})
B2  Instance = spouse_of(person)
B5  CheckExpand(spouse_of(person),expand(1))
C3  Build({spouse_of(mr_s),spouse_of(mrs_s))})
B2      Instance = spouse_of(mr_s)
B5      CheckExpand(spouse_of(mr_s),mrs_s)
B2      Instance = spouse_of(mrs_s)
B9      {mr_s} = GetValuesFromUser(spouse_of(mrs_s))
B11     spouse_of(mrs_s) ⊘R mr_s
B15     CheckExpand(spouse_of(mrs_s),mr_s)
```

```
B2  Instance = spouse_of(TRUE)
B9  {} = GetValuesFromUser(spouse_of(TRUE))
B2  Instance = spouse_of(FALSE)
B9  {} = GetValuesFromUser(spouse_of(FALSE))
B2  Instance = heart_ok(person)
B5  CheckExpand(heart_ok(person),TRUE)
B2  Instance = heart_ok(TRUE)
B9  {} = GetValuesFromUser(heart_ok(TRUE))
B2  Instance = heart_ok(FALSE)
B9  {} = GetValuesFromUser(heart_ok(FALSE))
B2  Instance = lungs_ok(person)
B5  CheckExpand(heart_ok(person),expand(1))
C3  Build({lungs_ok(mr_s), lungs_ok(mrs_s)})
B2      Instance = lungs_ok(mr_s)
B5      CheckExpand(lungs_ok(mr_s),TRUE)
B2      Instance = lungs_ok(mrs_s)
B5      CheckExpand(lungs_ok(mrs_s),FALSE)
B2  Instance = lungs_ok(TRUE)
B9  {} = GetValuesFromUser(lungs_ok(TRUE))
B2  Instance = lungs_ok(FALSE)
B9  {} = GetValuesFromUser(lungs_ok(FALSE))
B2  Instance = alive(person)
B5  CheckExpand(alive(person),TRUE)
B2  Instance = alive(TRUE)
B9  {} = GetValuesFromUser(alive(TRUE))
B2  Instance = alive(FALSE)
B9  {} = GetValuesFromUser(alive(FALSE))
```

A1.2. The Compiled Version of Designation `simpsons`

```
/*-------------------------------------------------------
    */
designation(simpsons).
compiled__(simpsons).
/*-------------------------------------------------------
    */
domain_element(mr_s,simpsons).
domain_element(mrs_s,simpsons).
domain_element(person,simpsons).
domain_element(false,simpsons).
domain_element(true,simpsons).
/*-------------------------------------------------------
    */
expand_in_domain__(mr_s,mr_s,simpsons).
expand_to_minimal_in_domain__(mr_s,mr_s,simpsons).
expand_in_domain__(mrs_s,mrs_s,simpsons).
expand_to_minimal_in_domain__(mrs_s,mrs_s,simpsons).
expand_in_domain__(person,person,simpsons).
expand_in_domain__(person,mr_s,simpsons).
expand_to_minimal_in_domain__(person,mr_s,simpsons).
expand_in_domain__(person,mrs_s,simpsons).
expand_to_minimal_in_domain__(person,mrs_s,simpsons).
expand_in_domain__(false,false,simpsons).
expand_to_minimal_in_domain__(false,false,simpsons).
expand_in_domain__(true,true,simpsons).
expand_to_minimal_in_domain__(true,true,simpsons).
/*-------------------------------------------------------
    */
expand_to_term__(mr_s,homer,simpsons).

expand_to_term__(person,mr_s,simpsons).

expand_to_term__(person,mrs_s,simpsons).

expand_to_term__(mrs_s,spouse_of(P),simpsons):-
    expand_compiled__(mr_s,P,simpsons).
expand_to_term__(mr_s,spouse_of(P),simpsons):-
    expand_compiled__(mrs_s,P,simpsons).
```

```
expand_to_term__(true,heart_ok(P),simpsons):-
    expand_compiled__(person,P,simpsons).

expand_to_term__(true,lungs_ok(P),simpsons):-
    expand_compiled__(mr_s,P,simpsons).
expand_to_term__(false,lungs_ok(P),simpsons):-
    expand_compiled__(mrs_s,P,simpsons).

expand_to_term__(true,alive(P),simpsons):-
    expand_compiled__(person,P,simpsons).

expand_to_term__(true,person(P),simpsons):-
    expand_compiled__(person,P,simpsons).
/*-------------------------------------------------------------
    */
expand_compiled__(Domain_element,Term,Designation_name):-
    var(Term),
    !,
    expand_to_minimal_in_domain__(Domain_element,Term,
Designation_name).

expand_compiled__(Domain_element,Term,Designation_name):-
    domain_element(Term,Designation_name),
    !,
    expand_in_domain__(Domain_element,Term,Designation_name)
    .

expand_compiled__(Domain_element,Term,Designation_name):-
    expand_in_domain__(Domain_element,Expanded_domain_elemen
    t,
Designation_name),
    expand_to_term__(Expanded_domain_element,Term,
Designation_name).
/*-------------------------------------------------------------
    */
```

A1.3. The Effects of the Rightwards Subchain System

To illustrate the effects of the rightwards subchain system, the first part of an SGLD deduction, for the Schubert's Steamroller problem (see appendix 2), is given below. The input chains used in this example (a subset of the input chains for Schubert's Steamroller) are as follows.

```
input_chain__(2,
    [b(++animal(Wolf),true), b(--wolf(Wolf),false)],
    wolf_is_an_animal, axiom, input_clause).
input_chain__(1,
    [b(++wolf(a_wolf),true)],
    there_is_a_wolf, axiom, input_clause).
input_chain__(1,
    [b(++grain(a_grain),true)],
    there_is_a_grain, axiom, input_clause).
input_chain__(8,
    [b(++eats(Animal,Plant),unknown),
     b(++eats(Animal,SmallAnimal),unknown),
     b(--animal(Animal),false), b(--plant(Plant),false),
     b(--animal(SmallAnimal),false),
     b(--plant(OtherPlant),false),
     b(--much_smaller(SmallAnimal,Animal),false),
     b(--eats(SmallAnimal,OtherPlant),unknown)],
    eating_habits, axiom, input_clause).
input_chain__(5,
    [b(--animal(Animal),false),b(--
animal(GrainEater),false),
     b(--grain(Grain),false), b(--eats(Animal,GrainEater),
     unknown), b(--eats(GrainEater,Grain),unknown)],
    prove_the_animal_exists, theorem, input_clause).
```

The chain `prove_the_animal_exists` is chosen as the top chain, due to its `theorem` status. The LISS of the complete input set is then extracted. It is {`~bird`/1, `~caterpillar`/1, `~fox`/1, `~snail`/1, `~wolf`/1, `~animal`/1, `~grain`/1, `~plant`/1, `~much_smaller`/2}. The trace below shows the activity of SGLD using the literal-selected search style, with an initial depth bound of 10. The rightwards subchain

system is used to guide the search, using the second designation provided for this problem (see appendix 2). Note that the deduction is shown in the standard chain format, rather than the SGLD internal representation which reverses the order of the centre chains.

```
~eats(GrainEater,Grain) ~eats(Animal,GrainEater) ~grain(Grai
n)
    ~animal(GrainEater) ~animal(Animal)
```

- Extends with `wolf_is_an_animal` to produce :

```
~eats(GrainEater,Grain) ~eats(Animal,GrainEater)
    ~grain(Grain) ~animal(GrainEater) [~animal(Animal)] 0
    ~wolf(Animal)
```

- Extends with `there_is_a_wolf` to produce :

```
~eats(GrainEater,Grain) ~eats(a_wolf,GrainEater) ~grain(Grai
n)
    ~animal(GrainEater) [~animal(a_wolf)] 0 [~wolf(a_wolf)] 0
```

- Truncates twice to produce :

```
~eats(GrainEater,Grain) ~eats(a_wolf,GrainEater) ~grain(Grai
n)
    ~animal(GrainEater)
```

  - The lemma `wolf(a_wolf)` is produced and subsumed.
  - The lemma `animal(a_wolf)` is added to the input set.

- Extends with the lemma `animal(a_wolf)` to produce :

```
~eats(a_wolf,Grain) ~eats(a_wolf,a_wolf) ~grain(Grain)
    [~animal(a_wolf)] 0
```

- Truncates to produce :

```
~eats(a_wolf,Grain) ~eats(a_wolf,a_wolf) ~grain(Grain)
```

  - The lemma `animal(a_wolf)` is produced and subsumed.

- Extends with `there_is_a_grain` to produce :

```
~eats(a_wolf,a_grain) ~eats(a_wolf,a_wolf) [~grain(a_grain)]
0
```

- Truncates to produce :

```
~eats(a_wolf,a_grain) ~eats(a_wolf,a_wolf)
```

  - The lemma `grain(a_grain)` is produced and subsumed.

- Extends with `eating_habits` to produce :

```
~eats(a_wolf,a_grain) [~eats(a_wolf,a_wolf)] 0
    eats(a_wolf,SmallAnimal) ~animal(a_wolf) ~plant(a_wolf)
    ~animal(SmallAnimal) ~plant(OtherPlant)
    ~much_smaller(SmallAnimal,a_wolf)
    ~eats(SmallAnimal,OtherPlant)
```

- Unit subsumed extends with the lemma `animal(a_wolf)` to produce :

```
~eats(a_wolf,a_grain) [~eats(a_wolf,a_wolf)] 0
    eats(a_wolf,SmallAnimal) ~plant(a_wolf)
    ~animal(SmallAnimal) ~plant(OtherPlant)
    ~much_smaller(SmallAnimal,a_wolf)
    ~eats(SmallAnimal,OtherPlant)
```

- The literal `~plant(a_wolf)` has an expected truth value FALSE, because it will become the top literal of a linear-input subdeduction (see the expected truth value field of the literal in the input chain `eating_habits`). As a result, this centre chain is rejected by the rightwards subchain system. Without the semantic deletion, a significant amount of useless deduction takes place, to remove the literals to the right of `~plant(a_wolf)`. Only after this wasted effort is it discovered that `~plant(a_wolf)` cannot be removed. The alternative, of extending on the second literal of `eating_habits`, is then tried. This produces :

```
~eats(a_wolf,a_grain) [~eats(a_wolf,a_wolf)] 0
    eats(a_wolf,Plant) ~animal(a_wolf) ~plant(Plant)
    ~animal(a_wolf) ~plant(OtherPlant)
    ~much_smaller(a_wolf,a_wolf) ~eats(a_wolf,OtherPlant)
```

- The literal `~much_smaller(a_wolf,a_wolf)` has an expected truth value FALSE. As a result, this centre chain is rejected by the rightwards subchain system. Without the semantic deletion, useless deduction again ensues before alternatives, at the fourth step of the deduction, are considered.

Appendix Two

Test Problems and Designations

This appendix contains the statements of the test problems used to test SGLD and descriptions of the designations used for testing SGLD's semantic guidance system.

Additive Algebra 01 (Algebra)
Demonstrates the associativity of addition in a form symmetric to that given in the axioms. Problem 28 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Additive Algebra 02 (Algebra)
(a-b)+c = a+(c-b). Problem 29 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Group Theory 03 (Algebra)
In a group the left identity is also a right identity. Problem 3 in [Chang, 1970].
Designation : An Abelian group with four elements (including the identity).

Group Theory 06 (Algebra)
If S is a non-empty subset of a group such that if x, y belong to S then $x \cdot y^{-1}$ belongs to S, then S contains x^{-1} whenever it contains x. Problem 6 in [Chang, 1970].
Designation : An Abelian group with four elements (including the identity).

Group Theory 11 (Algebra)
In a group, the inverse of an inverse is the original. Problem 8 in [Wos, ~1965]. Obtained from the SPRFN problem set [Plaisted, 1991].

Monoids 01 (Algebra)
In an associative system with an identity element, if the square of every element is the identity, the system is commutative. Problem 2 in [Chang, 1970].
Designation : An Abelian group with four elements (including the identity).

Semi-Groups 01 (Algebra)
In an associative system with left and right solutions, there is a right identity element. Problem 1 in [Chang, 1970].

Semi-Groups 04 (Algebra)
In a semi-group with left inverses and left identity, every element has a right inverse. Problem 5 in [Wos, ~1965]. Obtained from the SPRFN problem set [Plaisted, 1991].

Subgroups 01 (Algebra)
A sub-group has an identity. Problem 12 in [Wos, ~1965]. Obtained from the SPRFN problem set [Plaisted, 1991].

Subgroups 02 (Algebra)
In a subgroup, there is an identity and its the same as for the group. Problem 13 in [Wos, ~1965]. Obtained from the SPRFN problem set [Plaisted, 1991].

Subgroups 03 (Algebra)
A subgroup is closed under inverse. Problem 14 in [Wos, ~1965]. Obtained from the SPRFN problem set [Plaisted, 1991].

Subgroups 10 (Algebra)
A subgroup is closed under inverse. This is a cut down version with only the required axioms supplied. Problem 26 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Intermediate Value Theorem (Analysis)
The mean value theorem in analysis. If a function f is continuous in a real closed interval [a,b], where $f(a) \leq 0$ and $0 \leq f(b)$, then there exists X such that $f(X) = 0$. Problem 2 in [Wang & Bledsoe, 1987]. This version of the Intermediate Value Theorem is the 1st order logic version. To prove the real version of the Intermediate Value Theorem, using the least upper bound axiom, it is necessary to instantiate a set variable. Once that variable has been instantiated by a particular value then one obtains this 1st order logic version. The real version of the theorem was proved by Ballantyne and Bledsoe [1977], by finding a suitable value for the set variable and then proving this resulting 1st order logic version. (This detail of the origin of the Intermediate Value Theorem has been provided by Bledsoe [1992].)

Prime Numbers 01 (Number Theory)
If a is a prime and $a = b^2/c^2$, then a divides b. Problem 7 in [Chang, 1970].

Prime Numbers 02 (Number Theory)
Any number greater than 1 has a prime divisor. Problem 8 in [Chang, 1970].

Prime Numbers 03 (Number Theory)
There exist infinitely many primes. This a cut down version with only the required clauses supplied. Problem 9 in [Chang, 1970].

Prime Numbers 04 (Number Theory)
There exist infinitely many primes. Problem 17 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Recursive Functions 01 (Number Theory)
Symmetry of equality can be derived. Problem 41 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Recursive Functions 05 (Number Theory)
Zero is less than all successor numbers. Based on problem 68 in [Lawrence & Starkey, 1974]. The surplus transitivity axiom has been deleted and the transitivity of less has been added. Original obtained from the SPRFN problem set [Plaisted, 1991].

Recursive Functions 10 (Number Theory)
If a<b then not b<a. Based on problem 76.1 in [Lawrence & Starkey, 1974]. Original obtained from the SPRFN problem set [Plaisted, 1991].
Designation : The normal semantics of natural numbers, with a mapped to 1 and b to 2.

Naive Set Theory 02 (Set Theory)
The union of a set with itself is equal to the set itself. Problem 103 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Naive Set Theory 03 (Set Theory)
A set is a subset of the union of itself with itself. Problem 105 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Naive Set Theory 04 (Set Theory)
A set is a subset of the union of itself and another set. Problem 106 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Naive Set Theory 06 (Set Theory)
If the intersection of two sets is the first of the two sets, then the first is a subset of the second. Problem 111 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Naive Set Theory 08 (Set Theory)
The difference of two sets contains no members of the subtracted set. Problem 115 in [Lawrence & Starkey, 1974]. Obtained from the SPRFN problem set [Plaisted, 1991].

Getting Bread (Planning)
The problem is to drive from Cheyenne, Wyoming to Des Moines, Iowa, buying a loaf of bread on the way. A portion of the road map is expressed in clause form. The allowable actions are to drive from a city to a neighbouring city, to buy a loaf of bread at a city and to wait in a city for one unit of time. Buying a loaf of bread takes one unit of time and driving to a neighbouring city takes two units of time. A problem in [Plaisted, 1981].
Designation 1 : Knows the adjacency of towns. Knows that it is stupid to go the wrong way, to buy more than one loaf, or to wait anywhere. These actions are mapped to FALSE. Bread can be bought anywhere. The sorts of the various objects and the argument sorts of the predicates are also known.
Designation 2 : Designation 1, but with no knowledge of sorts. Meaningless universe and base elements which are not interpreted by designation 1, are interpreted as "meaningless" and FALSE respectively.

Getting There 1 (Planning)
The problem is to travel from one place to another. Certain paths are passable at different times of the year, so a conditional plan must be generated. Either all situations are cold or all situations are warm. There is a river which may be crossed only in winter when it is covered with ice and a mountain range that may be crossed only in summer. The problem is to get from city F to city A. Problem 5.7 in [Plaisted, 1982].
Designation : It is meaningless to travel from a place if you are not there and no-one travels in circles (mapped to FALSE). The travel modes are limited to those in the clauses.

Monkey and Banana (Planning)
The state space representation of the Monkey and Bananas problem, as formulated in the SPRFN problem set [Plaisted, 1991].
Designation 1 : Knows the sorts of the various objects and the argument sorts of the predicates. Also knows that the monkey must be on the ladder to get the bananas.
Designation 2 : Designation 1 extended to know that the ladder must be at the same location as the bananas.

Aunt Agatha (Puzzle)

A slight variant of the version of this problem given by Manthey and Bry [1988, p. 430]. The problem is described in [Pelletier, 1986, p. 206] : Someone who lives in Dreadsbury Mansion killed Aunt Agatha. Agatha, the butler and Charles live in Dreadsbury Mansion and are the only people who live therein. A killer always hates his victim and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Agatha is not the butler. Therefore : Agatha killed herself.

Borders (Puzzle)

There is a database of assertions about various countries and oceans and their relationships. Find which ocean borders on African and Asian countries. Problem 5.6 in [Plaisted, 1982]

Designation 1 : The designation reflects the real geography of the situation. Sorts are known.

Schubert's Steamroller (Puzzle)

Wolves, foxes, birds, caterpillars and snails are animals and there are some of each of them. Also there are some grains and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants. Therefore there is an animal that likes to eat a grain-eating animal. Problem 47 in [Pelletier, 1986]

Designation 1 : Knows about sorts. Knows that only animals eat and that only two animals can be compared in size. Meaningless sort literals, `eats` literals and `much_smaller` literals are not interpreted. Meaningful `eats` literals are all interpreted as UNKNOWN_TRUTH_VALUE.

Designation 2 : Same as designation 1, but meaningless sort literals, `eats` literals and `much_smaller` literals are interpreted as FALSE. Should give results similar to designation 1, as LISS analysis detects the problems with sort literals and `much_smaller` literals.

Truthtellers and the Liars (Puzzle)

On a certain island the inhabitants are partitioned into those who always tell the truth and those who always lie. I landed on the island and met three inhabitants A, B and C. I asked A, 'Are you a truthteller or a liar?' He mumbled something which I couldn't make out. I

asked B what A had said. B replied, 'A said he was a liar'. C then volunteered, 'Don't believe B, he's lying!' What can you tell about A, B and C? A problem in [Lusk & Overbeek, 1985].

Blind Hand 2 (Miscellaneous)
A version of Popplestone's [1970] Blind Hand Problem. Problem DBABHP in [Michie, Ross, & Shannan, 1972].

Blind Hand 3 (Miscellaneous)
A variant of Blind Hand 2, obtained by excluding clauses regarding hand movement.
Designation : The object is initially here, being held. The object is then dropped, picked up again and taken there.

Computing 2 (Miscellaneous)
A computing state space, with eight states - P1 to P8. P1 leads to P3 via P2. There is a branch at P3 such that the following state is either P4 or P6. P6 leads to P8, which has a loop back to P3, while P4 leads to termination. The problem is to show that there is a loop in the computation, passing through P3. Problem BURSTALL in [Reboh, Raphael, Yates, Kling, & Verlarde, 1972]. Obtained from the SPRFN problem set [Plaisted, 1991].
Designation 1 : The designation knows the layout of the state space and which states can be reached from which others. It also knows the argument sorts for all predicates and functions.
Designation 2 : Designation 1, but with no knowledge of sorts. Meaningless universe and base elements which are not interpreted by designation 1, are interpreted as "meaningless" and FALSE respectively.

Computing 3 (Miscellaneous)
A variant of Computing 2, obtained by considering failure in the state space, rather than success.
Designation : The designation knows the layout of the state space and which states can be reached from which others.

Has Parts 2 (Miscellaneous)
Shows that the boy John has ten fingers. Problem HASPARTS-T2 in [Reboh et al., 1972]. Obtained from the SPRFN problem set [Plaisted, 1991].
Designation : The normal semantics of human anatomy. The designation knows about the sorts involved, but due to the clauses' structures this does not affect deductions.

Latin Squares (Miscellaneous)

The inconstructability of a Graeco-Latin Square for t=0 in 4t + 2. A problem in [Robinson, 1963].

Designation : The 'non-square' in which the Greek and Latin layers are identical.

Pigeon 4 (Miscellaneous)

Suppose there are N holes and (N + 1) objects to put in the holes. Every object is in a hole and no hole contains more than one object. The representation of this situation produces an inconsistent set of clauses. Problem 72 in [Pelletier, 1986].

Designation : The Nth pigeon is placed in the Nth hole. The (N+1)th pigeon is left out.

XOR evaluation (Miscellaneous)

The evaluation of exclusive OR represented in clausal form, the goal being to evaluate ((((T^F)^F)^T)^T), where ^ represents exclusive OR. Problem 5.1 in [Plaisted, 1982].

Designation : The usual semantics of exclusive OR. Everything is meaningful in this designation.