Spring 5-31-1999

Recasting Cohn's many sorted logic into a constrained logic

Christopher Brendan Koelbl
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

RECASTING COHN'S MANY SORTED LOGIC INTO A CONSTRAINED LOGIC

by
Christopher Brendan Koelbl

The use of a many sorted logic for theorem proving carries many advantages over a traditional unsorted logic. By placing restrictions on the search space, a many sorted logic can significantly reduce the amount steps in the resolution process. However, as a logic becomes more efficient, it increases in complexity.

One of these efficient logics is Cohn's Many Sorted Logic, LLAMA. It uses complex data structures such as the sort lattice and sort arrays to maintain information about the sorts. Recasting LLAMA into Bürckert's constrained logic will keep the functionality of LLAMA while using a format that reduces the complexity and maintains a more traditional resolution rule.

RECASTING COHN'S MANY SORTED LOGIC INTO A CONSTRAINED LOGIC

by
Christopher Brendan Koelbl

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

May 1999

APPROVAL PAGE

RECASTING COHN'S MANY SORTED LOGIC INTO A CONSTRAINED LOGIC

Christopher Brendan Koelbl

Dr. Richard B. Sherl, Thesis Advisor — Date
Assistant Professor of Computer Science, NJIT

Dr. Thomas Featheringham, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. James McHugh, Committee Member — Date
Full Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Christopher Brendan Koelbl

Degree: Master of Science in Computer Science

Date: May 1999

Undergraduate and Graduate Education:

- Master of Science in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 1999
- Bachelor of Science in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 1999

Major: Computer Science

For My Mother

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Automated Reasoning is a field within Artificial Intelligence that develops methods of proving theorems. Very rarely is this done in a natural language, such as English. Usually the problem is translated into a specialized language, referred to as a logic. There is ongoing research into faster, more efficient methods of problem solving. One step that has made a significant improvement in the deductive process has been breaking up the universe into smaller subsets, called sorts. With this one small change, the number of possibilities for resolution at each steps dramatically decreases over normal, unsorted resolution. The advantage of sorted resolution doesn't come at much of a cost, as will be shown in a review of Frisch's Substitutional Framework. With one small change to the deduction rule, the benefits of a many sorted logic can be gained. However, further optimization comes at a higher cost with greater complexity, usually requiring special rules.

The objective of this thesis is to reconstruct Cohn's Many Sorted Logic, LLAMA [2], utilizing a constrained logic that will provide the same expressiveness and functionality as LLAMA, but uses a more understandable format.

An overview of Frisch's General Framework for Sorted Deduction [4] is given as an example of a basic many sorted logic. The use of the word basic does not imply an inherent lack of functionality or expressiveness but is used to indicate that it has not been optimized for efficiency beyond its many sorted

nature. Frisch's substitutional framework provides a good start for understanding how sorted deduction works.

Second, Cohn's logic, LLAMA, will be covered and compared to Frisch's approach. The advantages of LLAMA will be discussed as well as the consequences that come with the increased efficiency.

Next, Bürckert's constrained logic [1] is presented as an alternative to sorted logics. Constraints, like sorts, provide an intuitive limitation on what a variable can stand for. However, the algorithm by which clauses are resolved in Bürckert's resolution principle provides a clearer explanation of why the clauses can be resolved.

Finally a framework for recasting LLAMA to Bürckert's constrained logic will be discussed. This system will be demonstrated with a brief example with proofs in both LLAMA and the constrained logic alternative.

CHAPTER 2

GENERAL FRAMEWORK FOR SORTED DEDUCTION

The most common type of a sorted logic is one in which the sort information only comes into play during the resolution of clauses. This provides a very distinct split between the logic and the restrictions on the variables. An example of this type of sorted logic is Frisch's Substitutional Framework [Frisch, 1991]. For this paper, only a brief overview of the logic will be discussed.

2.1 Adaptation of First Order Predicate Calculus

The name Substitutional Framework comes from the notion that sort information only comes into play when making substitutions during unification. The language Frisch uses is a modified version of First Order Predicate Calculus called Sorted First Order Predicate Calculus (SFOPC). The only syntactic difference between the two is that in SFOPC the variables are restricted to specified sorts. These restricted variables lead to having two distinct pieces to SFOPC, which Frisch calls A-sentences and S-sentences. A-sentences are like ordinary FOPC sentences except that they can have their variables bounded by sorts. To specify that a variable is sorted, the variable name is followed by a semicolon and the sort to which it is bound, as in the case of x: CAT. For readability, Frisch writes the sorts in capital letters. This notation will be continued throughout the discussion of the Substitutional Framework. An example of a complete A-sentence would be:

$\forall$x: CAT Eats(x: CAT, tuna) $\vee$ Sleeps(x: CAT)

Another convention is to refrain from using the same name for two variables ranging over different sorts. If one variable x is bounded by the sort CAT, there should be no other variable x bound by anything other than CAT, as in the case x: DOG. The advantage of this convention is the ability to remove the declaration of the sorts from within the sentence and placed beside the universal quantifier.

The second piece to SFOPC is S-sentences, which are used for declaring the relationships between the sorts. Although they are constructed similarly to A-sentences, S-sentences contain sort symbols as predicates. Examples of an S-sentence are:

$$\text{CAT(molly)} \wedge \text{BIRD(polly)}$$

$$\forall x\ \text{CAT}(x) \rightarrow \text{LOVABLE}(x)$$

As is easily seen, S-sentences allow a definition of a sort hierarchy in which sorts can be subsorts of another sort as well as defining specific instances in which a term or terms is declared to be of a specific sort. The complete package of all S-sentences in a representation is known as the sort module containing the sort theory.

Therefore, a complete representation will contain both the sort module and a set of A-sentences. A problem being represented in SFOPC does not imply that the problem cannot be represented in FOPC. SFOPC is just a specialized version of FOPC that is capable of using sorted resolution. Any A-sentence can be converted to a sentence in FOPC.

2.2 The Substitutional Framework

The main idea behind the Substitutional Framework is not the language, but the way in which it handles resolution. In this case, the deductive system is an extension of unsorted deduction in which substitutions are required to be well-sorted. The well-sortedness of a substitution is based upon the sort theory. Frisch calls substitutions that are well-sorted with respect to the sort theory of the representation Σ-substitutions. These Σ-substitutions fall within the restrictions placed on the variables by their sorts. By utilizing these Σ-substitutions, at each step in the deductive system, the sort module is referenced to determine appropriate information about Σ. Using this method, the sort theory is only used in determining the unification of two sentences and is kept separate from the rest of the deduction. With this approach, it is a simple matter to extend any unsorted deduction method to work with this sorted deduction. The only thing that needs to be changed is the unification algorithm so that it utilizes the sort module. This is the heart of Frisch's Substitution Framework. The goal was to create a simple sorted logic that could easily be applied to a variety of deduction methods without the need to create special rules to insure completeness.

2.3 Limitations of the Framework

However, because of the generality of this framework, there are limitations. These limitations generally cause the proof to require more steps than a more complex, more specialized logic. Some of these, like the lack of polymorphic formulae, do not effect the outcome of the proof. However, the restriction that Σ must entail atomic sentences, even if the information is already implicitly stated in

the sort theory, can limit effectiveness. An example given by Frisch is that of the baby and the dog:

$$\Sigma = \{ \text{BABY(Ralph)} \vee \text{DOG(Ralph)} \}$$

$$\alpha = \{ \quad \forall x\text{: DOG Annoys}(x, \text{Alan}),$$

$$\forall x\text{: BABY Annoys}(x, \text{Alan}) \}$$

In this example, the union of Σ and α results in Annoys(Ralph, Alan). However, while Σ has two minimal Herbrand models, BABY(Ralph) and DOG(Ralph), it does not have any atomic sentences, making the ground instances of α empty relative to Σ. To solve this problem, Frisch uses the Least Model Lemma, which requires that if the sort theory has a disjunction of atomic formulas, it will also contain just those atomic formulas. However, this would result in:

$$\Sigma = \{ \quad \text{Baby(Ralph)}, \text{Dog(Ralph)}, \text{Baby(Ralph)} \vee \text{Dog(Ralph)} \}$$

But this sort theory is too restrictive. It states that there is a baby with the name Ralph as well as a dog with the name Ralph. This is just a limitation that will have to be accepted when using Frisch's Substitutional Framework. Making the changes required would cause the creation of special rules which is precisely what Frisch is looking to avoid. If this limitation restricts a problem from being accurately stated, another type of logic will be required.

Another limitation of the Substitutional Framework is that there isn't a method for dealing with information on the sorts themselves. The sorts within a problem must be disjoint and provide a complete cover of the universe. There is

no notion of a subsort or overlapping between sorts. This does not restrict the Substitutional Framework from being able to solve any type of problem, however, any information about relationships between sorts must be resolved and cannot be dealt with in the "black box" of the sort theory.

CHAPTER 3

MORE EXPRESSIVE FORMULATION OF MANY SORTED LOGIC

A more expressive alternative to Frisch's Substitutional Framework is Cohn's Many Sorted Logic, LLAMA. Cohn did not intend for LLAMA to be a model that is applicable to other deductive methods, but a highly specialized, efficient logic.

3.1 Developing the Sort Lattice

The basis of a for representation in LLAMA is the sort lattice. This is the structure that contains information about the sort hierarchy. The sort lattice contains, for any sort, any associated subsorts or supersorts. The structure of the sort lattice is a complete Boolean lattice. The size of the sort lattice will be 2^{S_1}, with S_1 being the set of base sorts (including both the universal sort and the empty set), and will contain every possible combination of the base sorts. An example sort lattice for S_1 = { $\top$, M, W, C, P, $\bot$ } is given in figure 3.1.

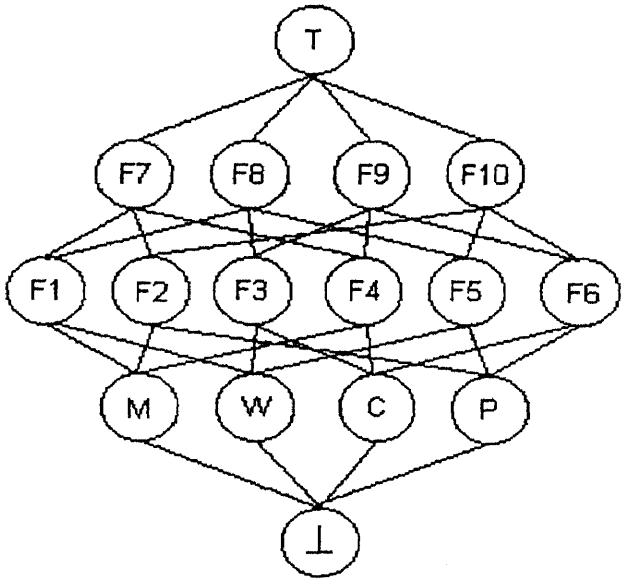

Figure 3.1 Complete Boolean Sort Lattice where S = {T, M, W, C, P, ⊥}

While the base sorts from the problem description will have names, it is not necessary for all the combinations to have identifiable, English names. In some cases, the combinations will never be utilized (such as a possible combinations of horses and golf balls). In other circumstances, there will simply be too many derived sorts to give each one an appropriate name. However, through the disjunction of subsorts and the use of unification, any sort within the sort lattice can be expressed as a combination of the base sorts.

3.2 Creating Sort Arrays

The biggest repository of information on the problem comes not from the sort lattice, but from the sort arrays. A sort array is a list of sort environments (each environment contains sorts corresponding to each variable). Each clause will have its own sort array. But before a sort array can be computed, sorting functions for each function symbol will need to be defined.

A predicate symbol α will have a corresponding sort $\ddot{\alpha}$. This sort can be of 4 possible types. The first two types, TT and FF, are almost self explanatory. These types correspond to TRUE and FALSE respectively. The third type of sort is UU. This sort corresponds to either TT or FF and requires more inferencing to determine what the actual sort is. The last type of sort that a predicate symbol could be is the type EE. This stands for an ill-sorted formula, usually resulting from the predicate being bound by unusable sorts. These sorts are useful for deduction and play a key role in the creation of sort arrays. These sorting functions in LLAMA are defined to be both strict and tight. By strict, Cohn means that if any part of a formula is ill-sorted, the whole formula is ill-sorted. This can

be also be stated to mean that if any part of a proof is ill-sorted, the whole proof is ill-sorted. The other restriction is that the sorting functions must be tight. This simply means that the least upper bounds must be preserved. This can also be referred to as continuity in other logics. These restrictions are part of the reason that LLAMA is a more efficient many sorted logic. Maintaining strictness allows a reduction in search space in the deduction process. When even a part of a formula becomes ill-sorted, the whole formula becomes ill-sorted and is removed, cutting off a branch of the search structure. Restricting sorting functions to be tight allows each sorting function to be a unique determinant based upon its values on the base sorts. This allows a more efficient representation of sortal information.

These sorting functions serve as the tools for creating the sort array (SA) for an expression. The method by which the SAs are derived is very important, because the order in which the steps are performed are not interchangeable. The reason for this is that LLAMA allows overlapping, the repeating of terms within a combination. Because of this, there is an interdependence among the subexpressions containing the same term. The strategy followed in building an SA is to compute all the SAs for the atoms in a top-down fashion, then work from the atoms upwards to get an SA for the whole formula. An example of the computation of an SA for a formula is taken directly from Cohn's paper. The clause in this example comes from the Schubert's Steamroller problem that will be examined in depth as an example of a problem reformulated from LLAMA into a constrained logic representation. The SA is calculated thus:

Initial Clause: { $\neg E[a_1, a_2]$, $\neg E[a_2, j[a_1, a_2]]$ }

Sorting Functions for E:

$$\ddot{E}(<A,T>) = UU$$
$$\ddot{E}(<W,G>) = FF$$
$$\ddot{E}(<W,F>) = FF$$
$$\ddot{E}(<B,S>) = FF$$
$$\ddot{E}(<B,C>) = TT$$

These functions can be combined and rewritten in the form:

$$\ddot{E}(<W,G \cup F>) = FF$$
$$\ddot{E}(<B,S>) = FF$$
$$\ddot{E}(<B,C>) = TT$$
$$\ddot{E}(<A,T>) \backslash <W,G \cup F> \backslash <B,S> \backslash <B,C>) = UU.$$

The final SA for $E[a_1, a_2]$ is:

	a_1	a_2
FF	W	G∪F
FF	B	S
TT	B	C
UU	A\W\B	T
UU	W	T\G\F
UU	B	T\S\C

Negating the formula only results in changing falses to trues and trues to falses.

The next step is to calculate the SA for the second formula, $\neg E[a_2, j[a_1, a_2]]$. This is a two step process, where the SAs for $E[a_1, a_2]$ and $j[a_1, a_2]$ are calculated and then combined, again with the final SA negated. The

SA for the first part of the formula is the same as the one we just calculated. The second SA is just the simple case:

	a_1	a_2
G	A	A

Since the two parts are interrelated, the sort of a_2 must correspond to the sort the second part, G. This allows the immediate removal of the 2nd, 3rd, and 5th environments. After combining the two SAs and simplifying the results, the final SA (after negating) is:

	a_1	a_2
UU	A	A\W
TT	A	W

Combining these two final SAs results in a final SA for the formula of:

	a_1	a_2
TT	W	F
TT	B	S
TT	A	W
UU	A\W\B	A\W
UU	W	A\F\W
UU	B	A\S\W

This example provides a brief look at how a sort array is computed from the base sorting functions. For a closer look at the calculation of SAs, the reader is referred to Cohn, 1986.

3.3 Many Sorted Inference Rules

Now that each formula in the problem can have a sort array calculated for it, the inference mechanism in LLAMA becomes the focus. Inferencing in LLAMA is different from most other sorted logics that use resolution. Through the use of the sort arrays, deriving the empty set is not a requirement for a proof. It is enough to show that a clause can be derived with a sort environment that is false (that is, the environment of the derived clause is FF). Compared to Frisch, LLAMA has a more complex deduction method. In addition to the normal inferencing rule, some additional rules (which Frisch intentionally tried to avoid) are needed.

The most common step in a LLAMA proof is not much different from a normal resolution step. To resolve two clauses together, there are two restrictions. The first, like a normal resolution step, requires that one clause has a positive instance of the predicate and the other clause has a negative instance of the predicate. The second requirement is that the two clauses have sort environments that are compatible. By compatible, the sort environments do not have to be exactly the same, but they must be unifiable. When combining two sort arrays, the greatest upper bound of the two is retained for the resulting clause. An example of a resolution step will help clarify the procedure:

Owns(x, y),Lives(x, c) resolved with ¬Owns(w, z),Bought(w, b)

	x	*y*
UU	M	C
UU	P	H

	w	z
UU	M	A
UU	P	B

results in:

Lives(x, c), Bought(x, b)

	x	c	B
UU	M	US	D

note: sorts for variables c and b were not included in the original two terms because they do not affect the outcome of the resolution step

Taking the sorts M and P to stand for man and person respectively, and the sorts C, H, A, and B to stand for car, home, apartment, and boat, we can see that the two clauses can be resolved together as long as M is a subsort of P and A is a subsort of H. This information is contained in the sort lattice and can be checked during resolution. The reason M and A were kept as the resultant sorts is that these are the greatest lower bounds (largest subsort contained in both sorts). The sort on y, A in this case, can be removed from the sort array because y has no more instances in the clause and will not be utilized in any further resolution steps using this resultant clause.

3.4 Characteristic Literals

The specialized rules for inferencing deal with a topic known as characteristic literals. Including characteristic literals in the logic is a feature relatively unique to LLAMA [3]. While requiring special rules to ensure completeness, characteristic literals can make the resolution process vastly more efficient. The reason for this is that characteristic literals allow reasoning on the sorts themselves. This is an immensely powerful tool for resolution. An example of resolving characteristic literals would be resolving American(mike) with African-Descent(mike) to obtain African-American(mike). American and African-Descent are resolvable because they both predicate the same term, mike. The resultant predicate, African-Descent is not "made up" to combine the two, but is a sort already contained within the sort lattice that is a Boolean combination of the original sorts. The use of characteristic literals is another result of allowing overlapping (terms appearing in more than one predicate in a clause). In certain situations, characteristic literals can be evaluated away immediately. This is allowed when a characteristic literal is predicating a variable whose sort is the one defined by the characteristic literal (i.e. P(p) where p is defined by the sort P). Characteristic literals are one of the features that makes LLAMA a more expressive, more precise logic for formulating a problem as well as allowing an inference step that can quickly reduce the amount of clauses that need to be calculated in a proof.

CHAPTER 4

RESOLUTION PRINCIPLE FOR CONSTRAINED LOGICS

An alternative to a many sorted logic is a constrained logic. A constrained logic can work in a very similar way to sorted logics, but is much more general allowing a greater degree of flexibility. This flexibility allows a proof to be as expressive as the complexity of the constraint theory will allow. It also allows constrained logic to be used for a wider range of problems by making the constraints a user definable part of the logic. Thus, constraint logic could be used to mimic (in a certain sense) a sorted logic by making the unification of sorts to be the constraints or it could be used for a mathematical problem by defining the constraints to be the unification of equations. Whatever the use, constrained logic is similar to the Substitutional Framework in that it views a problem in terms of two parts, the foreground and the background. By breaking up the problem, constrained logic allows different types of information to be declared in different languages that are properly suited to the information to be conveyed.

4.1 The Restricted Quantification System

The structure of importance in this constrained logic is called the restricted quantification system, or RQS for short. This structure contains all the information required for the problem, and can even be thought of as the problem space, with different RQSs representing different problems. The RQS is split into three different subsections, the constraint formulae, the constraint theory, and the Δ signature.

The Δ signature is the list of statements that resolution will take place upon. These statements are usually in clause form, although there is no restriction that they have to be. They could be mathematical formulas. The advantage of using a clausal form for the Δ signature is that the Herbrand universe can be taken as the restriction theory. By taking this restriction, the derivation of the empty clause is all that is required for a proof. Each statement in the Δ signature must have some set of constraints associated with it. In a problem that will use a clause form for the Δ signature, the constraints would restrict the variables within the clauses. These constraints are made up of predicates from the constraint formulae and are required to be consistent with the constraint theory. If the clause form is chosen for the Δ signature, there are no restrictions as to how many times a variable can appear within a clause.

The constraint formulae, as mentioned before, is the set of all predicates that can make up the constraints on the Δ signature. These predicates are obtained from the original problem. Unlike the Substitutional Framework, the constraint formulae is not limited to monadic predicates. This allows relationships between two variables to be considered as part of the constraints rather than parts of the Δ signature. The constraint formulae works hand in hand with the constraint theory. In programming terms, the constraint formulae is the interface while the constraint theory is the actual functions.

So finally, the constraint theory completes the RQS structure. The constraint theory is the actual theory that the constraints must adhere to. This theory will contain information about each predicate in the constraint formulae.

The theory is not restricted to definite clauses as is the case in the Substitutional Framework. If the goal is to mimic a many sorted logic, this can include information about relationships between different sorts. This is also the place where any skolem functions are defined. When two statements from the Δ signature are resolved, the unification of the constraints must not conflict with the constraint theory, otherwise the resolved clause has no meaning. In LLAMA, this would be referred to as an ill-sorted clause. One of the topics of interest in LLAMA was its ability to be able to handle characteristic literals. This type of information about the restrictions on variables can be handled by this constrained logic as well. To accomplish this, within the sort theory, add the information concerning the relationships between the restrictions. Then, when the constraints are tested for unification during resolution, this information will come into play and appropriately restrict the constraints on the variables.

An example RQS structure, taken from the Baby and Dog problem in Chapter 2, is:

Constraint Formulae:

Baby(x) Dog(x)

Constraint Theory:

Baby(Ralph) $\vee$ Dog(Ralph)

Δ signature:

Annoys(x, Alan) || Baby(x)

Annoys(x, Alan) || Dog(x)

4.2 Constrained Resolution

With a set of constraint clauses, a resolution rule is required to allow deduction. The key to resolution within a constrained logic is that, in addition to having two clauses that have a positive and negative instance of a common element, the constraints on those two clauses must be unifiable. By unifiable, it means that the two constraints on the same variable can be made equivalent within the constraint theory. A simple example, given in Bürckert's paper, of a resolution step is:

Constraint Theory:

Herbrand Universe

Δ signature:

$\{ P(x, y), Q(y) \,||\, x = f(v) \wedge y = f(a) \}$

$\{ \neg P(x', y') \,||\, x' = y' \}$

have the RQ-resolvent:

$\{ Q(y) \,||\, x = f(v) \wedge y = f(a) \wedge x' = y' \wedge x = x' \wedge y = y' \}$

This shows that if two clauses have resolvable elements, they can be resolved as long as the unification of their constraints does not conflict with the constraint theory. In this example, with the constraint theory given by the Herbrand Universe, it is enough just to equate the constraints.

If the constraint theory is not given by the Herbrand Universe, it is not enough to resolve one empty clause to prove that a theorem is universally true. Resolving one empty clause proves that the theorem is true for one model of the constraint theory. This is called R-satisfiable. To be universally true (the

equivalent of deriving one empty clause with a constraint theory given by the Herbrand Universe), the empty clause must be derived for every model in the constraint theory. This is called R-valid. An example of a problem that is R-valid is the Baby/Dog problem from section 4.1. The RQS for the Baby/Dog problem was given:

Constraint Formulae:

Baby(x) Dog(x)

Constraint Theory:

Baby(Ralph) $\vee$ Dog(Ralph)

Δ signature:

Annoys(x, Alan) || Baby(x)

Annoys(x, Alan) || Dog(x)

This constraint theory has two models, one for Baby(Ralph) and one for Dog(Ralph). So, for this problem to be R-valid, the Δ signature must be able to derive the empty clause for both models. In this problem, that is very simple to see. To prove that Ralph annoys Alan, ¬Annoys(Ralph, Alan) is added to the Δ signature. This can be resolved with Annoys(x, Alan) || Baby(x) which complies with Baby(Ralph) from the constraint theory to derive the empty clause. It could also be resolved with Annoys(x, Alan) || Dog(x) which complies with Dog(Ralph) in the constraint theory. This shows how two empty clauses have to be derived in an RQS that contains a constraint theory with two models for a theorem to be universally valid.

One of the advantages of a constrained logic is that while the derivation of an empty clause (when the Herbrand universe is part of the constraint theory) proves a theorem, the constraints on that empty clause justify the solution. In a problem where the theorem to be proven is "There is a person who drives a sports car" and the Δ signature is made up of a lot of clauses restricted by various people and cars, the derivation of the empty clause will have constraints that will justify the answer. That means that the constraints on that empty clause will include a sports car and a person that makes the statement correct.

CHAPTER 5

RECASTING LLAMA INTO A CONSTRAINED LOGIC

Now that all the individual types of logic have been covered, it is possible to begin to develop a recasting of Cohn's LLAMA into Bürckert's constrained resolution. A set of guidelines will be developed, followed by an example to help illustrate the key points. The problem used will be Schubert's Steamroller, selected because it was the example Cohn used to when discussing LLAMA and therefore has a complete, correct proof.

5.1 Using Bürckert's Notation

The first step in recasting LLAMA into a constrained logic is to insert the information into an RQS. This is done from the English description of the problem. It is important to remember that in creating the constraint theory and constraint formulae, predicates need not be limited to one term. When performing a recast of LLAMA, it is important to make sure that all the sorting functions in LLAMA have an equivalent representation in the constraint theory.

5.2 Example: Schubert's Steamroller

The problem used as the example for this reformulation is a famous challenge to deduction systems, one given by Schubert back in 1978. The problem, presented in a natural language, is:

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.

The proof of the Steamroller problem in LLAMA only requires 6 steps, 4 less than Walther's Axiomatization. This amounts to a 40% savings over Walther's Axiomatization, which is significant when dealing with problems containing more clauses and larger sort lattices than the example here. The three clauses from which Cohn starts are :

(1') {E[a1, p1], ¬E[a2, p2], E[a1, a2]}

	a1	a2	p1	p2
UU	B	S	P	P
UU	F	B	P	P
UU	W	F	P	P

(2) {E[cs1, h[cs1]]}

	cs1
UU	C ∪ S

(3) {¬ E[a1, a2], ¬ Ea2, j[a2, a2]]}

	a1	a2
UU	A\W\B	A\W
UU	W	A\W\F
UU	B	A\W\S

When arriving at a solution, there are a few rules to remember from the discussion of LLAMA. For a resolution step to be made, the sort arrays of the parent clauses must each contain a row that is unifiable. Also, unlike traditional resolution, Cohn's proof does not end with the empty clause, but instead ends when the sort array of the clause can be equated to false. Below each step in the proof, the number of the two clauses used to resolve to the current clause is given. The predicates within those two clauses used in the resolution is given in parenthesis next to the number of the clause.

(4) {E[a1, p1], ¬ E[a2, p2], ¬ E[a2, j[a1, a2]]}
1'(3) + 3(1)

	a1	a2	p1	p2
UU	F	B	P	P

(5) {E[a1, p1], ¬ E[a2, j[a1, a2]]}
factor of 4

	a1	a2	p1
UU	F	B	P

(6) {E[a3, p1], ¬ E[a2, p2], E[a1, a2]}
5(2) + 1'(1)

	a1	a2	a3	p1	p2
UU	B	S	F	P	P

(6') {E[a3, p1], ¬ E[a2, p2]}

	a2	a3	p1	p2
UU	S	F	P	P

(7) {E[a3, p1]}
6'(2) + 2(1)

	a3	p1
UU	F	P

(8) {E[a1, p1], E[a1, a2]}
7(1) + 1'(2)

	a1	a2	p1
UU	W	F	P\G
FF	W	F	P

There are two things of note in this proof. Going from clause 6 to 6' can be considered a simplification. The reason why a clause is dropped is that the dropped clause is always false and will not contribute at any point in the proof. The other item of note is the final clause. The reason this is the end is that this clause is the final clause is because the sort environment has been proven to be false. By creating structures such as the sort array and the sort lattice, LLAMA becomes a very efficient algorithm.

Now, the reformulation of this problem into a constrained logic will be covered. The first, and most important, step is creating the proper RQS. From the first sentence of the problem description, "Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them", gives 6 predicates to be added to the constraint formulae, and a relationship among them to place in the constraint formulae. At this point, the RQS looks like:

Constraint Formulae:

W(x)	F(x)
B(x)	C(x)
S(x)	A(x)

Constraint Theory:

$W(x) \rightarrow A(x)$
$F(x) \rightarrow A(x)$
$B(x) \rightarrow A(x)$
$C(x) \rightarrow A(x)$
$S(x) \rightarrow A(x)$

Δ signature:

<empty>

The second sentence, "Also there are some grains, and grains are plants", gives two more constraint predicates and one more relationship. The updated RQS is:

Constraint Formulae:

W(x)	F(x)
B(x)	C(x)
S(x)	A(x)
G(x)	P(x)

Constraint Theory:

$W(x) \rightarrow A(x)$
$F(x) \rightarrow A(x)$
$B(x) \rightarrow A(x)$
$C(x) \rightarrow A(x)$
$S(x) \rightarrow A(x)$
$G(x) \rightarrow P(x)$

Δ signature:

<empty>

The third sentence in the description gives the first clause to be placed in the Δ signature. "Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants." The clause to be added, with constraints, to the Δ signature is:

$E(a1, p1), E(a1, a2), E(a2, p2) \parallel A(a1) \wedge A(a2) \wedge M(a2, a1) \wedge P(p1) \wedge P(p2)$

The next sentence, "Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves", presents 4 sorting functions. These are part of the sort theory because they deal with specific instances of one of the constraints, M(a2, a1), of the first clause. The four sorting functions are:

$C(x) \wedge B(y) \rightarrow M(x, y)$
$S(x) \wedge B(y) \rightarrow M(x, y)$
$B(x) \wedge F(y) \rightarrow M(x, y)$
$F(x) \wedge W(y) \rightarrow M(x, y)$

The next sentence gives four clauses to be added to the delta signature. These are clauses instead of sorting functions because they deal with the same relationship that is the focus of the resolution. "Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails" states the following four clauses:

$\neg E[a1, p1] \,||\, W(a1) \wedge G(p1)$
$\neg E[a1, a2] \,||\, W(a1) \wedge F(a2)$
$E[a1, a2] \,||\, B(a1) \wedge C(a2)$
$\neg E[a1, a2] \,||\, B(a1) \wedge S(a2)$

The next sentence gives two more clauses. "Caterpillars and snails like to eat some plants" equates to:

$E[a1, p1] \,||\, C(a1) \wedge P(p1)$
$E[a1, p1] \,||\, S(a1) \wedge P(p1)$

Finally, the last sentence gives the final clause and an addition to the constraint theory. This clause is the theorem that is meant to be proven true or false. "Therefore there is an animal that likes to eat a grain-eating animal" results in the clause:

$E[a1, a2], E[a2, j[a1, a2]] \,||\, A(a1) \wedge A(a2)$

The addition to the constraint theory:

$A(x) \rightarrow G(j[x, x])$

allows the deductive process to handle the Skolem function. Note, because the final clause is the clause that has to be proved, it is negated before being added to the delta signature. So, the final RQS is:

Constraint Formulae:

W(x)	F(x)
B(x)	C(x)
S(x)	A(x)
P(x)	G(x)

Constraint Theory:

$A(x) \rightarrow G(j[x, x])$

$G(x) \rightarrow P(x)$
$W(x) \rightarrow A(x)$
$F(x) \rightarrow A(x)$
$B(x) \rightarrow A(x)$
$C(x) \rightarrow A(x)$
$S(x) \rightarrow A(x)$

$C(x) \wedge B(y) \rightarrow M(x, y)$
$S(x) \wedge B(y) \rightarrow M(x, y)$
$B(x) \wedge F(y) \rightarrow M(x, y)$
$F(x) \wedge W(y) \rightarrow M(x, y)$

Δ signature:

1 $\neg E[a1, a2], \neg E[a2, j[a1, a2]] \parallel A(a1) \wedge A(a2)$

2 $E[a1, p1] \parallel C(a1) \wedge P(p1)$
3 $E[a1, p1] \parallel S(a1) \wedge P(p1)$

4 $\neg E[a1, p1] \parallel W(a1) \wedge G(p1)$
5 $\neg E[a1, a2] \parallel W(a1) \wedge F(a2)$
6 $E[a1, a2] \parallel B(a1) \wedge C(a2)$
7 $\neg E[a1, a2] \parallel B(a1) \wedge S(a2)$

8 $E(a1, p1), E(a1, a2), E(a2, p2) \parallel A(a1) \wedge A(a2) \wedge M(a2, a1) \wedge P(p1) \wedge P(p2)$

The proof is obtained by continually resolving clauses until the empty clause is resolved. This is possible because the constraint theory for this problem only has one model. The proof for this RQS is as follows:

9=8+5 E[a1, p1], ¬E[a2, p2] || W(a1) ∧ F(a2) ∧ M(a2, a1) ∧ P(p1) ∧ P(p2)

10=8+7 E[a1, p1], ¬E[a2, p2] || B(a1) ∧ S(a2) ∧ M(a2, a1) ∧ P(p1) ∧ P(p2)

11=8+9 ¬E[a2, p2], E[a1, a2] || F(a1) ∧ A(a2) ∧ M(a2, a1) ∧ P(p2)

12=11+10 ¬E[a3, p3], E[a1, a2] || F(a1) ∧ B(a2) ∧ S(a3) ∧ M(a2, a1) ∧ M(a3, a2) ∧ P(p3)

13=12+3 E[a1, a2] || F(a1) ∧ B(a2) ∧ M(a2, a1)

14=13+1 ¬E[a2, j[a1, a2]] || F(a1) ∧ B(a2) ∧ M(a2, a1)

15=14+10 ¬E[a3, p2] || F(a1) ∧ B(a2) ∧ M(a2, a1) ∧ S(a3) ∧ M(a3, a2) ∧ P(p2)

16=15+3 || F(a1) ∧ B(a2) ∧ M(a2, a1) ∧ S(a3) ∧ M(a3, a2) ∧ P(p2)

This proof gives the same final result as Cohn's proof in LLAMA, in some instances even having constraints similar to Cohn's sort arrays. By allowing constraints like M(x, y) that work on two variables, this reformulation maintains some of the advantages of LLAMA over an ordinary sorted logic, like Frisch's Substitutional Framework. This can be seen in our proof, requiring only 8 resolution steps compared to Walther's axiomatization, which takes 9. 8 steps is still two more than used by Cohn. However, it is important to keep in mind that this reformulation does not need to calculate sort arrays for every clause. Still, is it possible to optimize this reformulation to produce a constrained logic that even more closely resembles LLAMA?

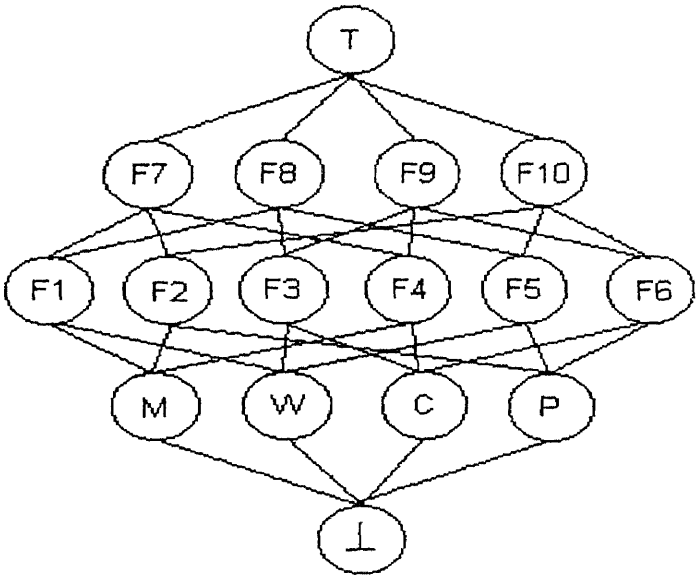

Figure 5.1 Complete Boolean Sort Lattice for S = {T, M, W, C, P, ⊥}

5.3 Future Work : Optimization

The first step in creating a constrained resolution proof that will keep the complete expressiveness of LLAMA intact is to find a way to represent all the information in the sort lattice. The sort lattice represents two key bodies of information. First, it acts as a list of every possible sort in the problem, including both anonymous and eponymous types. And secondly, the sort lattice contains information about the relationships between the sorts. Unlike Cohn, the relationships between the sorts will be contained explicitly. The reason for this has to do with the ability to handle characteristic literals, which will be discussed later in further detail. By choosing to keep all the information explicitly, the size of the proof will be larger. However, much of the information will not be utilized in the actual proof. To help illustrate the procedure for converting a sort array into

constraints, a small example will be constructed using the base sorts man, woman, child, and pet (S = {T, M, W, C, P, ⊥}). The complete lattice sort structure is given in figure 5.1. The method for generating this sort lattice can also be used to generate the constraint formulae and theory. The sort lattice is developed by grouping together combinations of the base sorts. At level 1 in the sort lattice, it is just the base sorts. At level 2, it is a combination of any two of the base sorts. Level 3 is the trickiest. Each node in level three corresponds to a combination of three of the base sorts. Links from nodes in level 3 to nodes in level 2 are inserted so that the node in level 3 is connected to the nodes in level 2 that contain only subsorts of the current sort. So, for example, sort F7 equates to M or W or C. Then, links from F7 go to those links in level 2 which correspond to subsorts of F7 (M ∨ W, M ∨ C, W ∨ C), namely F1, F2, and F4. So, for every node in the sort lattice, the name of the sort is inserted into the constraint formulae, and, for every combination of base sorts (the sorts F1 – F10 in the example), the disjunction of its constituent nodes is inserted into the constraint theory. When applying these rules to the example, the constraint formulae and constraint theory look like:

Constraint Formulae:

M(x), W(x), C(x), P(x), F1(x), F2(x), F3(x), F4(x), F5(x), F6(x), F7(x), F8(x), F9(x), F10(x)

Constraint Theory:

$F1(x) \leftrightarrow M(x) \vee W(x)$
$F2(x) \leftrightarrow W(x) \vee C(x)$
$F3(x) \leftrightarrow M(x) \vee P(x)$
$F4(x) \leftrightarrow M(x) \vee C(x)$
$F5(x) \leftrightarrow W(x) \vee P(x)$
$F6(x) \leftrightarrow C(x) \vee P(x)$

$F7(x) \leftrightarrow F1(x) \vee F2(x) \vee F4(x)$
$F7(x) \Rightarrow M(x) \vee W(x) \vee C(x)$
$F8(x) \leftrightarrow F1(x) \vee F3(x) \vee F5(x)$
$F8(x) \Rightarrow M(x) \vee W(x) \vee P(x)$
$F9(x) \leftrightarrow F3(x) \vee F4(x) \vee F6(x)$
$F9(x) \Rightarrow M(x) \vee C(x) \vee P(x)$
$F10(x) \leftrightarrow F2(x) \vee F5(x) \vee F6(x)$
$F10(x) \Rightarrow W(x) \vee C(x) \vee P(x)$

So, as can be seen from the example, by converting the complete sort lattice into constraint theory and constraint formulae, a complete coverage of all the possible sorts will be embedded into the constraints. The reason why the arrows in the constraint theory go both ways as compared to the one way is that sorts are closed in LLAMA. All possible sorts are listed, with no belief in any sort outside those given in the problem. While this most likely will include sorts that aren't used in the solution of the problem (especially in problems with a large sort domain that includes different 'types' of sorts such as people and cars), it is more important to have all sort possibilities rather than save space.

Still, the constraint theory is not complete. Any Skolem functions used when translating the problem into clausal form will need to be attributed to a sort. This information will be added in the next step, where the problem is translated into clausal form.

Determining what clauses are to be defined by the problem is the same as in any many sorted logic. Stickel appropriately points out that great care must be given to converting a natural language description of the problem into a logical representation [5]. However, this is a concern that is applicable to any problem solver. The area of interest when creating the clauses in this type of constraint logic is to make sure that the information contained in Cohn's sort arrays is accurately formulated. There are two possible solutions for this concern. The first, and easiest solution, is to perform a sort array calculation as per Cohn and then just convert the sort array directly into constraints for the clause. The second option is to create a method of resolution using the sorting functions to directly compute constraints for a specific clause. The initial solution of first computing the sort arrays and then converting those to constraints is the best choice, and a direct calculation of constraints is left to future work. The reason for this choice is that by computing the sort arrays first and then calculating the constraints, the problem is broken up into two distinct steps, similar to the rationale behind a sorted logic. So, aside from the considerations discussed in Chapter 3 concerning the calculation of a sort array, there are two other considerations in the creation of constraints for each clause.

The first point is to make sure that any Skolem functions are appropriately added to the constraint theory. There are two cases when adding a Skolem function. The rules for each case are given below:

Case 1: The Skolem function predicates one sort.

$$\ddot{h}(< S >) = W$$

In this case, the sort S should imply the characteristic predicate W with the Skolem function $\ddot{h}$ as its term.

$$S(x) \rightarrow W(\ddot{h}[x])$$

Case 2: The Skolem function predicates two (or more) terms of the same sort.

$$\ddot{g}(< R, R >) = Y$$

In this case, the sort R should imply the characteristic predicate Y with the Skolem function $\ddot{g}$ as its term.

$$R(x) \rightarrow Y(\ddot{g}[x, x]) .$$

The second consideration is dealing with clauses whose sort arrays contain more than one sort environment. Again, there are two possible solutions to this problem. However, in this case, both solutions are equally valid. The choice is based upon which part of the problem the cost of deduction is to take place. The two choices are either to have multiple instances of the same clause, each with a different set of constraints. This option places more of the work in deduction on the resolution step. When looking for a clause with which to resolve, multiple instances of one choice will be available as in the case:

$E(x, y) \,||\, F(x) \wedge G(y)$

$E(x, y) \,||\, H(x) \wedge G(y)$

The second option is to allow a disjunction of constraints associated with a clause. With this method, only one instance of each clause will be found when looking for a possible resolution. However, the process for unifying constraints becomes more complicated as shown:

$$E(x, y) \,||\, [\, F(x) \wedge G(y)\,] \vee [\, H(x) \wedge G(y)\,]$$

In this transformation, the second option will be used to represent multiple sort environments for a clause. The reason for this choice is that this removes the burden of resolving constraints from the resolution step and reiterates the separation of background and foreground information. However, by taking this route, a new rule during constraint unification is required. This rule is that when resolving two clauses, one of the constraint sets (denoted by [] in the constraint of the clause) in each clause must be unifiable with the other. This will be dealt with in more detail when resolution within this formulation is discussed.

Resolution in this constrained logic is very similar to resolution in either the Substitutional Framework or LLAMA. At each resolution step, the constraints on the variables have to be unifiable. Failure to do this is the same as trying to resolve clauses with two different sorts in LLAMA or SFOPC. As mentioned before, this means that one constraint set from each clause must be resolvable. It is important to note that there won't be a case where two different constraint sets on the same clause are resolvable with a constraint set on the second

clause. The reason for this is that the calculation of the sort array will unify the sort environments as much as possible until only distinct environments exist. In the resolution, there is one big difference between constrained logic and a many sorted logic in that the unification is explicitly stated in a constrained logic. Take the example for the many sorted inference rule in Chapter 3.

$$\text{Owns}(x, y), \text{Lives}(x, c) \;||\; [\, M(x) \wedge C(y) \wedge G(c) \,] \vee [\, P(x) \wedge H(y) \wedge G(c) \,]$$

is to be resolved with

$$\neg\text{Owns}(w, z), \text{Bought}(w, b) \;||\; [\, M(w) \wedge A(z) \wedge F(b) \,] \vee [\, P(w) \wedge B(z) \wedge F(b) \,]$$

Unlike LLAMA, where just the result is presented, the unification of variables must be explicitly stated in the resolved clause as such:

$$\text{Lives}(x, c), \text{Bought}(x, b) \;||\; M(x) \wedge A(y) \wedge F(b) \wedge G(c) \wedge x = w \wedge y = z$$

This is the proper method of denoting a resolution step. The further in the proof you go, which equates to more resolution steps, the list of constraints gets longer and the constraints themselves get more restrictive, or precise. To deal with this, a shorthand notation can be implemented. In this notation, if a variable does not occur anywhere in the clause, the constraints on that variable can be removed from the constraint list. Thus, the resolved clause from the previous example can be written:

$$\text{Lives}(x, c), \text{Bought}(w, b) \;||\; M(x) \wedge F(b) \wedge G(c) \wedge x = w$$

A second piece of shorthand is the ability to remove the constraint x = w in the next resolved clause. This shorter version is useful in that the constraint list will only contain those variables to be considered when resolving with another clause. However, it may be important to go back and obtain a complete list of constraints after the problem is finished.

A key point to remember in the resolution process is that a proof is complete only after the empty clause is resolved or there are no more resolvable clauses left. Since sort arrays are not generated for every resolved clause, there is no way to check whether or not the sort environment is false. However, it can be seen by inspection of Cohn's solution to Schubert's Steamroller that an empty clause with the same sort array can be resolved from his final clause in which the sort environment is false.

A big consideration in this constrained logic is how to deal with characteristic literals. In this circumstance, they will be dealt with as an extension of simplification instead of an offshoot of resolution. After a clause has been resolved, if there are two predicates who have the same term as an argument, they can be checked to see if there is a Boolean combination of the two predicates that can replace their occurrences. Thus, this becomes a step similar to factoring in the proof.

The conversion from the proof in LLAMA to a constrained logic follows the rules given in the previous sections, allowing the constrained logic to mimic the advantages of LLAMA. Here is a quick review of the "algorithm":

Step 1: Create a constraint formulae and constraint theory that contains the sort hierarchy given in LLAMA's sort lattice.

Step 2: Create the initial clauses from the information given in the problem. This piece has two components

1. Create arrays from the clauses and sorting functions given in the problem.

2. Convert the sort arrays to constraints for each clause.

Step 3: Apply the resolution principles to the initial clauses in an attempt to derive the empty clause. Some key points to remember for this stage are:

1. Resolution is handled in the standard way. Negate the desired conclusion and apply resolution until the result is the empty clause.

2. At each step, a constraint set from one clause must be unifiable with a constraint set from the second clause.

3. After a new clause is resolved, check to see if it can be simplified or have characteristic literals unified.

So, the first step is to create constraints that contain the sort hierarchy. Due to the size of the sort lattice created by all the possible combination of sorts, only a subset of the sort lattice will be given here. Coincidentally, it happens to be the exact subset of sorts used in the solution of the problem.

Constraint Formulae

A(x), W(x), F(x), B(x), C(x), S(x), G(x), P(x), Foo1(x), Foo2(x), Foo3(x), Foo4(x), Foo5(x)

Constraint Theory

$Foo1(x) \leftrightarrow C(x) \vee S(x)$
$Foo2(x) \leftrightarrow F(x) \vee C(x) \vee S(x)$
$Foo3(x) \leftrightarrow F(x) \vee B(x) \vee C(x) \vee S(x)$
$Foo4(x) \leftrightarrow B(x) \vee C(x) \vee S(x)$
$Foo5(x) \leftrightarrow F(x) \vee B(x) \vee C(x)$

The next step is to convert the clausal forms of the axioms (from LLAMA) into a clausal form that can work within the bounds of the constrained logic. After applying this process to the initial clauses from the proof in LLAMA, the following clauses are added to the constrained logic clause list:

(1) $\{ E[a1, p1], \neg E[a2, p2], E[a1, a2] \,||\, [B(a1) \wedge S(a2) \wedge P(p1) \wedge P(p2)] \vee [F(a1) \wedge B(a2) \wedge P(p1) \wedge P(p2)] \vee [W(a1) \wedge F(a2) \wedge P(p1) \wedge P(p2)] \}$

(2) $\{ E[b1, h[b1]] \,||\, Foo1(b1) \}$

(3) $\{ \neg E[c1, c2], \neg E[c2, j[c1, c2]] \,||\, [Foo2(c1) \wedge Foo3(c2)] \vee [W(c1) \wedge Foo4(c2)] \vee [B(c1) \wedge Foo5(c2)] \}$

In addition, the following are added to the constraint theory to account for the Skolem functions:

$Foo1(x) \rightarrow P(\ddot{h}[x])$
$A(x) \rightarrow G(\ddot{j}[x, x])$

Finally, the actual resolution is performed on the clauses. This process is given below and performed in such a manner to show that in this constraint logic, a proof that follows the same steps as Cohn's proof can be successfully derived.

(4) { E[a1, p1], ¬E[a2, p2], ¬E[a2, j[a1, a2]] || F(a1) ∧ B(a2) ∧ P(p1) ∧ P(p2) ∧ a1 = c1 ∧ a2 = c2}
1(3) + 3(1)

(5) { E[a1, p1], ¬E[a2, j[a1, a2]] || F(a1) ∧ B(a2) ∧ P(p1) }
factor of 4

(6) { E[a1, p1], ¬E[a2', p1'], E[a1', a2'] || B(a1') ∧ S(a2') ∧ F(a1) ∧ P(p1) ∧ P(p1') ∧ a1' = a2 ∧ j[a1, a2] = p1'}
5(2) + 1(1)

(6') { E[a1, p1], ¬E[a2', p2'] || S(a2') ∧ F(a1) ∧ P(p1) ∧ P(p1') }
literal 6(3) is always false and can be removed

(7) { E[a1, p1] || F(a1) ∧ P(p1) ∧ a2' = b1 ∧ p1' = h[b1] }
6'(2) + 2(1)

(8) { E[a1'', p1''], E[a1'', a1] || W(a1'') ∧ F(a1) ∧ P(p1'') }
7(1) + 1(2)

At this point, the environment is false as can be seen from Cohn's proof. However, in this constrained logic, the empty clause is required for a refutation. While this adds some extra steps, they are just simple resolutions utilizing the sorting functions that remove expressions from the clause without adding anything new to the constraints.

(9) { E[a1'', p1''] || W(a1'') ∧ P(p1'') }
8(2) + base information

(10) { □ || W(a1'') ∧ P(p1'') }
9(1) + base information

Now we have our empty clause and the proof to our problem. Notice that by using simplification to make moving through the proof easier, the constraints don't give the information needed to answer the problem. That is why there may be times where it is important to go back and create a complete constraint list for the solution. The information we are looking for is the constraint on a1. Looking back at step 8, the last step a1 appeared in, the constraint on a1 was F. Since a1 doesn't appear in any of the later constraints, it is not changed and is the final constraint on a1. Thus, the solution to this problem is F, or fox. This is precisely the answer desired from this problem, as given by Stickel [5] (an English explanation of the answer is also given).

This example does a good job of showing some of the reasons why an optimized constrained logic alternative to LLAMA provides better clarity and understanding. The same steps as in Cohn's proof were taken in the creation of this proof. However, in this reformulation, the answer to the original problem is much more apparent. If a complete constraint set is kept throughout the proof, the answer to the problem is contained in the constraint set of the empty clause.

It should be stressed that this optimized constrained logic reformulation of LLAMA has not been proven to be a complete deductive system. There are questions about it like whether or not multiple constraint sets could be attached to one clause.

CHAPTER 6

CONCLUSION

This thesis has demonstrated that it is possible to maintain the functionality and expressiveness of Cohn's Many Sorted Logic, LLAMA, through the use of Bürckert's constrained logic. While differences in what determines the final clause needed to prove a theorem cause the constrained logic to have more steps in the proof, this is partially offset by the lack of having to calculate sort arrays for each clause in the delta signature. The reason that the constrained logic can mimic LLAMA, so to speak, is that it allows predicates with more than one term to be added to the constraints of the clause. This allows relationships between sorts to be dealt with in the constraint theory, instead of having to explicitly deal with it in the clauses themselves. At this point, this reformulation is a happy medium between the efficiency of LLAMA and the understandability of the Substitutional Framework. The question remains whether or not it is possible to optimize this "new" logic to gain more of the benefits of LLAMA while maintaining the current level of understandability.

REFERENCES

1. Hans-Jürgen Bürckert, "A Resolution Principle for Constrained Logics," *Artificial Intelligence*, vol. 66, pp. 235 – 271, 1994.

2. Anthony G. Cohn, "A More Expressive Formulation of Many Sorted Logic," *Journal of Automated Reasoning*, vol. 3, pp. 113 – 200, 1987.

3. Anthony G. Cohn, "On the Appearance of Sortal Literals: a Non Substitutional Framework for Hybrid Reasoning," in *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, Canada, pp. 55 – 66, 1989.

4. Alan M. Frisch, "A General Framework for Sorted Deduction: Fundamental Results on Hybrid Reasoning," in *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, Canada, pp. 126 – 136, 1989.

5. Mark E. Stickel, "Schubert's Steamroller Problem: Formulations and Solutions," *Journal of Automated Reasoning*, vol. 2, pp. 89 – 101, 1996.