A Tableaux Based System for Propositional NonMonotonic Logics

Frank M. Brown

University of Kansas
Lawrence, Kansas, 66045
brown@eecs.ku.edu

Abstract. Different nonmonotonic logics are often viewed as being very different. This is true even though there are a number of interesting results comparing one nonmonotonic logic to another. Herein, we describe the commonality and differences among several different nonmonotonic logics by implementing their propositional logic versions with finite stream based algorithms involving maps and filters written in Scheme which maximize commonality and minimize differences. The result of organizing nonmonotonic computations in this fashion is to make apparent in an elementary way, that different nonmonotonic systems embody many of the same basic ideas and in fact differ by often only a few bits or pointers. The nonmonotonic systems investigated are the Closed World Assumption, the kernel of Autoepistemic Logic, Frame Logic, Default Logic, Justified Default Logic, Constrained Default Logic, and Parallel Circumscription. The Scheme code which produces all the fixedpoints for all these systems for all propositional problems is given.

1 Introduction

Growing out of the need for logics and automatic deduction systems for common sense reasoning phenomena such as default reasoning, reasoning about an agent's knowledge or lack thereof, and reasoning about the consequences of robotic actions, researchers have developed a number of logical systems generally called nonmonotonic logics. These various systems at first appeared to be very different partly because of the different formalisms involved and partly because the interrelationships were not at first understood or were misunderstood. However, there is now a number of known important relationships between the different nonmonotonic logics. These results are often fairly esoteric metatheorems which state that certain structures in one nonmonotonic logic have fixedpoint solutions that are related to the fixedpoint solutions produced by certain other structures in some other nonmonotonic system. Herein we take a different approach to comparing nonmonotonic systems. Our approach is to begin by studying the propositional structure of nonmonotonic systems while ignoring the quantificational structure. Later, we hope to extend this research by restoring widening ranges of quantificational structure to the propositional nonmonotonic logics studied herein. Because we here limit ourselves to propositional nonmonotonic logics everything is

finite and we will be able to present the different nonmonotonic systems as algorithmic code. This code, will define the propositional versions of these nonmonotonic systems by computing all the fixed points for any possible problem. By writing the code in the elegant fashion of (finite) stream programs involving generators, maps, filters, sieves, and accumulators, we are then able to extract out the differences among different nonmonotonic systems, thereby providing an interesting elementary way to compare them. The Scheme code for all these systems is given so that the reader may use it to experiment on their favorite nonmonotonic examples.

Section 2 presents a tableaux deduction algorithm for the Propositional Logic based on the Gentzen's LK Sequent Logic roughly equivalent to Wang's algorithm. This tableaux deduction system is used as a basic subroutine in all the different nonmonotonic systems studied herein. This is presented herein, so that all the code of our automatic deduction system for nonmonotonic logics will be available.

Section 3 presents the generic code for testing whether defaults hold. This code implements the basic ideas of testing defaults and generating possible fixedpoints which are common to all the nonmonotonic systems discussed herein. Since we are limiting this study to propositional logic the above tableaux algorithm in section 2 is decidable returning true or false. For this reason we don't need to represent the nonmonotonic default structure itself in a formal logic such as in the modal logic style of [Brown 1986] and [Brown 1989], nor in the tableaux style of [Bonatti, & Olivetti 2002] and [Olivetti 1992]. This results in a significant simplification of the deduction systems involved, allowing each to be described in a few lines of Scheme code thereby making these systems available to the many AI scientists who know LISP like languages but are not familiar with modal logics or sequent logic notations. For the Closed World Assumption, the kernel of Autoepistemic Logic, Default Logic, and Parallel Circumscription, the correctness of the algorithms embedded in the code follow from the modal representations of these systems given in [Brown 1989].

[Bonati & Olivetti 2002] suggest that "complete axiomatizations should explicitly model the very process of defeasible assumption making". We agree with this suggestion because a more sophisticated automatic theorem prover could potentially reuse intermediate deduced structures in other deductions thereby shortening proofs. Shorter proofs even at the cost of additional overheads for the explicit representations could still result in improved automatic deduction ability. In addition, from this viewpoint, their remark would have even more force when applied to non-decidable cases of nonmonotonic logics since the possibility for reuse would be greater due to the greater sophistication of the theorems involved. This is one reason we used the modal representations of [Brown 1986] and [Brown 1989] in proving theorems in the nondecidable cases of various nonmonotonic logics such Circumscription and in our quantified extensions to the Closed World Assumption, the kernel of Autoepistemic Logic and Default Logic. [Brown 1989, Brown & Araya 1991, Leasure 1993].

The remaining sections define the Propositional Logic versions of the Closed World Assumption (section 4), the kernel of Autoepistemic Logic (section 5), Frame Logic (section 6), Default Logic (section 7), Justified Default Logic (section 8), Constrained Default Logic (section 9) and Parallel Circumscription (section 10). Finally, some conclusions are drawn in section 11.

2 A Tableaux Sequent Calculus for Propositional Logic

The sentences of the Propositional Logic used herein are elementary sentences and any list structure obtained by replacing the variables p, q, p_1,...,p_n in one of the forms: #t, #f, (and p_1,...,p_n), (or p_1,...,p_n), (not p), (if p q), (iff p q) by other sentences. The elementary sentences are distinct symbols such as P or list structures not beginning with a logical symbol such as (Loves John Mary).

Conjectures are proven using Wang's Algorithm [McCarthy 1962] which is based on reversing the rules of Gentzen's LK Sequent logic. A sequent is essentially a list of hypotheses and a list of goals:

h_1,...,h_m => g_1,...,g_n

Commas are used to separate sentences within a sequent. The basic idea of this Tableaux Sequent Calculus is to try to prove a conjecture C from a set of axioms A_1,...,A_n represented as the sequent: A_1,...,A_n=>C by applying rules to the hypotheses and goals of each sequent. Each of these rules produce new sequents by replacing a sentence in a sequent by its immediate subparts thereby reducing the complexity. Each rule specifies that the sequent above the line is to be replaced by zero or more sequents below the line. The & sign is used to separate sequents. The sequent rules are given in Figure 1 and are organized into rules which manipulate hypotheses and rules which manipulate goals. In general, there is one hypothesis rule and one goal rule for each logical symbol. The atom rule given in the last line of the table says that a sequent holds if the same proposition occurs in opposite sides of the sequent. For example, to prove that q follows from the axioms p and (if p q) we apply the following steps to the sequent: p,(if p q)=>q. First, if=> is used to replace the initial sequent by two sequents p=>p,q and p,q=>q which are both eliminated using the atom rule twice. Since all sequents are eliminated the axioms imply the conjecture.

Table 1. The Tableaux Rules for Propositional Logic are as follows

name	hypothesis rule	name	goal rule
#t=>	#t => =>	=>#t	=> #t
#f=>	#f=>	=>#f	=>#f =>
and=>	(and p_1,....,p_n) => p_1,...,p_n =>	=>and	=>(and p_1,....,p_n) =>p_1 &...& =>p_n
or=>	(or p_1,....,p_n)=> p_1=> &...& p_n=>	=>or	=>(or p_1,....,p_n) => p_1,...,p_n
not=>	(not p)=> =>p	=>not	=>(not p) p=>
if=>	(if p q)=> =>p & q=>	=>if	=>(if p q) p=>q
iff=>	(iff p q)=> =>p,q & p,q=>	=>iff	=>(iff p q) p=>q & q=>p
atom	p => p		

A Scheme program implementing this Tableaux Sequent Calculus is given in Figure 1. The function => takes 4 arguments:

h -- a list of hypotheses g -- a list of goals

h* -- a list of elementary hypotheses g* -- a list of elementary goals

The rules work by removing one by one the logical connectives from the sentences in h and g and eventually placing all the simple sentences onto the lists h* and g*. The main function is prove whose arguments are a list of axioms h and a conjecture c. If no rule is applicable then the conjecture is not provable from the axioms.

```
(define(prove h c) (=> h '()'()(list c)))
(define(=> h h* g* g)
 (cond((not(null?(intersection h* g*))) #t)
  ((pair? h) (let((x(car h)))
     (cond((eqv? x #t) (=>(cdr h)h* g* g))
          ((eqv? x #f) #t)
          ((pair? x) (case(car x)
             ((and) (=>(append(cdr x)(cdr h))h* g* g))
             ((or) (forall(lambda(y)
                    (=>(cons y(cdr h))h* g* g))(cdr x)))
             ((not) (=>(cdr h)h* g*(cons(cadr x)g)))
             ((if)(and(=>(cons(caddr x)(cdr h))h* g* g)
                      (=>(cdr h)h* g*(cons(cadr x)g))))
             ((iff) (and(=>(list*(cadr x)(caddr x)
                                 (cdr h))h* g* g)
                        (=>(cdr h)h* g*
                           (list*(cadr x)(caddr x)g))))
             (else (=>(cdr h)(cons x h*)g* g))))
         (else (=>(cdr h)(cons x h*)g* g)))))
  ((pair? g) (let((x(car g)))
    (cond((eqv? x #t) #t)
          ((eqv? x #f) (=> h h* g*(cdr g)))
          ((pair? x) (case(car x)
            ((or) (=> h h* g*(append(cdr x)(cdr g))))
            ((and)(forall(lambda(y)
                  (=> h h* g*(cons y(cdr g))))(cdr x)))
            ((not) (=>(cons(cadr x)h)h* g*(cdr g)))
            ((if) (=>(cons(cadr x)h)h* g*
                             (cons(caddr x)(cdr g))))
            ((iff) (and(=>(cons(cadr x)h)h* g*
                           (cons(caddr x)(cdr g)))
                       (=>(cons(caddr x)h)h* g*
                           (cons(cadr x)(cdr g)))))
             (else (=> h h*(cons x g*)(cdr g)))))
          (else (=> h h*(cons x g*)(cdr g))))))
      (else #f)))
(define(forall p L)
       (if(pair? L)(and(p(car L))(forall p(cdr L)))#t))
```

```
(define(intersection a b)
       (filter(lambda(x)(member x b))a))
(define(filter p L)
     (accumulate(lambda(x M)(if(p x)(cons x M)M))'()L))
(define(accumulate c i L)
       (if(pair? L)(c(car L)(accumulate c i(cdr L)))i))
```

Fig. 1. Automatic Theorem Prover for Propositional Logic based on the Tableaux Sequent calculus. Calling the function prove with a list of axioms and a conjecture returns #t or #f depending on whether the conjecture follows from the list of axioms or not.

3 Nonmonotonic Default Inference Rules

Nonmonotonic inferences in Propositional Logic may be specified by non-logical inference rules called "defaults" of the form:

$$\frac{\alpha_1,...,\alpha_m : \beta_1,...,\beta_n}{\chi}$$

where $\alpha_1,...,\alpha_m$, $\beta_1,...,\beta_n$, and χ are sentences. Such a nonlogical inference rule is interpreted to mean that if each α_i holds in a given theory and if each β_i is consistent with respect to a given theory then χ may be inferred. If we suppose that the given theory is the same for each α_i sentence, then the inference rule may be rewritten as:

$$\frac{\alpha : \beta_1,...,\beta_n}{\chi}$$

where α is (and $\alpha_1...\alpha_m$) since α holds in a given theory if and only if each α_i holds in that theory. The case where there are no α_i sentences may then be represented by letting α be #t. However, if we suppose that the given theory is the same for each β_i sentence, the $\beta_1,...,\beta_n$ sentences cannot likewise be replaced by one large β since each β_i sentence may be consistent with a given theory without (and $\beta_1,...,\beta_n$) itself being consistent with the theory as, for example, in the theory (not(and p q)) where β_1 is p and β_2 is q. A nonlogical inference rule or default will be represented as a three element list of the form: $(\alpha(\beta_1 \ldots \beta_n)\chi)$

Assuming that the given theory is the same for all $\beta_1,\ldots,\beta_n$, a Scheme program to determine whether a nonlogical inference rule is applicable is given in Figure 2. The scheme function test takes as arguments: h1 – the given theory for α, h2 – the given theory for $\beta_1 \ldots \beta_n$, and – a default.

```
(define(test h1 h2 d)
       (define(pv x) (prove h1 x))
       (define(pos x) (not(prove h2(list 'not x))))
       (and(pv(car d))(forall pos(cadr d))))
```

Fig. 2. Procedure to determine whether a default is applicable.

This Scheme program assumes that the theories h1 and h2 are known before one tests any default. However, if either of the theories on which the α and $\beta_1,\ldots,\beta_n$ sentences of each default are tested itself includes the χ sentences of those defaults which fired, then we need to already know which χ sentences fired in order to test the α and $\beta_1,\ldots,\beta_n$ sentences. This circularity may be avoided by picking a subset of the set of defaults to be the fired subset and then testing that each default in the chosen set is applicable and that each default not in the chosen subset is not applicable. By successively choosing as the set of fired defaults each possible subset of the initial set of defaults we get all possible solutions.

A Scheme program called testall? which determines whether a subset s of the set of defaults Ds is a fired subset of defaults is given in Figure 3. This program calls test+ which checks to see that every default which should fire (i.e. those in s) does and a program: test- which checks to see that every default which is not supposed to fire (i.e. those not in s), do not. It also calls the program setdifference which computes the defaults which are not fired.

Also given in Figure 3 are the fp and subsets programs. The fp program produces a list of all the χ sentences of the fired defaults appended to the set of axioms As. The subsets program generates the list of all possible subsets of the set of defaults.

```
(define(testall? h1 h2 s Ds)
(and(test+ h1 h2 s)(test- h1 h2(set-difference Ds s))))
(define(test+ h1 h2 s)
       (forall(lambda(d)(test h1 h2 d))s))
(define(test- h1 h2 s)
       (forall(lambda(d)(not(test h1 h2 d)))s))
(define(set-difference Ds s)
       (filter(lambda(x)(not(memq x s)))Ds))
(define(fp As s)(append(map caddr s)As))
(define(subsets L)
    (accumulate
      (lambda(x m)(append(map(lambda(s)(cons x s))m)m))
     '(())L))
```

Fig. 3. Procedures to test whether a subset of defaults is a firing set of defaults.

4 The Closed World Assumption

The Closed World Assumption [Reiter 1978] (which is related to Completion systems [Clark 1978]) is a rule which allows the inference of a sentence of the form $\neg(\pi\ \delta_1...\delta_m)$ where π in an m-ary predicate and $\delta_1...\delta_m$ are variable free terms, whenever that sentence is possible with a given theory:

$$\frac{: \neg(\pi\ \delta_1...\delta_m)}{\neg(\pi\ \delta_1...\delta_m)}$$

Since no variables occur in $(\pi\ \delta_1...\delta_m)$ we may think of it as being a propositional constant χ.

$$\frac{:\chi}{\chi}$$

We generalize the closed world assumption so that there may be an α sentence, any number of β_i sentences which may be different from χ:

$$\frac{\alpha: \beta_1,\ldots,\beta_n}{\chi}$$

and interpret this structure to mean that there is a set of axioms As such that if α follows from As and each β_i is possible with As then χ. A Scheme program to compute the result of applying such defaults to an initial set of axioms is given in Figure 4. This program works by generating all the subsets of the defaults, filtering them by testing which defaults fire, and then by constructing the resulting fixedpoints.

```
(define(cwa-fixedpoints As Ds)
       (map(lambda(s)(fp As s))
           (filter(lambda(s)(testall? As As s Ds))
                  (subsets Ds))))
```

Fig. 4. Fixedpoint Deducer for The Closed World Assumption

Since the χ sentences of the defaults are not part of the set As which is being used to test the defaults, there is only one set of firing defaults. The resulting theory is simply the result of appending the χ sentences of the firing defaults to that set. For this reason there is a single cwa fixedpoint which may be computed by the more efficient program given in Figure 5 which simply tests each default to see if it fires and then constructs the fixedpoint from the axioms and the firing defaults.

```
 (define(cwa-fixedpoint As Ds)
        (fp As(filter(lambda(d)(test As As d))Ds)))
```

Fig. 5. Efficient Fixedpoint Deducer for the Closed World Assumption

Example 1: This example has one axiom: A and one default:

$$\frac{:B}{B}$$

The set of axioms is represented as a list of axioms `'(A)` and the set of defaults is represented as a list of defaults `'((#t(B)B))`. Since there is no α part of this default, it is represented as `#t`. To compute the fixedpoint we simply apply cwa-fixedpoint to the set of axioms and the set of defaults:

```
(cwa-fixed-point '(A) '((#t(B)B)))  => (A B)
```

Example 2: Here is an example illustrating the problem cwa has in dealing with two contradictory defaults:

$$\frac{:A}{A} \qquad \frac{:\neg A}{\neg A}$$

```
(cwa-fixedpoint '()'((#t(A)A)(#t((not A))(not A)))
=> (A(not A))
```

which is a fixedpoint from which `#f` may be derived by the prove function.

5 The Kernel of Autoepistemic Logic

Autoepistemic Logic [Moore 1985] for a Propositional Logic includes the syntax of Propositional Logic supplemented with a unary operation L which has the properties of an S4.5 modal logic. These modal properties allow each sentence of this Propositional Autoepistemic Logic to be rewritten [Konolige 1987] as a finite set of sentences of the form:

$$((L\alpha) \wedge (\neg L\neg\beta_1)\wedge ...\wedge(\neg L\neg\beta_n)) \rightarrow \chi$$

where the $(L\alpha)$ expression is optional in any sentence and n may be 0. Such a sentence can then be thought of as being the inference rule:

$$\frac{\alpha: \beta_1,...,\beta_n}{\chi}$$

where α follows from the given theory k and each β_i is possible with that theory k. The given theory k will always be a fixedpoint. The fixedpoints can be obtained by enumerating all possible subsets of defaults, filtering out those which are not firing subsets and conjoining all the χ sentences of the firing defaults to the initial axioms. A Scheme program to do this, called aek-fixedpoints, is given in Figure 6. [Bouix 1998] gives an analogous program written in Logistica.

```
(define(aek-fixedpoints As Ds)
 (map(lambda(s)(fp As s))
     (filter(lambda(s)(testall?(fp As s)(fp As s)s Ds))
            (subsets Ds))))
```

Fig. 6. Fixedpoint Deducer for the Kernel of Autoepistemic Logic

Since the χ sentences of the defaults are part of the set being used to test the defaults there is not necessarily a single fixedpoint as in the case of cwa. For example, the empty set of axioms and the default $A:/\neg A$ has no fixedpoints. Likewise the empty set of axioms and the two defaults: $:A/A$ and $:\neg A/\neg A$ has two fixedpoints, namely one consisting of A and the other consisting of $\neg A$

Aek is a powerful logic capable of representing, as defaults rules, action logics involving both precondition/result pairs of actions and the necessary frame laws:

$$\frac{\text{preconditions-action(t):}}{\text{results-action (t+1)}} \qquad \frac{\alpha(t): \chi(t+1)}{\chi(t+1)}$$

In these laws t is a specific number representing the time (represented as an additional argument to each predicate in the sentence) at which the action is applied. The first default states that the results of an action applied at time t holds at time t+1 if the preconditions held at the previous moment of time t. The second law is the frame law which states that a property α which holds at time t causes a property χ to holds at time t+1 if it is consistent for χ to do so. In the simpler cases (usually discussed in the literature) α is identical to χ.

6 Frame Logic

Frame Logic [Brown 1987] represents in a direct manner action logics involving both the precondition/result pairs of actions and the necessary frame laws. A Propositional Frame Logic involves a set of axioms and default laws representing precondition/action pairs and the necessary frame laws of the following forms:

preconditions-action: $\frac{\alpha: \chi}{\chi}$

results-action

Unlike Autoepistemic Logic (aek) no numeric subscripts representing time are needed because the sentences before the : in any default rule follow from a given theory representing what holds at the preceding moment in time. Let OLD be the theory which specifies what holds in the previous moment of time. An action law then says that if the preconditions held in OLD then the results hold in the fixedpoint (which represents what now holds). Likewise, the frame laws say that a property α which holds in OLD causes a property χ to hold in the fixedpoint if χ is possible. Again in the simpler cases (usually discussed in the literature) α is identical to χ. [Brown 1989] discusses more sophisticated cases dealing with Newtonian Mechanics.

We generalize Frame Logic defaults to:

$$\frac{\alpha: \beta_1,\ldots,\beta_n}{\chi}$$

where α holds in the old theory and each β_i is consistent with a resulting fixedpoint which includes the χ sentences of the firing defaults. The fixedpoints can be obtained by enumerating all possible subsets of the defaults, filtering out those which are not firing subsets and conjoining all the χ sentences of the firing defaults to the initial axioms. A Scheme program to do this, called frame-fixedpoints, is given in Figure 7.

```
(define(frame-fixedpoints old As Ds)
       (map(lambda(s)(fp As s))
          (filter(lambda(s)(testall? old(fp As s)s Ds))
                 (subsets Ds))))
```

Fig. 7. Fixedpoint Deducer for Frame Logic

7 Default Logic

Default Logic [Reiter 1980] for a Propositional Logic is essentially a set of axioms As and a finite set of default rules of the form:

$$\frac{\alpha: \beta_1,\ldots,\beta_n}{\chi}$$

where α follows from a theory k and each β_i is possible with that theory k. The theory fixedpoint k must be constructable by adding to the axioms the χ sentences of those defaults whose a sentences are already deducible from the axioms and previously deduced χ sentences. The requirement does not allow the α sentence of a default to be proven by using its own χ sentence. The supported nature of Default Logic, perhaps, more closely represents defaults such as those used in taxonomies and

other cases than do logics such as Autoepistemic logic. Other authors prefer other Default Logics such as Justified Default Logic or Constrained Default Logic. [Antionu 1997] discusses the merits of different alternatives.

The fixedpoints of a system of defaults of Default logic can be obtained by enumerating all possible subsets of defaults, filtering out those which are not firing subsets and those which are no properly constructed and then conjoining all the χ sentences of the firing defaults to the initial axioms. Since Default logic as defined here differs from Autoepistemic kernels by a single filter (i.e. the supported filter) it is obvious that the fixedpoints of Default Logic are a subset of the fixedpoints of Autoepistemic Kernels as Konolige suggested and eventually proved for the original descriptions of these logics. (see [Konolige 1987]). A Scheme program to derive all the fixedpoints, called dl-fixedpoints, is given in Figure 8.

```
(define(dl-fixedpoints As Ds)
  (map(lambda(s)(fp As s))
    (filter(lambda(s)(supported? As s))
     (filter(lambda(s)(testall?(fp As s)(fp As s)s Ds))
            (subsets Ds)))))
(define(supported? As s)
   (define(iter rn s)
      (define fdl(filter(lambda(d)(prove rn(car d)))s))
      (if(null? fdl)s(iter(append(map caddr fdl)rn)
                          (set-difference s fdl))))
   (null?(iter As s)))
```

Fig. 8. Fixedpoint Deducer for Default Logic

There are many possible improvements to this program. For example, since the order of applying filters does not essentially change the result of stream code, we may swap the order sometimes producing a profound effect on efficiency. For example, a new program for computing the fixedpoints of Default Logic could be obtained by swapping the order of the two filters in Figure 8.

[Risch & Schwind 1990] describes an automatic procedure for computing fixedpoints for Default Logic as originally defined whereby each default contains at least one possibility. Thus their algorithm does not allow defaults such as:

preconditions-action(t): α:
results-action (t+1) (not α)

though the default :

preconditions-action(t): #t
results-action (t+1)

might serve as a reasonable facsimile in the former case. This work is formulated to apply to First Order Logic with a finite domain which as is well known is essentially equivalent to propositional logic. Their algorithm has three main steps: (1)The first step is to enumerate the maximally consistent subsets or essentially enumerating the subsets of the defaults and then filtering them by a test determining which are maximally consistent with the axioms. (2) The second step tests the possibility statements and appears to be a leaner version of our testall filter which ignores testing the α sentences, and the third step is analogous to our supported? filter. In addition to

these three steps, they state that "it still has to be verified that extensions are minimal and do not contain each other". This suggests that there is also a sieve like process such as the one we use in defining Justified Default logic in section 8 below. We could not determine whether their algorithm is more efficient than the one given herein. Their generation step (step 1) involving a filter appears to be more expensive, their step 2 appears to be less expensive than testall, their step 3 is the same as our supported filter, and finally they have the additional cost of the above mentioned sieve.

[Turner 1996] suggests ways of reducing the complexity of default logic by dividing defaults into a sequence of layers so that only the defaults of a given layer need be considered at any given time. This work applies only to those default structures and axioms which fit certain syntactic criteria. Thus, this work is not an algorithm in itself for Default logic, but may serve as a valuable adjunct to such algorithms.

8 Justified Default Logic

Justified Default Logic [Lukaszewics 1990] for a Propositional Logic is essentially a set of axioms As and a finite set of default rules of the form:

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\chi}$$

where α follows from a theory k and each β_i is possible with that theory k and with every other B sentence in every other default which fired. The fixedpoints of a system of defaults can be obtained by enumerating all possible subsets of defaults, filtering out those firing defaults which do not fire (using TEST+) filtering out those which are not supported, eliminating any fixedpoints with subsumed sets of firing defaults and conjoining all the χ sentences of the firing defaults to the initial axioms. A Scheme program to do this, called jdl-fixedpoints, is given in Figure 10.

```
(define(jdl-fixedpoints As Ds)
  (map(lambda(s)(fp As s))
    (sieve not-contains?
       (filter(lambda(s)(supported? As s))
           (filter(lambda(s)(test+(fp As s)(fp As s)s))
                  (subsets Ds))))))
(define(sieve p s)
       (accumulate(lambda(x L)(cons x(sieve p
                  (filter(lambda(y)(p x y))L))))'()s))
(define(not-contains? x y)
       (not(null?(set-difference y x))))
```

Fig. 10. Fixedpoint Deducer for Justified Default Logic

`sieve` is an abstraction of the Sieve of Eratosthenes. When applied to `(lambda(x y)(not(=(remainder y x)0)))` and a stream of consecutive ascending integers starting from 2, it produces the prime numbers.

9 Constrained Default Logic

Constrained Default Logic [Schaub 1992] for a Propositional Logic is essentially a set of axioms As and a finite set of default rules of the form:

$$\frac{\alpha : \beta_1,\ldots,\beta_n}{\chi}$$

where α follows from a theory k and each β_i is possible with that theory k and with every other B sentence in every other default which fired. The fixedpoints of a system of defaults can be obtained by enumerating all possible subsets of defaults, filtering out those firing defaults which do not fire (using TEST+) whose β theory includes the β sentences of all the firing defaults conjoined to the fixedpoint, filtering out those which are not supported, eliminating any fixedpoints with subsumed sets of firing defaults and conjoining all the χ sentences of the firing defaults to the initial axioms. A Scheme program to do this, called cdl-fixedpoints, is given in Figure 11.

```
(define(cdl-fixedpoints As Ds)
  (map(lambda(s)(fp As s))
      (sieve not-contains?
         (filter(lambda(s)(supported? As s))
             (filter(lambda(s)(test+(fp As s)
                   (append(flatmap cadr s)(fp As s))s))
                 (subsets Ds))))))
(define(flatmap f L) (flatten(map f L)))
(define(flatten L)(accumulate append '()L))
```

Fig. 11. Fixedpoint Deducer for Constrained Default Logic

10 Circumscription

A nonmonotonic system such as the Closed World Assumption produces precisely one fixedpoint. In such a case one may determine that a conjecture follows from that fixed point by simply applying the prove function to the fixedpoint and the conjecture. However, most nonmonotonic systems do not always produce precisely one fixedpoint. In such a case the question arises as to what fixedpoint should be used to derive further consequences. We could choose arbitrarily but a more conservative approach is to require that the theorem hold in all fixedpoints. Since $(fp_i \Rightarrow t)$ & ... & $(fp_n \Rightarrow t)$ is equivalent to $(fp_1$ or ... or $fp_n) \Rightarrow t$ we need to prove t from the disjunction of all the fixedpoints. From this perspective the problem is to compute this disjunction. It turns out that there is a nonmonotonic system, namely Parallel Circumscription [McCarthy 1980, McCarthy 1986, Lifschitz 1985] that produces just this disjunction [Konolige 1989, Brown 1989] when all the defaults are of the form:

$$\frac{:\chi}{\chi}$$

Specifically, Circumscription for a Propositional Logic may be thought of as being a rule which allows the inference of sentences of the form $\neg(\pi\ \delta_1...\delta_m)$ where π in an m-ary predicate and $\delta_1...\delta_m$ are variable free terms, whenever that sentence is possible with a given theory:

$$\frac{: \neg(\pi\ \delta_1...\delta_m)}{\neg(\pi\ \delta_1...\delta_m)}$$

such a predicate is said to be Circumscribed. Since no variables occur in $(\pi\ \delta_1...\delta_m)$ we may think of it as being a propositional constant. In addition to Circumscribed Predicates, there may also be Variable Predicates and Fixed Predicates. Variable Predicates involve no additional rules, but Fixed Predicates involve two contradictory rules of the form:

$$\frac{: \neg(\pi\ \delta_1...\delta_m)}{\neg(\pi\ \delta_1...\delta_m)} \qquad \frac{: (\pi\ \delta_1...\delta_m)}{(\pi\ \delta_1...\delta_m)}$$

These default structures may be interpreted as an Autoepistemic Kernel structure, a Frame Logic structure, or as a Default Logic structure since all three interpretations are the same for the defaults used in Circumscription.

We generalize the defaults of circumscription so that there may be an α sentence, any number of β_i sentences, and so that last occurrences of χ may be any sentence:

$$\frac{\alpha: \beta_1,...,\beta_n}{\chi}$$

A Scheme program for Circumscription by interpreting its defaults as Autoepistemic Kernel defaults is given in Figure 9. This program accumulates the fixedpoints together by simply returning the disjunction of the conjunction of all the sentences in each Autoepistemic Kernel fixedpoint (see [Brown 1989, Konolige 1989]).

```
(define(circumscription As Ds)
        (cons 'or(map(lambda(k)(cons 'and k))
                     (aek-fixedpoints As Ds))))
```

Fig. 12. Automatic Theory Constructor for Parallel Circumscription with circumscribed, fixed, and variable predicates.

11 Conclusion

The nonmonotonic systems discussed herein are summarized in Table 2 in terms of the filters, sieves, and accumulators that are used. The main choice is whether testall or test+ is used and in either case what theories are used to test the α sentences and β sentences of the defaults. There is also the question of whether the supported? filter is used, whether the non-contained? sieve is used, and whether the disjunction accumulator is used. Thus in summary, the difference in all these nonmonotonic systems amounts to two pointers (for the α and β theories), and four bits (for testall vs test+, supported?, the not-contains? sieve, and disjunction accumulator. Table 2 suggests the existence other possible nonmonotonic systems with different combinations of pointers and bits.

Table 2. The differences among the different NonMonotonic Systems. As is the initial set of axioms, k is the fixedpoint, and kβ is the conjunction of the axioms in k and each of the β sentences of all firing defaults.

nonmonotonic system	**filters**	**sieve**	**accum-ulator:**
Closed World Assumption	λs(testall As As s)		
Autoepistemic Kernel	λs(testall k k s)		
Frame Logic	λs(testall old k s)		
Default Logic	λs(testall k k s), λs(supported? As s)		
Justified Default Logic	λs(test+ k k s), λs(supported? As s)	not-contains?	
Constrained Default logic	λs(test+ k kβ s), λs(supported? As s)	not-contains?	
Circumscription	λs(testall k k s)		disjunc-tion

Sometimes one only wants a single fixedpoint solution rather than all fixedpoint solutions. This is elegantly achieved by delaying the second argument to each cons used on a stream. In this case, each cons, cdr, and car (including the cons in Scheme's built-in functions such as the map function) used to implement a stream is replaced by cons-stream, cdr-stream, and car-stream defined as follows:

(cons-stream x s) =df (cons x(lambda() s))
(cdr-stream x s) =df (cdr(s))
(car-stream s) =df (car s)

Calling one of these nonmonotonic systems (except Circumscription which does an accumulation of all fixedpoints) then produces a delayed stream of fixedpoints whose car is the first fixedpoint found. The work to search for any additional fixedpoints would then be completely avoided.

Acknowledgements

This research was supported by grants EIA-9818341 and EIA-9972843 from the National Science Foundation. I thank Guy Jacobs, Greg Seibel, and the anonymous reviewers for their comments.

References

[Antoniou 1997] Antoniou, Grigoris 1997. *NonMonotonic Reasoning*, MIT Press.

[Bonatti, & Olivetti 2002] , Nicola, Bonatti, Piero Andrea & Olivetti, Nicola, "Sequent Calculi for Propositional Nonmonotonic Logics", ACM Transactions on Computational Logic, Vol. 3, noNo. 2, April 2002, p226-278.

[Bouix 1998] Bouix, Sylvain, "Solving Autoepistemic Equivalences for Propositional logic", May 7, 1998. personal notes.

[Brown 1986] Brown, F.M., "A Commonsense Theory of NonMonotonic Reasoning", *Proceedings of the 8h International Conference on Automated Deduction, CADE8,* Lecture Notes in Computer Science, vol 230, Oxford, England, July 1986, Springer Verlag, New York.[Brown 1986

[Brown 1987] Brown, Frank M. 1987. "The Modal Logic Z", In *The Frame Problem in AI"; Proc. of the 1987 AAAI Workshop*, Morgan Kaufmann, Los Altos, CA.

[Brown 1989] Brown, Frank M. 1989. "The Modal Quantificational Logic Z Applied to the Frame Problem", advanced paper *First International Workshop on Human & Machine Cognition*, May 1989 Pensacola, Florida. Abreviated version published in *International Journal of Expert Systems Research and Applications, Special Issue: The Frame Problem. Part A*. eds. Keneth Ford and Pattrick Hayes, vol. 3 number 3, pp169-206 JAI Press 1990. Reprinted in *Reasoning Agents in a Dynamic World: The Frame problem*, editors: Kenneth M. Ford, Patrick J. Hayes, JAI Press 1991.

[Brown & Araya 1991] Brown,Frank M.& Carlos Araya, "A Deductive System for Theories of NonMonotonic Reasoning," *Transactions of the Eighth Army Conference on Applied Mathematics and Computing, 1991*.

[Clark 1978], Clark, Keith 1978. "Negation as Failure", in, *Logic and Databases*, eds. Gallaire and Minker, J., Plenum Press, Ney York, p-293-322.

[Konolige 1987] Konolige, Kurt 1987, "On the Relation between Default Theories and Autoepistemic Logic", IJCAI87.

[Konolige 1989] Konolige, Kurt 1989. "On the Relation between Autoepistemic Logic and Circumscription Preliminary Report", IJCAI89.

[Leasure 1993] Leasure David, "A Logistica Deduction System for Solving NonMonotonic Reasoning Problems Using the Modal Logic Z" Ph.D dissertation, University of Kansas, 1993.

[Lifschitz, 1985] Lifschitz, V. 1985. "Computing Circumscription" *IJCAI 9*,pages 121-127.

[Lukaszewics 1990] Lukazszewics, W. 1990. "Considerations on Default logic", Computational Intelligence 4 pgs1-16.

[McCarthy 1962] McCarthy et. al. 1962. "Lisp 1.5 Programmer's Manual" MIT Press, second edition 1985.

[McCarthy 1080] McCarthy, J "Circumscription -- A Form of Nonmonotonic Reasoning", *Artificial Intelligence*, volume 13. 1980.

[McCarthy 1986] McCarthy, J. 1986. "Applications of Circumscription to Formalizing Common-Sense Reasoning", *Artificial Intelligence*, vol. 28.

[Moore 1985] Moore, R. C. 1985. "Semantical Considerations on Nonmonotonic Logic" *Artificial Intelligence*, 25.

[Olivetti 1992] Olivetti, Nicola 1992, "Tableaux and Sequent Calculas for Minimal Entailment", Journal of Automated Reasoniong 9: 99-139.

[Reiter 1978] Reiter, R. 1978. "On Closed World Databases", in, *Logic and Databases*, eds. Gallaire and Minker, J., Plenum Press, Ney York.

[Reiter 1980] Reiter, R. 1980. "A Logic for Default Reasoning" *Artificial Intelligence*, 13.

[Schaub 1992] Schaub, T. 1992. "On Constrained Default Theories", Proc 10th European Conference on Artificial ntelligence, Wiley pgs304-305.

[Schwind 1990] Schwind, Camilla 1990. "A Tableaux-Based Theorem Prover for a Decidable Subset of Default Logic", 10th International Conference on Automated Deduction, Kaiserslautern. Springer Verlag. Lecture Notes in AI vol 449.

[Turner, 1996] Turner, Hudson, 1996, "Splitting a Default Theory"., "Proceedings of the thirteenth National Conference on Artificial Intelligence, p645-651, AAAI Press, 1996.