Intuitionistic Logic in Business Systems

William S. Miles

Abstract— **Business computer systems model the real world where the absolute truth cannot always be established. This can lead to different points of view and a problem with client satisfaction. We consider intuitionistic logic as a more natural way for a business system to implement situational reasoning and we show how a proposition and its refutation can be used to develop both a specialized and a generalized extension to a rule-based business model.**

Index Terms— **Intuitionistic logic, Business systems**

I. Introduction

BUSINESS depends on client satisfaction for continuing success and this is why corporations invest time and effort to improve their customer service. Unfortunately, information systems developed to support new service offerings are often designed around business models that are incomplete and incapable of describing all situations. Not surprisingly, their ability to deal with unique client needs and changing business requirements is often less than satisfactory.

We propose that, ideally, business information systems should be designed to be responsive to many different situations rather than simply following one fixed set of policies and procedures. Business rules should adapt to the situation at hand. With inflexible rules a business model can make poor decisions in situations where unique circumstances prevail.

In this paper we show how intuitionistic logic can be used to develop an extensible reasoning system which supports the ideas of argument and refutation as a natural way to address situational differences. We show how a proposition and its refutation can lead to both a specialized and a generalized extension of a rule-based business model.

The generalized extension attacks the problem of business system inflexibility by providing a way to add new knowledge to the system without restricting the original model rules. The specialized extension attacks the problem by refining the business model by using new knowledge to constrain the solution space.

We approach the situational dependency problem by modifying the computational evaluation of an `If ... Then ...` statement. We use the ideas of intuitionistic logic to establish the validity of the rule implication during automated program execution. The first part of this paper presents an elementary review of logic and a conceptual argument for the use of refutation within business applications. The remaining sections discuss an implementation of this technique in a typical business application.

W. Miles is a Ph.D. student at the University of New Brunswick. E-mail: wmiles@unb.ca

II. Business Systems

Knowledge in business systems is frequently represented through the use of business rules [8]. A business rule, in the context of this paper, is defined as a specific policy or procedure that describes how an enterprise carries out its business. For example, financial rules might require that the interest rate set on a bank loan must not be negative or that the loan must be either secured or partially secured. Similarly, the sale of a commodity may require that the customer have an approved line of credit established before the sale can take place. Business rules are definitive rules which frequently act as constraints or filters to the data, thus ensuring that only valid business decisions are made.

Business rules, when expressed in an executable form, are written as:

`If` *`condition`* `Then` *`consequent`*

These rules are condition-action pairs that relate necessary business conditions to a business action. The action of applying a business rule or a series of business rules to a situation confirms or denies a business decision.

However, there are two problems that can occur with this type of rule development. The first, as noted by Minsky [6], is that the rules are often developed to describe what to do, as opposed to what not to do. Minsky recognized that we tend to think of our knowledge in a positive way, such that `If X` happens, `Do Y`. But this can ignore ‘negative’ knowledge which can be used to refute certain courses of action, such as `If` the loan applicant has a gun, `Don't` deny the loan.

In practice, many business systems encode only positive knowledge about the world and use this to filter the data into something that satisfies the business model constraints. A direct proof or justification for any business decision can easily be produced by simply showing that the known data does not violate any policy or rule. Conversely, if the rule base describes what not to do, then this makes the proof of the solution more difficult because a direct deductive resolution chain from the premise set to the conclusion may not exist. The solution is what is left over after all other options have been rejected.

The second problem pertains to the context within which we reason, or more precisely, the truth relationship between the business conditions and the real world. Business policies are often developed which assume only one situation or point of view of the world. We tend to think in terms of the system owner’s context, such as `If` payment received late, `Then` bad pay history. But this can ignore the client’s point of view, such as `If` payment made on time, `Then` good pay history. Such a situation can easily occur if some fault occurs with the payment transaction and the payment date differs from the receipt date. In this case both points of view can be correct depending on the assumed context of the situation. Unfortunately, business systems do not, as a matter of course, deal very well with contradictory assumptions. This can lead to a problem with client satisfaction.

Business systems often reason within a closed world which assumes that any fact that is not known to be true is assumed to be false. If a form is not completely filled in then it is either rejected outright or default values are assumed for the missing fields. A direct proof or justification for any business action can easily be produced by simply showing that the current facts do not violate any known rule *within the assumed context of the system*.

However, no rule base can be complete or sufficient to cover all possible situations when the domain of knowledge over which the rules apply is vague. This will always be the case when we are dealing with real-world situations. For example, assume that a customer purchases a product and then successfully argues for the seller to accept its return because the product was defective. In this situation either the business policies for the original sale did not explicitly consider the product condition as a necessary factor contributing to the sale, or the knowledge base assumed that the product was in a saleable condition. In the first case the business logic did not describe the complete situation and in the second case a default assumption was not true. In each case a piece of the domain knowledge was missing.

III. TRUTH

Truth is an attribute of things that are known, but the business system has to deal with situations where absolute truth cannot always be established. If a loan applicant once missed a loan payment and was now making an application for a new loan, and if the business system had a policy to deny new applications if the applicant had a poor loan payment history, then the decision can be taken not to approve the new loan. This decision may not be an optimum business decision, particularly if the payment was made on time and just happened to arrive late due to a postal disruption. The problem is that the business system cannot know all the facts of the situation and thus the absolute truth.

The truth of a proposition is a relationship between the proposition and reality and this relationship depends on the context in which the real world is viewed. Unfortunately, business information systems reason and routinely make decisions on the validity, eligibility, and classification of data using rules and facts which either directly or indirectly assume a single context for establishing truth. Reasoning uses context-free logic rules, frequently embedded in the code, which are of the form:

```
If premise Then conclusion
```

The rule is valid if the conclusion Q is not false when the premise P is true.

But what is the meaning of the statement: P is true? We propose that truth in computer information systems should be viewed not in terms of an absolute belief that data is true in all contexts but instead as real-world evidence that a fact or premise has been confirmed in some context-dependent situation.

For example, given two facts A and B, the proposition $A \leftrightarrow B$ [1] is often implemented in computer systems as the predicate `equals(A,B)` and treated as a boolean-valued function during execution, returning the value 0 or 1. The proposition is true in the sense that it is confirmed, because the value or representation for A has been examined and verified to be equal to B. The returned function value 1 (or T) is a confirmation of the proposition, and 0 (or F) is a refutation of the proposition.

Computer programs routinely confirm or refute propositions rather than establish their absolute truth. Computer programs implement the view espoused by the intuitionistic logicians who believe that truth cannot be known absolutely and instead suggest that logic semantics should be based on direct experience or evidence that a proposition has been confirmed or refuted.

We propose that if our knowledge base is known to be incomplete and if the truth of our known knowledge cannot be assumed to be absolute, then we should base our business reasoning on a logic that requires constructive proofs.

IV. INTUITIONISTIC LOGIC

Intuitionistic logic[2] replaces the traditional concept of truth with the idea of confirmation. Brouwer, the originator of intuitionistic logic, held that mathematical objects such as numbers, sets, functions, and so on exist because we construct them or show the means for their construction. Mathematical propositions, then, are not true in some absolute sense, but are only confirmable, refutable or neither as demonstrated by our proofs, derivations, and calculations [2].

Not every proposition can be confirmed or refuted. To follow Brouwer's example, the proposition that there is a distinct sequence of digits such as 0123456789 in the decimal expansion of π $(3.14159\cdots)$ cannot, in general, be refuted. We may be able to confirm the proposition, but if we cannot find this sequence then we may have to search through an infinity of digits to refute the proposition. This is impossible. Other undecidable problems are similar, such as Turing's halting problem which asks us to determine if an arbitrary program will halt or run forever, or Russell's Paradox which considers the set of all sets that are not members of themselves and then asks whether this set is a member of itself. Each of these problems poses a question that must be answered as yes or

[1]See Table I for a review of logic notation.

[2]See [7], pp. 427-438. My presentation in this section is based on this material.

TABLE I
LOGIC NOTATION

Symbol	Meaning
$\leftrightarrow$	Equivalence
$\rightarrow$	Implication
$\vdash$	Entailment
$\wedge$	And
$\vee$	Or
$\sim$	Not

no, yet no such answer can be given. The proposition cannot always be confirmed or refuted.

Thus, there may not always be sufficient evidence to confirm that the disjunction $P \vee \sim P$ is valid, or always true, in an intuitionistic sense. Intuitionistic logic rejects the law of excluded middle which says that every proposition P can be either confirmed or refuted. When this law does not hold it is equivalent to having two contradictory propositions both being simultaneously true. When such a situation occurs there is an apparent paradox.

Considering our loan example once more, it is paradoxical to consider that the loan payment was both late and not late. The paradox arises because there are different contexts within which the proposition can be proved. In the context of the business system the payment was late. In the context of the applicant the payment was not late. The truth of the proposition depends on the relationship between the proposition and the context within which the real world is viewed.

Intuitionistic logic is a two-valued logic as shown in Figure 1.

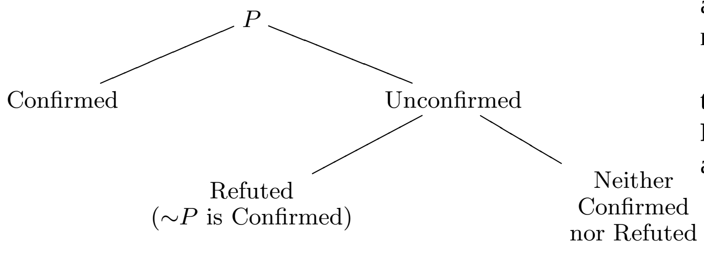

Fig. 1: Intuitionistic Logic Values

For another example, consider the logic of double negation. The sequent $\sim\sim P \vdash P$ is not intuitionally valid. A negation of P is a refutation of P. Therefore, $\sim\sim P$ is read as "it is refuted that P is refuted". This is not the same as confirming or proving P. If we can show that we will never have a proof that P will never be proved, this does not amount to a proof of P. From $\sim\sim P$ an intuitionist may not in general infer P.

To see this, suppose that that the proposition is made that Jonah, after being swallowed by a whale, is dead. A life insurance agency refutes this proposition by arguing that death cannot be confirmed without a body. If a beneficiary can prove that the whale swam away and thus a body will never be found, the beneficiary's refutation of the agency's argument has not, of necessity, confirmed that Jonah is truly deceased. A proof of the proposition is still required.

Semantic Interpretation of Intuitionistic Logic

Intuitionistic logic semantics differ from classical logic. Classical logic does not provide a way to easily interpret $P \rightarrow Q$, the relationship between the truth of P and the truth of Q, in different contexts. It is impossible to find a classic valuation that would make $P \rightarrow \sim\sim P$ valid, but not $\sim\sim P \rightarrow P$. How do we imagine a valuation where $\sim\sim P$ is true but P is false?

But with intuitionistic logic, all that $\sim\sim P$ means is that we have a refutation of a refutation of P. Or, in other words, it is absurd to suppose that P is unconfirmed. There is no need to actually provide a specific instance of a proof of P. P in our specific context can remain unconfirmed.

Similarly, to assert $P \rightarrow Q$ is to state that in any context where P can be confirmed, Q can also be confirmed. In other words, if there is a proof of P then there is a way to turn the proof of P into a proof of Q.

The biconditional $P \leftrightarrow Q$ remains a conjunction of the conditionals $P \rightarrow Q$ and $Q \rightarrow P$. The biconditional states that there is no context or evidential state which can distinguish between the two.

Intuitionistic semantics for $P \wedge Q$ and $P \vee Q$ are the same as classical semantics except that the values of T and F have been replaced by the notion of a proof. $P \wedge Q$ is confirmed if there is a proof of P and a proof of Q. Similarly, $P \vee Q$ is confirmed if there is a proof of P or a proof of Q.

V. Refutation in Business Systems

A natural way to attack the problem that we cannot know all the facts and thus the absolute truth is to utilize an extendable business system logic which incorporates the notion of refutation and uncertainty to establish truth.

Consider the loan application example discussed earlier. If the logic for this example is stated in terms of rules **R** and facts **F**, then one representation of this example can be expressed as shown in Figure 2.

R1	If $\sim GoodPayHistory$ Then $\sim ApproveLoan$
R2	If $LatePayment$ Then $\sim GoodPayHistory$
F1	$LatePayment$

Fig. 2: Model 1

The logic model is evaluated bottom up in the sequence F1, R2, R1. If we believe the $LatePayment$ fact F1 to be confirmed, then by rule R2 and the deductive process we must believe that this is a refutation of $GoodPayHistory$. Consequently by rule R1 $ApproveLoan$ is unconfirmed.

Now, in a business system that supports refutation, consider that the real world truth behind the late payment fact is challenged by the client with evidence of a postal service disruption. It is a business decision to consider if this argument is valid. If the argument is pursued then we add the new postal disruption fact as fact F2 to our knowledge base and revise our logic appropriately. For a $LatePayment$ fact to be valid, $PostalDisruption$ must be unconfirmed, either through an explicit refutation or lack of knowledge. Figure 3 shows how Rule R2 can be rewritten as rule R3 to reflect this logic.

R1	If $\sim GoodPayHistory$ Then $\sim ApproveLoan$
R3	If $LatePayment \wedge \sim PostalDisruption$
	Then $\sim GoodPayHistory$
F1	$LatePayment$
F2	$PostalDisruption$

Fig. 3: Model 2

If the $LatePayment$ fact and $PostalDisruption$ facts are confirmed, then by rule R3 we can no longer refute

$GoodPayHistory$ and thus cannot provide a refutation for $ApproveLoan$. The client has successfully argued his case.

However, if the postal service disruption argument is refuted by the business with a new argument that claims courier service was an alternate means for the client to pay on time, then we can introduce a new $Courier$ fact into our model as fact F3. Figure 4 shows how the model can be revised to reflect the logic by rewriting rule R3 as rule R4.

R1	If $\sim GoodPayHistory$ Then $\sim ApproveLoan$
R4	If $LatePayment \wedge \sim(PostalDisruption \wedge \sim Courier)$ Then $\sim GoodPayHistory$
F1	$LatePayment$
F2	$PostalDisruption$
F3	$Courier$

Fig. 4: Model 3

In this case the $Courier$ fact refutes $PostalDisruption$. By rule R4, because $LatePayment$ is confirmed we can now refute $GoodPayHistory$. By rule R1 we can now refute $ApproveLoan$.

These steps have explicitly modelled the argument and counter-argument reasoning process. Each argument extended the business model by adding new situational knowledge to the set of facts. The rule base is refined in a specialized way to lead to the correct solution.

Specialized Refutation Method

In general, every proposition P can have a refutation P' such that $P \wedge \sim P' \rightarrow P$. We call this the *conjunctive* axiom. The conjunctive axiom asserts that a proposition is valid if it does not have a refutation. The application of this axiom gave the rule evolution R2 to R3 to R4.

Let P = $LatePayment$
Let P' = $PostalDisruption$ be a refutation of P
Let P'' = $Courier$ be a refutation of P'
Let Q = $GoodPayHistory$
Let R = $ApproveLoan$

Then, Model 1 is expressed symbolically as:

R1:	$\sim Q \rightarrow \sim R$
R2:	$P \rightarrow \sim Q$
F1:	P

Now, introduce P' as a refutation for P. By the conjunctive axiom we have

$$P \wedge \sim P' \rightarrow P$$

Applying this to rule R2 gives

$$(P \wedge \sim P') \rightarrow P \rightarrow \sim Q$$

Through the interpretation of conjunction we can simplify and rewrite rule R2 as new rule R3: $P \wedge \sim P' \rightarrow \sim Q$. We have now derived Model 2, expressed symbolically as:

R1:	$\sim Q \rightarrow \sim R$
R3:	$P \wedge \sim P' \rightarrow \sim Q$
F1:	P
F2:	P'

Now, introduce P'' as a refutation for P'. By the conjunctive axiom we have

$$P' \wedge \sim P'' \rightarrow P'$$

Applying this to rule R3 and simplifying as before gives new rule R4: $P \wedge \sim(P' \wedge \sim P'') \rightarrow \sim Q$. We have now derived Model 3, expressed as:

R1:	$\sim Q \rightarrow \sim R$
R4:	$P \wedge \sim(P' \wedge \sim P'') \rightarrow \sim Q$
F1:	P
F2:	P'
F3:	P''

Generalized Refutation Method

A refutation may also be viewed as a way of confirming $\sim P$. We have $\sim P \vee P' \rightarrow \sim P$. We call this the *disjunctive* axiom. The disjunctive axiom asserts that a refutation is sufficient to produce an unconfirmed proposition.

Beginning with Model 1 as the initial problem specification we introduce P' as a refutation for P. By the disjunctive axiom we have

$$\sim P \vee P' \rightarrow \sim P$$

Through the interpretation of disjunction we have

$$\sim P \rightarrow \sim P \qquad \text{or} \qquad P' \rightarrow \sim P$$

The first case is a tautology and can be ignored. The second case is a new constraint C1: $P' \rightarrow \sim P$ introduced into our model by the new fact P'.

Similarly, by introducing P'' as a refutation for P' and applying the disjunctive axiom we can generate the new constraint C2: $P'' \rightarrow \sim P'$. Adding the new constraints C1 and C2 and associated facts to the original problem gives the model shown in Figure 5. We refer to this as a generalized or disjunctive model.

R1:	$\sim Q \rightarrow \sim R$	If $\sim GoodPayHistory$ Then $\sim ApproveLoan$
R2:	$P \rightarrow \sim Q$	If $LatePayment$ Then $\sim GoodPayHistory$
C1:	$P' \rightarrow \sim P$	If $PostalDisruption$ Then $\sim LatePayment$
C2:	$P'' \rightarrow \sim P'$	If $Courier$ Then $\sim PostalDisruption$
F1:	P	$LatePayment$
F2:	P'	$PostalDisruption$
F3:	P''	$Courier$

Fig. 5: Model 4

For the disjunctive model to be evaluated correctly we require that a refutation of a proposition supersede a proof of a proposition, or that $P \wedge P' \rightarrow \sim P$. This is the *refutation* axiom and it is used to enforce a precedence on rule evaluation. Thus, constraint C2, which infers a refutation of

$PostalDisruption$, is sufficient evidence to supersede the proof of $PostalDisruption$ established by fact F2.

The disjunctive model is a generalized version of the original model. The original rules R1 and R2 and associated facts are unchanged, thus the original model is fully contained within the disjunctive model.

Note that $P' \vdash \sim P$ is valid, whereas $\sim P \vdash P'$ is not. The existence of a refutation denies the proposition, but an unconfirmed proposition does not ensure that a refutation exists.

VI. Business Rule Processing

The determination of whether a business action should occur depends on the truth value of the implication in a computational `If` statement. To evaluate a business rule implication we relate the business rule $(condition, consequent)$ pair to the logic statement $(premise, conclusion)$ pair. The business rule condition is the proposition logical premise and the logic conclusion establishes the validity of the business consequent. To discuss the relationship of logical propositions to program variables the following definitions [3] are followed.

Definition 1: A *Term* is a word or phrase which has specific meaning for the business. Examples of terms are ‘payment date’, ‘interest rate’, and ‘courier’. Terms may be of two types. *Literal* terms are terms with a specific instance or value and *Types* are terms which are a named abstraction for a set of instances or values. Terms can be equated to variables, objects or data items within a business application.

Definition 2: A *Fact* is a relationship between two or more terms. For example, a late payment fact is established when a payment date is compared to a deadline date. Facts may be classified as follows. *Base Facts* are simply asserted and *Derived Facts* are computed or inferred from mathematical calculations or other business rules.

We now make the following observations.

- A refutation questions the truth or validity of a fact. Every term has a base fact that represents the implicit relationship between the value of a term and the real world. A refutation can apply to the value of a literal term or the membership of a value in a type.
- A refutation can also apply to any fact derived during the logic derivation if the truth or validity of the computation is questioned. For example, a client may dispute the reasonableness of comparing the date of payment receipt to the mandated due date in the event of a postal strike.
- Not all facts in a business decision are necessarily approved candidates for a refutation argument. It is a business decision to determine which facts may be refuted. For example, facts derived from legislation are generally irrefutable.
- Any refutation applied to a base or derived fact may change the truth of the premise of a logic statement. The logical implication of the business rule establishes the validity of the business conclusion.
- By tracing the premise and conclusions of a logic derivation sequence we can trace the history and effect of a refutation on the business decision.
- Conventional programming languages and their implementations do not directly provide a way to easily assess the truth or validity of a fact. The truth relationship is established during program execution based upon the current values of program variables and without consideration of the value validity.

Figure 6 shows how we can implement a refutation-based logic system in an automated program from an architecturally independent perspective. A new logic interpretation layer is created by modifying the programming language grammar to introduce new expression evaluation syntax and semantics for the condition of a business rule $(condition, consequent)$ pair. Each rule condition is evaluated by a PROLOG logic engine to ascertain the propositional truth. The PROLOG system maintains external refutation logic rules and the business fact proposition database.

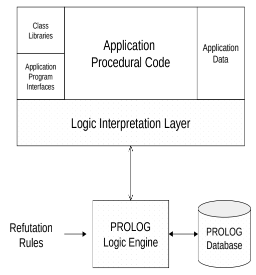
Class Libraries
Application Procedural Code
Application Data
Application Program Interfaces
Logic Interpretation Layer
Refutation Rules
PROLOG Logic Engine
PROLOG Database

Fig. 6: System Architecture

The system provides services to identify refutable business facts, refutation logic rules, and evaluate the business logic during program execution.

New refutation based reasoning operators (see Table II) for fact assertion and premise evaluation are introduced into the programming language and its implementation through Generalized Operator Overloading [5]. These new operators apply to business rule premise expressions coded in the program source. Each operator applies to a named *fact*.

TABLE II
Refutation Operators

Operator	Syntax	Result
$\mathcal{T}$	$\mathcal{T}$,*fact*	*fact* is Confirmed
$\mathcal{F}$	$\mathcal{F}$,*fact*	*fact* is Refuted
$\mathcal{P}$	$\mathcal{P}$,*fact* expr	Truth(fact)

The $\mathcal{T}$ operator asserts a new fact into the fact base as a confirmed proposition. The $\mathcal{F}$ operator asserts a new refuted fact into the fact base. The $\mathcal{P}$ operator establishes the truth of a fact as the truth value of the associated expression unless the fact has been refuted by a prior refutation argument in the rule base.

VII. A Simple Example

Figure 7 shows a small sample of C code from a simple system constructed to implement our business rules to authorize a loan. The facts established by the business rules are named through use of the operator $\mathcal{P}$.

```
approve_loan = TRUE ; good_pay_history = TRUE ;

printf("Input payment date\n") ;
scanf("%d",&payment_date) ;

// Rule R1
if (P,"late_payment" payment_date > 990501)
   good_pay_history = FALSE ;

// Rule R2
if (P,"not_good_pay_history" !good_pay_history)
   approve_loan = FALSE ;

printf("Approve loan = %s \n\n",
   (approve_loan) ? "TRUE" : "FALSE") ;
```

Fig. 7: Code Example

Generalized Operator Overloading translates the enhanced C source code in Figure 7 into standard C by converting the premise of rules R1 and R2 into procedural calls. Figure 8 shows the translated result. The procedure calls access the logic interpretation layer to evaluate the truth of the rule premise.

```
// Rule R1
if (P(payment_date > 990501,"late_payment"))
   good_pay_history = FALSE ;

// Rule R2
if (P(!good_pay_history,"not_good_pay_history"))
   approve_loan = FALSE ;
```

Fig. 8: Translated Code

The logic interpretation layer implements procedure $\mathcal{P}$. This procedure has code to invoke a PROLOG logic engine, enter the value of the rule expression as a derived fact into a fact database, and then verify the truth of the fact by establishing the named proposition as a goal to be proved. Our implementation of procedure $\mathcal{P}$ is specific to SWI Prolog [9] and uses the foreign language interface to construct a `prove_premise("late_payment")` goal and a `prove_premise("not_good_pay_history")` goal for PROLOG to evaluate.

Our PROLOG code is shown in Figure 9. PROLOG is a goal-driven search engine that searches the fact base for solutions that satisfy the proposition goal [1]. Our `prove_premise` predicate implements our refutation axiom and establishes the premise truth only if the proposition is confirmed and not refuted.

Base facts and refutation constraint rules are isolated from the program logic and maintained within the PROLOG database as shown in Figure 10. The example implements the complete logic for our generalized model described in Figure 5.

Facts are represented in our system as a PROLOG term of the form `fact(`*`name`*`,`*`state`*`)` where *name* identifies the fact and *state* is one of {*confirmed*,*refuted*}. Rules are represented in the form `rule(`*`name`*`,[`*`predicate_list`*`])` where *name* identifies the rule and *predicate_list* is a list

```
% Prove the implication for premise X.
prove_premise(X) :-
   \+ check_refuted(X), confirmed(X).

% Evaluate constraints for a rule premise.
check_refuted(X) :-
   find_refutation_constraint(X,Y,Z),
   evaluate_rule(Z).
check_refuted(X) :- refuted(X).

% Is our rule premise refuted?
find_refutation_constraint(X,Y,Z) :-
   rule(Y,Z), member(refute_fact(X),Z).

% Evaluate the rule constraint predicates.
evaluate_rule([]).
evaluate_rule([X|Y]) :-
   call(X), evaluate_rule(Y).

% Predicates to determine the state of a fact.
refuted(X)     :- fact(X,refuted).
confirmed(X)   :- fact(X,confirmed).
unconfirmed(X) :- refuted(X) ; \+ confirmed(X).

% Predicates to assert facts in the database.
confirm_fact(X) :- assert(fact(X,confirmed)).
refute_fact(X) :- assert(fact(X,refuted)).
```

Fig. 9: PROLOG Logic

```
% The known fact declarations.
fact(postal_disruption,confirmed).
fact(courier,confirmed).

% Constraint rule C1.
rule(postal_disruption,
   [prove_premise(postal_disruption),
    refute_fact(late_payment)]).

% Constraint rule C2.
rule(courier,
   [prove_premise(courier),
    refute_fact(postal_disruption)]).
```

Fig. 10: Example Fact and Rule Database

of PROLOG predicates to be evaluated when the rule is invoked. New constraint rules introduced as refutations into a disjunctive model have the form $P' \rightarrow \sim P$, represented in the predicate list as the predicates `prove_premise(`P'`)` and `refute_fact(`P`)`.

Rules are invoked if and only if they can create a refuted fact for the named proposition P. For constraint rules this generates a PROLOG sub-goal to search for a proof or confirmation of P'. A valid proof of P' requires that a refuted fact for the named proposition exist in the fact base.

This example, when executed, produces the results as shown in Table III. Column `G` gives the resulting truth value for the decision to approve the loan. Propositions P, P', and P'' refer to the `late_payment`, `postal_disruption` and `courier` facts as established in the fact database. Values `C` and `R` indicate if the proposition has been confirmed or refuted. Constraint rules `C1` and `C2`, when specified, are indicated with a $\surd$. Blank values in the table represent cases where a fact has not been entered into the database or is unconfirmed.

Model 1 without refutation rules has behaviour consistent with a traditional rule-based application. Model 4 introduces both the `postal_disruption` and the `courier` refutation

TABLE III
EXAMPLE RESULTS

	G	P	P'	P''	**C1**	**C2**
Model 1	F	C				
	T	R				
Model 4	F	C			√	√
	T	R			√	√
	T	C	C		√	√
	T	R	C		√	√
	F	C		C	√	√
	T	R		C	√	√
	F	C	C	C	√	√
	T	R	C	C	√	√

of the `postal_disruption` argument. Behaviour when the courier fact is confirmed is consistent with Model 1.

The business decision to approve or deny the loan is now determined not only by the logic coded in the application program, but also by the refutation constraints and facts established through our logic system.

VIII. Unconfirmed Facts

Unconfirmed facts are neither confirmed nor refuted. These facts do not have a `fact` term stored in our PROLOG fact base and this signifies that no identified relationship exists between a term in the program and the real world. If our business logic is to carry forward the idea of unconfirmed facts throughout the decision process then any derived fact which depends on the value of an unconfirmed business term must remain unconfirmed in the fact base. Validation rules are used to prohibit the assertion of a derived fact, normally done during the initial stages of searching for a proof of a business rule premise. The logic interpretation layer which implements the $\mathcal{P}$ operator applies the PROLOG `validate_fact(`P`)` predicate to establish if all facts relevant to the calculation of the derived expression are confirmed. Figure 11 shows the PROLOG code.

```
% Validate that the premise can be asserted.
validate_fact(X) :-
  find_validation_rule(X,Y,Z),
  evaluate_rule(Z), !,
  Z = [].

find_validation_rule(X,Y,Z) :-
  rule(Y,Z), member(invalidate(X),Z).
find_validation_rule(_,[],[]).

invalidate(_).

% The specification of what is provable.
rule(payment_date,
  [unconfirmed(payment_date),
  invalidate(late_payment)]).
```

Fig. 11: Validation of Facts

If derived fact validations are required then the $\mathcal{T}$ and $\mathcal{F}$ operators shown in Table II can be used to assert known base facts `F` into the PROLOG fact base. Validation rules reference these base facts and are of the form `rule(`*`name`*`,[`*`predicate_list`*`])` where *predicate_list* implements the inference `unconfirmed(F)` $\rightarrow$ `invalidate(`P`)`.

A derived fact will be validated only if a validation rule for the associated business rule premise has been specified. In this case the `validate_fact(`P`)` predicate will succeed only if all verification rules for proposition P reference not unconfirmed facts. If no validation rule is specified for a proposition then the truth value of the associated derived fact will, by default, be asserted during premise truth evaluation.

If our loan example is executed with the validation rule from Figure 11 included in the rule base and `payment_fact` is not confirmed, then the premise of rule R1 will never be confirmed. The resulting truth value for the `approve_loan` variable will retain its initial value. This is consistent for all cases shown in Table III.

IX. Discussion

Most business rules and policies have, over time, become complex and specialized to account for many specific circumstances and exceptional situations. These special cases arise from changing business requirements and the desire for a business to become more efficient and effective in delivering goods and services to the marketplace.

A sound understanding of business policies and rules leads to better specifications of simple, consistent business models designed to handle most business cases. Analysis of the business processes can reduce the overall complexity of the model and increase understanding of the key business objectives. If this is done, then the generalized refutation method is one way that the normal flow of business logic can be enhanced by introducing new situational knowledge into the model without the necessity to change the fundamental business rules.

A common business practice has been to revise and extend existing policies and rules so that they can correctly process new situations. This technique follows the specialized refutation method. New knowledge and facts are added into the business model in a specialized way. One difficulty with this technique is that it can lead to increasingly complex system implementations.

The solution discussed in this paper separates the business rule logical inference process from the program implementation. Changes to the business inference process can be made without the necessity of re-architecting or changing the business rules. This technique can be applied to any business system implemented through a procedurally specified language or rule-based specification that derives business facts and makes decisions.

Intuitionistic logic requires a constructive proof of a proposition before an inference decision can be made. This does not require absolute truth. Thus, although the business database may be populated with true information, in practice the validity of any fact in the database can be questioned. A refutation of a fact may not change the truth of the fact, but business needs may require that the fact be used in different ways to arrive at a decision.

X. Conclusions

To sum up, it is intuitively obvious that birds can fly. Unless, of course, the bird is a penguin. But then, even penguins can fly if they are in an aeroplane. Unless, of course, the aeroplane is grounded because of bad weather. And on it goes. So what is the absolute truth, or does it even matter?

We have shown that one way to approach this problem is to consider the deductive process in terms of intuitionistic logic. This treats the problem as an original proof and a sequence of refutations. The logic semantics are based upon direct experience or the evidence that a proposition has been confirmed or refuted. Unknown information is neither confirmed nor refuted.

Refutations can be considered as 'negative' knowledge, or rules which describe things not to do. One of the reasons why refutations or negative logic statements are not commonly employed in business systems is because there are always far more ways to dispute a proposition than to prove it. Thus, it becomes impossible to bound or restrict the problem to a known set of refutations. Clearly, the set of things that we don't know far exceeds that which we do know. Therefore, a reasonable way to attack this problem is to develop an extensible system that permits the addition of new facts and refutation arguments into the business model without the necessity to alter the original policies or knowledge state.

Two approaches to extending the business logic are discussed. The first approach, developed through application of the conjunctive axiom, modifies the business model to refine and specialize it to account for the new facts. This approach requires that each rule definition be revised to account for the new situation. This can lead to an application maintenance problem and reduce the flexibility of the system to adapt to change. The second approach, developed by applying the disjunctive axiom, generalizes the business model by the adding new rules to it with a defined ordering sequence. This approach extends the business model without requiring changes to the original policies or rules.

Both approaches provide a framework in which a generalized business system can be constructed. The specialized approach can increase the business model complexity. The generalized approach can improve the business model flexibility. Better business systems will improve client satisfaction and business results.

Acknowledgements

The author would like to acknowledge the guidance of LeRoy Johnson, Brian Cupples and Lev Goldfarb, all of whom motivated and encouraged this work.

References

[1] W.F. Clocksin and C.S. Mellish *Programming in PROLOG, Fourth Edition*, Springer-Verlag, New York ISBN 0-387-58350-5

[2] Michael Dummott, *Intuitionistic Mathematics and Logic, Part I*, Mathematical Institute, Oxford, November 1974

[3] GUIDE *Business Rules Project, Final Report, October 1997*, GUIDE International, 401 North Michigan Avenue, Chicago, IL, 60611-4267 http://www.guide.org/apbrules.htm

[4] John McCarthy, *Generality in Artificial Intelligence*, Turing Award Lecture, Communications of the ACM, vol. 30, No. 32, December 1987, pp. 1030-1035

[5] William Miles and LeRoy Johnson, *Implementing Generalized Operator Overloading*, Software Engineering and Practise, vol. x, No. y (1995), pp. xx-xx

[6] Marvin Minsky, *Negative Expertise*, International Journal of Expert Systems, vol. 7, No. 1 (1994), pp. 13-18

[7] John Nolt, *Logics*, Wadsworth Publishing Co., 1997. ISBN 0-534-50640-2

[8] Ronald G. Ross, *The Business Rule Book: Classifying, Defining and Modelling Rules*, Business Rule Solutions Inc., Houston, Texas, 1997 ISBN 0-941049-03-5

[9] SWI Prolog, Jan Wielemaker, http://swi.psy.uva.nl/projects/SWI-Prolog

PLACE PHOTO HERE

William Miles is an experienced consultant and technical architect for commercial software engineering projects. He consults on Internet technologies, client/server systems, data warehouse and distributed systems, natural language processing, information theory and formal methods. He is actively researching new ways to extend and improve the system delivery services and his research interests are with the computational processing of language, software engineering and nonmonotonic reasoning. He is a member of the Institute of Electrical and Electronic Engineers (IEEE) and the IEEE Technical Council on Software Engineering. He is also a member of the Information Metrics Institute of New Brunswick and he performs contract work for a number of private companies and government agencies.