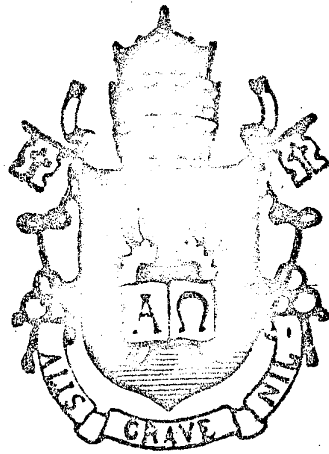


PUC

Series: Monografias em Ciência da Computação,
No. 22/91

SAFO: AN ENVIRONMENT FOR LOGICS

Roberto Lins de Carvalho
Newton José Vieira
Cláudia M.G.M. Oliveira

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Series: Monografias em Ciência da Computação, No. 22/91
Editor: Carlos J. P. Lucena November, 1991

SAFO: AN ENVIRONMENT FOR LOGICS *

Roberto Lins de Carvalho **

Newton José Vieira

Cláudia M.G.M. Oliveira

* This work has been sponsored by the Secretaria de Ciência e Tecnologia of Presidência da República Federativa do Brasil.

** PUC Rio and LNCC.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio - Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
Brasil

Tel.:(021)529-9386 Telex:31078 Fax:(021)511-5645
E-mail:rosane@inf.puc-rio.br

Abstract:

SAFO is an environment for the programming of deductive systems. There are two operational levels: (1) procedural level which consists of a logical programming language, akin of PROLOG, and (2) a general theorem prover. The interaction of this two levels, plus the existence of several bases, make possible the definition of "reasoning agents", i.e. the definition of different logics.

SAFO:An Environment for Logics

Roberto Lins de Carvalho
PUC - RJ / LNCC
Rio de Janeiro - Brazil

Newton José Vieira
Departamento de Computação
UFMG - Belo Horizonte - Brazil

Cláudia M.G.M. de Oliveira
Department of Computing
Imperial College - London - UK

October 30, 1991

Abstract

SAFO is an environment for the programming of deductive systems. There are two operational levels: (1) Procedural Level which consists of a logical programming language, akin of PROLOG, and (2) a general theorem prover. The interaction of this two levels, plus the existence of several bases, makes possible the definition of *reasoning agents*, i.e., the definition of different logics.

1 Introduction

Theorem provers have been devised for more than 20 years: many have been proposed, only a few have been built. There are several problems to be solved before a theorem prover can be useful. Such problems, tipically due to the exponetially explosive First Order Logics proof systems, can be minimized by the use of non-standard extra-logical methods and strategies.

SAFO (System for Automatic knowledge FOrmalization), is an environment containing a theorem prover, which has been developed following the idea that logical and non-logical paradigms must cooperate in order that significant knowledge processing becomes feasible.

SAFO is basically composed of two levels: the declarative logical level and the procedimental level. They function cooperatively so that SAFO handles knowledge represented in First Order Language as well as in programs, which in turn are able to act upon declarative knowledge and programs themselves.

The architecture of SAFO is schematically shown in fig.1. The following sections will present the functionallity of the modules in detail.

The communication between the two levels is realized through the use of an environment, which is not represented in the figure. This communication is possible because the languages of both levels are first order languages which differ only in their generality. The theoretical justification for this communication can be better appreciated in [1] or in [4].The language of the procedural level is similar to PROLOG[10] with minor modifications and the execution of programs in this language uses an environment technologically similar to the usual PROLOG implementations, but with variable occurrency check.

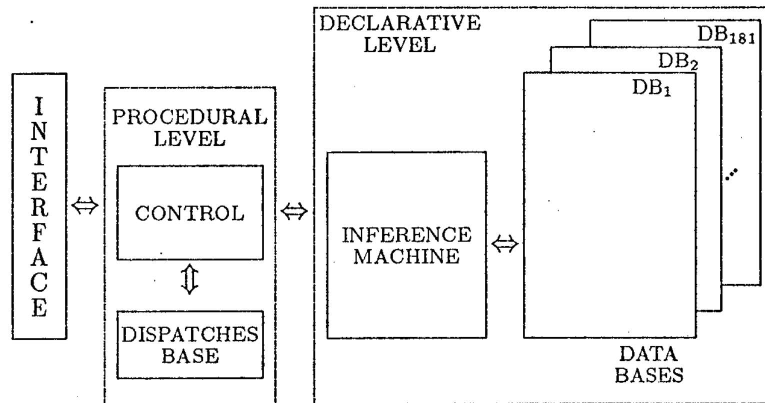


Figure 1: Basic Architecture

The language of the declarative level has the representation power of a first order language in which the existential quantifiers were substituted by skolem functions. The inference engine of the declarative level is a theorem prover, closer in its basics to Loveland's Model Elimination [13] than to Robinson's Resolution[18].

The next two sections will present the declarative level and the procedural level in detail. In section 4 we describe the interlevel communication mechanism. In order to materialize the theoretical ideas presented, section 5 gives the user's view of a working section with SAFO.

2 The Declarative Logical Level

The logical level consists of a series of databases and an inference machine. Data is stored in clausal form to comply with the theorem prover specifications, although all input/output is performed in accordance to the user's input/output forms. In the original project there is no limit imposed on the number of databases. Because of implementations restrictions, the current number of databases is 182. There is no relationship among them but their serialized

numbering (0 to 182).

The inference machine is similar to the MESON system as described in [11], where search spaces are presented in the problem reduction format. Such format suggests, not only an *efficient implementation model* based on the concepts of structure sharing and activation records [19], but also *explanation models* based on intuitive inference rules and meta-theorems.

Next, the underlying formal system, some strategic options adopted, and the explanation subsystem are sumarized.

2.1 The formal system

The language of the formal system has two kinds of entities:

clauses, as usual, are sets (disjunctions) of literals; they encode the input data.

structured clauses (SC's), which are linearizations of *proof trees.* A structured clause is a literal followed by a set of literals and/or SC's. The notation is $L\{X_1, X_2, \ldots, X_n\}$, where X_i is a literal or a SC. L is said to be the *root literal* of the SC. A *sub-SC* of a SC X is any SC that occurs in X (X is a sub-SC of itself). The root literals of all sub-SC's of X are also called *expanded literals* of X; the remaining literals are of two kinds: *reduced literals* and *candidate literals.* To distinguish a reduced literal from a candidate literal, the former is enclosed in angle brackets.

The inference rules of the formal system are:

codification: produces a SC from an input clause. The result of the codification of the clause $L_1 \vee L_2 \vee \cdots \vee L_n$ is the SC $\bot\{L'_1, L'_2, \ldots, L'_n\}$, where $\bot$ is the complement of $\top$ and each L'_i is the complement of L_i.

expansion: produces a SC from a previous SC and an input clause. Let L be (an occurrence of) a candidate literal in a SC X and K a literal in a clause C, such that there exists a m.g.u. σ of L and K. Then the result of an expansion of X through C is the substitution of L in X by the SC $L\{K_1, K_2, \ldots, K_n\}$ where K_1, K_2, $\ldots$, K_n are the complements of the literals of C except K, followed by the application of σ to the resulting SC.

reduction: produces a SC from a previous SC. Let L be a candidate literal that occurs in a sub-SC of X whose root is K, such that there exists a m.g.u. σ of the atoms of L and K and $\sigma(L)$ is the complement of $\sigma(K)$. Then the result of a reduction of X is the substitution of the candidate literal L in X by a reduced literal $\langle L \rangle$, followed by the application of σ to the resulting SC.

As can be seen above, the only way to begin a derivation is through the application of the rule of codification. That rule introduces $\bot$ as a root literal,

and the other rules preserve this literal. Starting with a SC produced by codification, all derivations are *pure linear* in the sense that one of the premises are always the last produced SC and no other SC is used as premise in a rule application. Let us see an example.

Example 2.1 *Given the database:*

1. $P(x) \vee Q(x) \vee R(x)$
2. $\neg P(x) \vee S(s)$
3. $\neg S(x) \vee \neg P(a)$
4. $\neg Q(a)$
5. $\neg R(y)$

a possible derivation would be:

6. $\bot\{R(y)\}$ (cod. 5)
7. $\bot\{R(y)\{\neg P(y), \neg Q(y)\}\}$ (exp. 1)
8. $\bot\{R(y)\{\neg P(y)\{\neg S(y)\}, \neg Q(y)\}\}$ (exp. 2)
9. $\bot\{R(y)\{\neg P(y)\{\neg S(y)\{P(a)\}\}, \neg Q(y)\}\}$ (exp. 3)
10. $\bot\{R(a)\{\neg P(a)\{\neg S(a)\{\langle P(a)\rangle\}\}, \neg Q(a)\}\}$ (red.)
11. $\bot\{R(a)\{\neg P(a)\{\neg S(a)\{\langle P(a)\rangle\}\}, \neg Q(a)\{\}\}\}$ (exp. 4)

Consider the SC 11:

$$\bot\{R(a)\{\neg P(a)\{\neg S(a)\{\langle P(a)\rangle\}\}, \neg Q(a)\}\}$$

It can be seen as "codifying" a set of *general contrapositives* [11] of some input clauses instances:

- $R(a) \Rightarrow \bot$ (instance of 5)
- $\neg P(a) \wedge \neg Q(a) \Rightarrow R(a)$ (instance of 1)
- $\neg S(a) \Rightarrow \neg P(a)$ (instance of 2)
- $P(a) \Rightarrow \neg S(a)$ (instance of 3)
- $\top \Rightarrow \neg Q(a)$ (instance of 4)

The objective of the inference machine will be to produce a derivation of a SC without candidate literals. The clause instances codified by such SC constitute a set of clauses whose (all) ground instances are insatisfiable in the spirit of the Herbrand Theorem. This implies that set of input clauses is insatisfiable (the other way is also true: if the set of input clauses is insatisfiable there exists a derivation of a SC without candidate literals).

The literal $\bot$ is a literal whose truth value is *false*, and it is present in all derivable SC's. In a certain sense, a SC without candidate literals corresponds

to a derivation of the empty clause in resolution based proof procedures ($\perp$, as the empty clause, indicates a contradiction in a SC free of candidate literals).

As was said above, there is a close relationship betwen this formal system and that of Model Elimination[11]. The rules of expantion and reduction are very similar to the rules of extension and reduction of Model Elimination. The main diference is that Model Elimination works with *ordered clauses*, representing paths of the proof tree, instead of *structured clauses*, and all sub-paths ending in a expanded (or A-literal in Model Elimination terminology) or reduced literal is discarded as soon as it is produced.

The representation of the whole tree has the following advantages:

- it gives one more possibility where to apply heuristics (choice of the next candidate literal to work with);
- explanation models can be developed from it.

Any way, the proof tree constitutes the base for an important class of machine inference implementations.

2.2 Strategy

The first choice to be fixed is what clause(s) to codify (using the codification rule). In SAFO only the clauses of the denial of the query can be codified. This makes the proof procedure incomplete but more suitable for the intended class of applications, as we don't need to trivialize the underlying theory by the introduction of an inconsistency in the database.

Several things can be made to prune the search space based on the structure and contents of the SC's, similar to that proposed for Model Elimination and the MESON system, without sacrificing completeness (pruning is effected either to remove a fuitless or a redundant path). Only one was implemented in the actual version: no sub-SC can have a root literal identical to a candidate literal that occurs in it.

There are two places where heuristics can be used to guide the search space traversal:

- choice of the next candidate literal to expand/reduce;
- choice of an input clause to be used in an expansion.

The first choice involves the choice of the inference rule to apply. Reduction appears to be better than expansion as it decreases the number of candidate literals by 1. But in SAFO it is given priority to expantion, because explanations are more clear to the layman when the SC don't have reduced literals. It is selected the candidate literal with minimal number of variables, as these usually unify with fewer clauses.

In case of expansion, it is chosen one of the clauses with minimal lenght available, as usual in other systems. Amongst those clauses with the same number of literals, it is given preference for the most recently introduced in the database.

More sofisticated prunning rules and heuristics could be used, and will certainly be used in other versions to come.

2.3 Explanation

In the actual version of SAFO, not all kinds of explanation developed was implemented yet. Thus, what follows only gives an idea of the explanation models *intended* to be implemented in further versions of SAFO.

Let us supose initially that only one clause of the denial of the query was used to construct a SC free of candidate literals.

All SC's in this section are suposed to be *free of candidate literals.*

It can be shown that to each sub-SC corresponds a *lema.* In a sub-SC free of candidate and reduced literals the lemas are just the root literal of the sub-SC (it is easy to see why, by considering the set of clauses codified by the sub-SC in implicative form). In general a lema, $\mathcal{L}(L\alpha)$, associated to the sub-SC $L\alpha$ is a clause whose literals are:

- L; and
- the complements of all reduced literals that occurs in α.

(it is easy to see why, by considering the lema in the form $K_1 \wedge K_2 \wedge \cdots \wedge K_n \Rightarrow L$, where the K_i's are the reduced literals of α).

A possible explanation schema for $\mathcal{L}(\mathcal{X})$, where X is a sub-SC of the form

$$L_0\{L_1\{\cdots\}, L_2\{\cdots\}, \ldots, L_n\{\cdots\}, \langle M_1\rangle, \langle M_2\rangle, \ldots, \langle M_k\rangle\}$$

is:

To conclude
$$\mathcal{L}(X)$$
it was used the clause:
$$L_1 \wedge \cdots \wedge L_n \Rightarrow L_0 \vee M_1' \vee \cdots \vee M_k'.$$
It was determined:
$$\mathcal{L}(L_1\{\cdots\})$$
$$\mathcal{L}(L_2\{\cdots\})$$
$$\vdots$$
$$\mathcal{L}(L_n\{\cdots\}).$$

where M_i' is the complement of M_i. We present here only the proposicional case. It is easy to extend this schema to the predicate logic case.

Example 2.2 *Let X be the sub-SC*

$$A\{B\{\langle\neg A\rangle, D\{\langle\neg B\rangle\}\}, C\{\}\}$$

Two steps of the explanation correponding, respectively, to the sub-SC's X and $B\{\langle\neg A\rangle, D\{\langle\neg B\rangle\}\}$ would be:

> To conclude
> A
> it was used the clause:
> $B \wedge C \Rightarrow A.$
> It was determined:
> $B \vee A$
> $C.$

> To conclude
> $B \vee A$
> it was used the clause:
> $D \Rightarrow B \vee A.$
> It was determined:
> $D \vee B.$

It is interesting to note that a step of explanation has the following pattern:

> To conclude
> $M_1 \vee \cdots \vee M_m \vee \Delta_1 \vee \cdots \vee \Delta_n$
> it was used the clause:
> $L_1 \wedge \cdots \wedge L_n \Rightarrow M_1 \vee \cdots \vee M_m.$
> It was determined:
> $L_1 \vee \Delta_1$
> $L_2 \vee \Delta_2$
> $\vdots$
> $L_n \vee \Delta_n.$

where each Δ_i is a disjunction of zero or more literals. Obviously, this schema is logicaly sound.

An answer with several disjuncts appears when the SC has several instances of clauses of de denial of the query (instances of the same clause can appear anywhere in the SC). In this case, the answer can be explained:

- by explaining one of the disjuncts assuming the denial of the others as hypotesis; or
- by expliciting under what (exclusive) conditions each of the disjuncts are verified (similar to the class C questions of [5]).

Several other explanation models can be developed based on the "reading" propiciated by the SC. These models are independent of the especific deduction used to produce the SC (the final SC can be manipulated so that a specific kind of explanation could be produced).

2.4 Semantical Nets

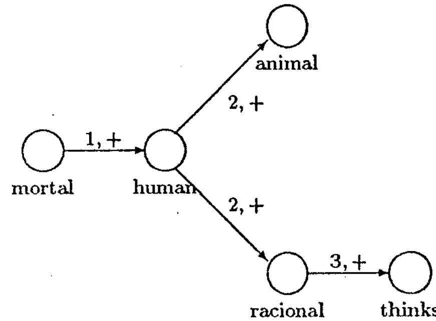
The use of semantical nets is very important. Let us consider the following base as given by the user:

$$\begin{array}{l} human(*x) \rightarrow mortal(*x) \\ animal(*x) \wedge rational(*x) \rightarrow human(*x) \\ thinks(*x) \rightarrow rational(*x) \end{array}$$

The clausal forms are:

1. $\neg human(*x) \vee mortal(*x)$
2. $\neg animal(*x) \vee \neg rational(*x) \vee human(*x)$
3. $\neg thinks(*x) \vee rational(*x)$

The semantical net for this base is:



This graph is used by the inference machine to give a natural direction of search for a literal in expansion. Notice that the data base does not contains any ground unit clause, and so cannot answer queries about a particular individual. So, the query: $mortal(John)$? could not be answered from these clauses. SAFO provides a command *knowacqs* that when used triggers an automatic **knowledge acquisition** mechanism, which sends to the user a request to additional ground literal:

```
?animal(John) (Y/N/U)

?>rational(John) (Y/N/U)
```

to which he may answer Yes, No or Unknown.

3 The Procedural Level

SAFO's procedural level is a logic programming environment. It consists of one base of programs, and a controlling module to execute the actions coded into the programs or input by the user via the interface. The programs in SAFO are called dispatches because of their communication of the two levels, the procedural level sends (receives) *informations* to (from) the declarative level.

The primitives of SAFO's procedural language can be classified as:

- interlevel communication commands, which perform the linkage between the two levels
- interface primitives
- string manipulation primitives
- basic arithmetics primitives

The syntax of SAFO's procedural language is partially BNF-defined below. The defined primitives are interlevel communication programs, with which we are primarily concerned here. In appendix A, the remaining commands and their syntax are presented.

```
<programm>          ::=  <elem-programm>
                       | <comp-programm>
<elem-programm> ::=  prove "(" <numbase> "," <sent1> ")"
                       | assert "(" <numbase> ","  <sent2> ")"
                       | forget "(" <numbase> ","  <sent1> ")"
                       | level "(" 0 | 1 | ... | maxint ")"
                       | skolem "(" (<sent1>|<sent2>) "," <variable> ")"
                       | <interface-commands>
<comp-programm>  ::=  ( <programm> "&" <programm> )
                       | ( <programm> "|" <programm> )
```

The Language which defines the syntactical variables <sent1> and <sent2> are dialects of first order languages. So, from a theoretical point of view, we have the following sets of symbols:

$\mathcal{V}$ - Variables $\{x_0, x_1, x_2, \cdots, x_n, \cdots\}$
$\mathcal{C}$ - Constants $\{a_1, a_2, \cdots, a_n, \cdots\}$
$\mathcal{F}$ - Functions $\{f_1, f_2, \cdots, f_n, \cdots\}$
$\mathcal{P}$ - Predicates $\{P_1, P_2, \cdots, P_n, \cdots\}$

Actually, in the declarative language of SAFO, the symbols of the alphabet are strings [1].

From this alphabet *terms* and *formulas* are constructed by induction, as in first order languages:

[1] Strings in SAFO are defined by the following grammar:

T_1 - Variables and constants are terms.
T_2 - If $t_1, t_2, \cdots, t_n$ are terms and f a function then $f(t_1, t_2, \cdots, t_n)$ is a term.
T_3 - The only terms are those expressions which follows from T_1 and T_2 above.

So, father-of(John), intersection(A,B) and f(A) are terms. Atomic formulas (positive facts) are defined by:

A_1 - If $t_1, t_2, \cdots, t_n$ are terms and P a predicate then $P(t_1, t_2, \cdots, t_n)$ is an atomic formula.
A_2 - The only atomic formulas are those expressions which follow from A_1 above.

Formulas are defined by:

F_1 - Atomic formulas are formulas.
F_2 - If α and β are formulas then

. $\neg\alpha$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \rightarrow \beta)$ and $(\alpha \leftrightarrow \beta)$ are formulas.

F_3 If x is a variable and α is a formula then $\forall x\, \alpha$ and $\exists x\, \alpha$ are formulas.
F_4 - The only formulas are those expressions which follow from F_1, F_2 and F_3 above.

In the present version of SAFO, for computational reasons, the conectives are:

and	*corresponds to*	$\wedge$
or	*corresponds to*	$\vee$
implies	*corresponds to*	$\rightarrow$
not	*corresponds to*	$\neg$

Formulas asserted into databases (<sent1>) are of the form:

$$\forall x_1 \cdots \forall x_n \Psi(x_1 \cdots \forall x_n)$$

i.e. universal formulas. So, we are assuming that the axioms have been previously skolemized. For instance, $\forall x \exists y(is - father(y, x))$ is represented by $is - father(father - of(*x), *x)$, where $father - of$ is the introduced skolem function. On the other hand, formulas used as queries (<sent1>) are existentially quantified. These assumptions make redundant the presence of quantifiers in the language.

$< letter >$	$\longrightarrow$	$a \mid b \mid \cdots \mid z$
$< letter >$	$\longrightarrow$	$A \mid B \mid \cdots \mid Z$
$< numeral >$	$\longrightarrow$	$0 \mid 1 \mid \cdots \mid 9$
$< let - name >$	$\longrightarrow$	$< letter > \mid < letter > < let - name >$
$< num - name >$	$\longrightarrow$	$< num > \mid < num > < num - name >$
$< name >$	$\longrightarrow$	$< let - name > \mid < let - name > < num - name >$
$< constant >$	$\longrightarrow$	$< let - name >$
$< variable >$	$\longrightarrow$	$* < let - name >$
$< skolem >$	$\longrightarrow$	$SKL < num - name >$
$< predicate >$	$\longrightarrow$	$< let - name > < num - name >$
$< function >$	$\longrightarrow$	$< let - name > < num - name >$

In the current version of SAFO there are some restrictions on the acceptable forms of the matrix Ψ, but these restrictions do not limit the representational power of first order languages.

4 The communication between levels

The introduction of programms in the dispatches base is made by a special command **definedsp** in the following way:

$$definedsp(< head >, < body >)$$

for example:

$$\begin{array}{l} definedsp(man(John), succeed) \\ definedsp(mortal(*x), man(*x)) \end{array} \qquad (1)$$

which is not essentially different from PROLOG.

The introduction of declarative knowledge into a database is made by another command **assert**, in the following way:

$$assert(< numbase >, < sent >)$$

For example:

$$\begin{array}{l} assert(1, man(John)) \\ assert(1, man(*x) \; implies \; mortal(*x)) \end{array} \qquad (2)$$

Now we have communication between the two levels. Note that the knowledge asserted to database 1 in (2) is the declarative form of the program in (2). Therefore, the command:

$$mortal(*x) \;\; \& \;\; msg(*x) \qquad (3)$$

corresponds to:

$$prove(1, mortal(*x)) \;\; \& \;\; msg(*x) \qquad (4)$$

In (2), the communication of instantiations of variables is from the declarative to the procedural level, and is used to display a message. In the next example, which encodes the semantics of the connective AND, the communication is in the opposite way:

$definedsp(conj(*x, AND(*y, *z)),$	$prove(*x, *y)$	*(commands the inference machine to prove $*y$ in base $*x$)*
	$\& \; prove(*x, *z)$	*(commands the inference machine to prove $*z$ in base $*x$)*
	$\& \; msg(yes))$	*displays a positive answer.)*

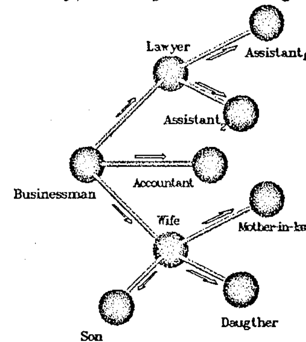
In the execution of command $conj(1, AND(P(*x,*y), Q(*x, f(*y))))$, the values of $*x$, $*y$ captured in the proof of $P(*x,*y)$ are used in the proof of $Q(*x, f(*y))$. The next program shows the use of this communication for the creation of a nonredundant and non-contradictory base:

```
definedsp(rnc(*x,*y,*z),  level(*z)                    ( defines the depth of deductions )
                          & skolem(*y,*v)              ( skolemizes *y )
                          &prove(*x,*v)                ( commands the proof of *v )
                          &msg("redundant")            ( the knowledge of *y
                                                       was known in base *x )
                          &level(30)                   ( the level is reset )
                          |level(*z)
                          &prove(*x,not *y)
                          &msg("contradictory")
                          &level(30)
                          |assert(*x,*y)
```

5 A Sample Application of SAFO

In order to show the realization of SAFO's features described so far we will present its use motivated by a practical application. This example was inspired by a Modal Logic lecture given by Prof. Dov Gabbay, at Imperial College.

A certain businessman wants to buy a new house. His decisions are based upon the advice of his lawyer, his accountant and his wife. The lawyer has got two assistants who help him with technical matters. The wife has to heed the needs of her children and her mother who also lives with them. The connections among the people involved is given in the figure.



The businessman wants a system that produces an answer to the question of buying or not the house. To solve this problem we will use the two levels of SAFO, each playing a role according to its adequacies.

First of all, the data pictured by the directed graph must be given to the system. The most abvious solution is to define the accessibility relation among the worlds (systems of beliefs) of each character. The predicate $listen-to(*p1, *p2)$ represents this relation with the following facts, each one corresponding to an edge of the graph:

```
listen-to(businessman, Wife)          listen-to(businessman, Lawyer)
listen-to(businessman, Accoutant)     listen-to(wife, Son)
listen-to(wife, Daughter)             listen-to(wife, Mother-in-Law)
listen-to(lawyer, Assistant-1)        listen-to(lawyer, Assistant-2)
```

Each person has got a system of criteria expressed as a set of axioms. These axioms are stored in databases classified by their holders. The linkage between a database and its owner is represented by predicate $base(*p, *b)$. Thue, we have the facts:

$base(businessman, 1)$	$base(wife, 2)$	$base(lawyer, 3)$
$base(accoutant, 4)$	$base(son, 5)$	$base(daughter, 6)$
$base(mother{-}in{-}Law, 7)$	$base(assistant{-}1, 8)$	$base(assistan{-}2, 9)$

Database 0 will hold the meta-knowledge of the system i.e. knowledge about the actors and their parts. Therefore, the initialization of our system would be performed by the program:

$$
(1) \quad \begin{array}{lll} initialization & \leftarrow & assert(0, listen{-}to(businessman, wife)) \\ & & \&\ assert(0, listen{-}to(businessman, lawyer)) \\ & & \vdots \\ & & \&\ assert(0, base(assistant{-}2, 9)) \end{array}
$$

In order to judge if the businessman may buy the house, their counselors must take some actions, typically applying some reasoning process to verify their knowledge. If they have counselors themselves they might want to reason over their opinions too. The most natural way of representing each one's criteria is by a sequence of actions: a program.

For example, suppose the wife listens to her children and awards 15% of her approval to each of their opinions. She may also listen to her mother and give her 20% credit. The remaining 50% is the value of the wife's own considerations. She then approves the house if she obtains at least 80% overall approval. The program representing the wife's reasoning would be stored in her database by predicate $belief{-}prog(*head, *body)$ as follows:

$$
(2) \quad \begin{array}{lll} assert(2, & belief{-}prog(\text{"}belief(wife)\text{"}, & \text{"} + (0, 0, *ap{-}value) \\ & & \&\ (prove(2, approved{-}by(son)) \\ & & \&\ inc(*ap{-}value, 15) \mid succeed) \\ & & \&\ (prove(2, approved{-}by(daughter)) \\ & & \&\ inc(*ap{-}value, 15) \mid succeed) \\ & & \&\ (prove(2, approved{-}by(mother{-}in{-}law)) \\ & & \&\ inc(*ap{-}value, 20) \mid succeed) \\ & & \&\ (prove(2, likes(wife, house)) \\ & & \&\ inc(*ap{-}value, 50) \mid succeed) \\ & & \&\ \geq (*ap{-}value, 80)\text{"})) \end{array}
$$

She would also have some rules to describe her own considerations. For instance, a very simplified set would be installed in her database by:

$$
(3) \quad assert(2, is{-}big(*x)\ and\ color(*x, pink)\ \ implies\ \ likes(wife, *x))
$$

All the other counselors, including the businessman himself have an analogous database, partly holding procedural knowledge, partly declarative knowledge.

Note that command (2) does not define a program as such: it is a declarative knowledge assertion in which the predicate asserted has a program description as argument. Before the system is able to accept a query, the programs representing the criteria of the relevant characters, and only theirs, must be installed in the dispatches base. This is set by a program that uses the accessability relation recursively in order to obtain the relevant belief programs in the system and install them, as follows:

$$
\begin{array}{llll}
 & relevant\text{--}beliefs(*root), & \leftarrow & prove(0, base(*root, *x)) \\
 & & & \&prove(*x, belief\text{--}prog(*head, *body)) \\
 & & & \&strestru(*head, *headstr)^2 \\
 & & & \&strestru(*body, *bodystr) \\
(4) & & & \&definedsp(*headstr, *bodystr) \\
 & & & \&cut \\
 & & & \&prove(0, ac(*root, *connection)) \\
 & & & \&relevant\text{--}beliefs(*connection) \\
 & & & \&fail)
\end{array}
$$

So at this stage the system is composed of:

- meta-knowledge base 0
- 9 databases with at least one atomic fact $belief\text{--}prog(*head, *body)$ and possibly other facts and rules
- 9 installed dispatches for program *belief*
- possibly other databases which are irrelevant to our system

Now we can define the program which answers the businessman's queries. It is important to notice that it is independent of the people that advises him (possible worlds), their actual connections (accessibilities) or their beliefs (logical systems) which are the data stored in the system.

$$
\begin{array}{llll}
 & aprove(*resultbase, *person) & \leftarrow & prove(0, listen\text{--}to(*person, *subperson)) \\
 & & & \&\ prove(0, base(*person, *baseperson)) \\
 & & & \&\ aprove(*baseperson, *subperson) \\
(5) & & & \&\ fail \\
 & & & |\ belief(*person) \\
 & & & \&\ assert(*resultbase, aproved\text{--}by(*person))
\end{array}
$$

Structurally, the program consists of a disjunction:

- the first sequence calls the program recursively using the accessibility relation in order to collect the results from the connected worlds;
- the second sequence triggers the *belief* program of the current world and if it succeeds predicate $approved\text{--}by(*person)$ is asserted to database indicated by $*resultbase$

Let us now examine the system after a query has been formulated. For the sake of simplification let us suppose that all the characters but the businessman and his wife blindly approve the house. Their belief programs would be instantiations of:

$$(6)\ belief(*x) \leftarrow succeed$$

The businessman reasoning would be expressed as:

$$(7)\quad belief(businessman) \leftarrow prove(1, approved\text{-}by(wife)) \ \&\ prove(1, approved\text{-}by(accountant)) \ \&\ prove(1, approved\text{-}by(lawyer))$$

After the query:

$$aprove(181, businessman)$$

the state of the relevant databases would be:

Base 1
belief−prog("belief(businessman)", ···)
approved−by(lawyer)
approved−by(accountant)

Base 2
belief−prog("belief(wife)", ···)
*is−big(*x) and color(*x, pink) implies likes(wife, *x)*
approved−by(son)
approved−by(daughter)
approved−by(mother−in−law)

Base 3
belief−prog("belief(lawyer)", ···)
approved−by(assistant−1)
approved−by(assistant−2)

Base 4
belief−prog("belief(accountant)", ···)

Base 5
belief−prog("belief(mother−in−law)", ···)

Base 6
belief−prog("belief(daughter)", ···)

Base 7
belief−prog("belief(son)", ···)

Base 8
belief−prog("belief(assistant−1)", ···)

Base 9
belief−prog("belief(assistant−2)", ···)

Since we are working under closed world assuption, the wife's belief would not be fullfilled: she does not know either if the house is big or its color. Hence,

the businessman would not buy the house. However, is we turn the knowledge acquisition strategy on, SAFO will try to obtain from the outside world information that it is lacking in order to continue a deduction. Therefore, the query:

$$knowaqs(1) \ \& \ aprove(181, businessman)$$

would cause two question to be asked to the businessman:

```
?>is-big(house) (Y/N/U)

?>color(house,pink) (Y/N/U)
```

to which he may answer Yes, No or Unknown. Of course, if the answer to both questions is Yes then the facts $is{-}big(house)$ and $color(house, pink)$ will be stored to database 2. Now the wife and consequently the businessman approve the house.

References

[1] Carvalho, R.L. *Some Results in Automatic Theorem-Proving with Applications in Elementar Set Theory and Topology* Ph.D. Thesis Dept. of Computer Sciences University of Toronto, 1974.

[2] Carvalho,R.L e Vieira, N. J. - SAFO : Um Ambiente para Desenvolvimento de Protótipos de Sistemas Especialistas Baseado em Lógica, II SBIA, São José dos Campos-SP, 1985.

[3] Carvalho,R.L. - Engenharia do Conhecimento, 2a. EBAI, Tandil-ARG, fev 1987.

[4] Carvalho,R.L. - Linear Resolution, Definitions and Functions, Progress Report, Imperial College, London, 1991.

[5] Chang,C.L. e LEE, R. C. T. - Symbolic Logic and Mechanical Theorem Proving, Academic Press, USA, 1973.

[6] Enderton, H.B. - A Mathematical Introduction to Logic, Academic Press, 1972

[7] Green, C.C. *The Application of Theorem Proving in Question-Answering Systems* Ph.D. Thesis, Stanford University, Stanford California, USA 1969.

[8] Hayes, P. J. - In Defense of Logic, Proceedings on IJCAI-5, 559-565,1977.

[9] Kowalski, R. Kuehner, D. Linear Resolution with Selection Functions *Artificial Intelligence 2* 1971 227–260.

[10] Kowalski, R.A. - PROLOG as a Logic Programming Language, Research Report DOC 81/26, Dept. of Computing, Imperial College of Science and Thecnology, 1981

[11] Loveland, D.W. - Automated Theorem Proving : A Logical Basis, North-Holland, 1978

[12] Loveland, D.W. Linear Format for Resolution. *Proc. of the IRIA Symposium of Automatic Demonstration.* Versailles France, 1968, 147–162.

[13] Loveland, D.W. Simplified Format for the Model-Elimination Theorem-Proving Procedure *Journal of The ACM, 16*, 1969, 349–363.

[14] Luckham, D. Refinement Theorem in Resolution Theory. *Proc. of the IRIA Symposium on Automatic Demonstration* Versailles, France, 1968 163–190.

[15] Menezes, C. S. e Tavares, O.L - SAFO - Um ambiente para processar conhecimento, V SBIA, Natal-RN, 1988

[16] Moore, R.C. - The Role of Logic in Knowledge Representation and Commonsense Reasoning, proceedings of NCAI, Pittsburg,PA,1982.

[17] Projeto SAFO, Especificação do Sistema, Rio, PUC/RJ - RDC, 1987, (documento reservado)

[18] Robinson, J.A. A Machine-Oriented Logic Based on the Resolution Principle *Journal of the ACM. 12*(1), 1965 23–41. Resolution Principle, JACM, 12(1) : 23-41, jan 1965.

[19] Vieira, N.J. *Máquinas de Inferência para Sistemas Baseados em Conhecimento*, Tese de Doutorado, PUC/RJ, Rio de Janeiro, 1987.

[20] Vieira, N. J. - Máquinas de Inferência para Processamento de Conhecimento, PUC/RJ - DI, Tese de Doutorado, out 1987.

[21] Vieira, N. J. - SAFO - Manual do Usuário, Rio, PUC/RJ - Depto de Informática, 1985 (circulação interna)