Extending Reflective Architectures

Timothy M. Lownie [1]
Dept. of Computing and Information Science,
Queen's University,
Kingston, Ont., K7L 3N6,
Canada

Abstract

The overhead incurred by reasoning in knowledge-based systems can be considerable when it is forced to rely on search. Even problems that are known to have tractable solutions can expend large amounts of computation when the inference method is too general. As discussed in this paper, reflective architectures provide a well-motivated framework for integrating specialized control with general reasoning in knowledge-based systems. However, progress in developing reflective architectures for more expressive languages such as first-order logic has encountered several problems of its own. Briefly, this paper considers a reflective architecture for general declarative languages, and describes how declarative and procedural requirements can be combined in a reflective system for first-order logic. As part of this example, two kinds of control, in the form control strategies and subsidiary deduction rules, are identified.

I. Introduction

Primarily, the interest in reflective architectures has focused on addressing the problem of search-intensive inference symptomatic of general knowledge-based systems. The basic approach has been to have the control component included alongside general inference: (i) generate secondary inferences to select among the best courses available to general inference, or (ii) apply subsidiary deduction rules to replace inferences that would otherwise have to be performed directly. The design is considered 'reflective' (or 'introspective') since it provides for a limited form of self-directed reasoning. Examples of systems proposed along these lines may be found in [Bowen and Kowalski 1982], [Davis 1980], [Gallaire and Lasserre 1982], [Genesereth and Smith 1983], [Kramer 1984], and [Weyhrauch 1980].

A clearer picture of the concept of reflection in knowledge-based systems can be obtained by considering how control information is to be expressed. The idea, as described in [Smith 1986], is to express 'properties of control' in the same way as 'properties of a domain', but to employ a separate knowledge-base (KB) for this purpose. One immediate advantage is that control information is not tied directly to any set of domain properties, and is therefore free to interact in solving different problems. A second advantage is that by representing control in terms of a knowledge-base, large-scale changes, such as those involved in re-programming an implementation, do not have to be made each time new control properties are introduced.

So far, combining reflection with pure declarative languages such as full first-order logic has run into several problems. The first is that, to be effective, secondary (or reflective) inference must be more constrained, less dependent on search, than the general inference processes over which it is to exercise control. However, it is not immediately apparent how to satisfy the demands of having a flexible control component, constraining reflective inference, and keeping the system deductively complete. The second problem centers on a pragmatic issue involving the relative ease with which control constructs can be expressed for a given representational language. In particular, an effective notion of control seems to depend heavily on the syntatic form of the sentences over which they operate. For example, consider the following definition of Ancestor:

$$\forall x, z\ \mathrm{Ancestor}(x, z) \equiv \mathrm{Parent}(x, z) \vee \exists y\ \mathrm{Ancestor}(x, y) \wedge \mathrm{Parent}(y, z).$$

To check if x is an ancestor z, a good strategy is to first-check if x is a parent of z, and if not, to find a y such that y is a parent of z. (Informally, the corresponds to traversing up the family tree.) However, representing the same sentence as a set of clauses,

$$\neg\mathrm{Ancestor}(x, z) \vee \mathrm{Parent}(x, z) \vee \mathrm{Ancestor}(f(x, z), z)$$
$$\mathrm{Ancestor}(x, z) \vee \neg\mathrm{Parent}(x, z)$$
$$\mathrm{Parent}(f(x, z), z) \quad \neg\mathrm{Ancestor}(x, y) \quad \neg\mathrm{Parent}(y, z),$$

it is not longer straightforward how the same control strategy can be used. Additionally, there is the problem of how to logically organize sets of clauses so that it is possible to recognize at what point particular strategies may be applied.

The object of this paper is to describe a reflective architecture for declarative languages, and show how it

financial assistance for this work was gratefully received from the School of Graduate Studies, and the Department of Computer Science, University of Toronto.

leads to a notion of selective inference for KB-systems based on first-order logic. The purpose of presenting a general architecture is to provide an account of the knowledge available to reflective (or non-reflective) inference, i.e. to make precise the representational properties of these architectures which is independent of particular languages, or ideas on process and control. The design of a reflective system for first-order logic, following in Section III, is considered an instance of this general architecture. The main part of the paper focuses on how a procedural correspondence between reflective and non-reflective inference can be obtained, laying the groundwork for two kinds of control, introduced in Section IV. Points in the design incorporate aspects of Smith's model of *procedural reflection* ([1982]), in conjunction with Murray's rule of NC- (Non-Clausal) Resolution ([1982]).

II. The Model

A. Theories, Meta-theories

A *logic* $\mathbf{L}$ can be described as a 2-tuple $< L, Cn >$, where L is a *language*, and Cn a *consequence operator*. That is, Cn is a mapping of $\mathcal{P}(L)$ (the power set of L) into itself, which satisfies, for all $X, Y \subseteq L$,

(i) $X \subseteq Cn(X) = Cn(Cn(X))$,

(ii) $Cn(X) \subseteq Cn(Y)$, whenever $X \subseteq Y$.

The model is based on the relationship between a *theory* and a *meta-theory*. Informally, a theory denotes a collection of statements about some domain, while a meta-theory denotes a collection of statements about a theory. Formally, by a 'theory' it is meant any set $A \subseteq L$ closed under Cn; that is, $A = Cn(A)$, or equivalently, $A = Cn(B)$, for some $B \subseteq L$. Note that $Cn(B)$ denotes the set of all logical consequences that follow from the statements (premises) in B.

The ability of a meta-theory to refer explicitly to parts of a theory is carried out by way of a naming relation between linguistic entities in the *object-language*, used to express the theory, and terms in the *meta-language*, used to express the meta-theory. For example, suppose *P(x)* is a formula of L. Designating L as the object-language, this formula might be represented in a meta-language V as

$$pcons(predsym(\ulcorner P \urcorner), varsym(\ulcorner x \urcorner)),$$

where *pcons*, *predsym*, and *varsym* are functions in L', and $\ulcorner P \urcorner$ and $\ulcorner x \urcorner$ are constants in L' used to designate P and x in L. Without defining such as relation here, in general $\ulcorner E \urcorner$ will be used to denote the meta-language representation of the object-language expression, or set of expressions, given by E.

Multiple theories are often useful for maintaining sets of statements about more than one domain, or for having different aspects of the same domain organized into separate contexts. In the model, multiple theories are expressed as distinct sets of premises to a logic $\mathbf{L}_0 = < L_0, Cn_0 >$. Thus, an arbitrary set of theories has the form $Cn_0(\mathrm{KB}_0), ..., Cn_0(\mathrm{KB}_n)$, where $\mathrm{KB}_0, ..., \mathrm{KB}_n$ denote sets of premises in L_0. Therefore, all theories are expressed in the same language and closed under the same consequence operation. Note that $Cn_0(\emptyset)$ (where '$\emptyset$' refers to the empty set) denotes exactly the set of sentences valid across all theories expressible by $\mathbf{L}_0$.

The advantage of this approach is that it allows a single logic $\mathbf{L}_1 =< L_1, Cn_1 >$ to express the meta-theories for any of these theories. A set of meta-theories has the form $Cn_1(\mathrm{mKB}_0), ..., Cn_1(\mathrm{mKB}_p)$, where $\mathrm{mKB}_0, ..., \mathrm{mKB}_p$ denote sets of premises in the meta-language L_1. In this way, a single meta-theory can be associated with an arbitrary set of theories, or alternatively, an arbitrary set of meta-theories can be associated with any theory.

Also incorporated into the model is a *reflection principle* ([Feferman 1962]), which establishes a correspondence between a theory and any of its meta-theories. The property defined is one of *provability*: in general, for $\mathbf{L} =< L, Cn >$, let $A \vdash_{\mathbf{L}} \alpha$ hold iff $\alpha \in Cn(A)$, where $A \subseteq L$ and $\alpha \in L$. Then,

$$\mathrm{KB}_k \vdash_{\mathbf{L}_0} \alpha \text{ iff } \emptyset \vdash_{\mathbf{L}_1} \beta_1(\ulcorner \mathrm{KB}_k \urcorner, \ulcorner \alpha \urcorner),$$

where $\beta_1(\cdot, \cdot)$ is a fixed 2-place expression of L_1 (with 'holes' for L_1 terms), $\beta_1(\ulcorner \mathrm{KB}_k \urcorner, \ulcorner \alpha \urcorner)$ is a formula of L_1, and $\ulcorner \mathrm{KB}_k \urcorner$ and $\ulcorner \alpha \urcorner$ are meta-language expressions denoting KB_k and α in L_0, respectively. Thus, '$\beta_1(\ulcorner \mathrm{KB}_k \urcorner, \ulcorner \alpha \urcorner)$' might be read as '$\alpha$ is a logical consequence of KB_k in $< L_0, Cn_0 >$'.

It follows from the definition that provability is the same across all meta-theories by virtue of the fact that $Cn_1(\emptyset) \subseteq Cn_1(\mathrm{mKB}_k)$, for all $\mathrm{mKB}_k \subseteq L_1$. In particular, any meta-theory $Cn_1(\mathrm{mKB}_k)$ can be viewed as arising from a set of '*axioms of provability*' $\mathrm{AP}_1 \subseteq L_1$ such that $Cn_1(\mathrm{mKB}_k) = Cn_1'(\mathrm{mKB}_k \cup \mathrm{AP}_1)$, for some consequence operator Cn_1'.

B. The Tower

By definition, any meta-theory is itself a theory, so that hypothetically this theory can be provided with its own meta-theory - and so on, *ad infinitum*. In its full generality, therefore, the model is presented in terms of a 'tower' of logics $\mathbf{L}_i$, $i \in \mathcal{N}$, such that $\mathbf{L}_0$ expresses all domain theories, and $\mathbf{L}_{k+1}$ expresses all of the meta-theories for $\mathbf{L}_k$, $k \in \mathcal{N}$. At each level of the tower, an arbitrary set of theories is supported, given by sets of premises (i.e. KBs). A generalized reflection principle can be stated as follows: for $i \in \mathcal{N}$,

$$\mathrm{KB}_k \vdash_{\mathbf{L}_i} \alpha \text{ iff } \emptyset \vdash_{\mathbf{L}_{i+1}} \beta_{i+1}(\ulcorner \mathrm{KB}_k \urcorner, \ulcorner \alpha \urcorner),$$

where $\beta_{i+1}(\cdot, \cdot)$ is a fixed 2-place expression of L_{i+1} (with 'holes' for L_{i+1} terms), $\beta_{i+1}(\ulcorner \mathrm{KB}_k \urcorner, \ulcorner \alpha \urcorner)$ is a formula of L_{i+1}, and $\ulcorner \mathrm{KB}_k \urcorner$ and $\ulcorner \alpha \urcorner$ are meta-language expressions denoting KB_k and α in L_i, respectively.

Note that no constraints have been given regarding what kinds of logics instantiate a tower, e.g. propositional, first-order, intuitionistic, etc., or even combinations of these, as long as any particular combination satisfies the reflection principle.

C. The Tower Interface

Two operations are defined for each level of the tower. In this case, the focus is on a computational setting. The idea is to view each level of the tower as comprised of a set of knowledge-bases (KBs) which a user can interact with independently.

If the intention is to use the architecture to perform inference, the logics L_i, $i \in N$ must be those where the rules of inference defining the consequence operations C_{ni} are recursively enumerable. This ensures that the theorems of any logic of the tower can be generated computationally. The two tower operations are analogues of those presented in [Levesque 1984]. Specifically, they allow a user to *tell* a new statement to a knowledge-base, and to *ask* if a statement is a logical consequence of those contained in a knowledge-base. In the following, let $k, n \in N$ Then,

$$\textit{meta}\text{-TELL}[n](\mathrm{KB}_{n,k}, \alpha) = (\mathrm{KB}_{n,k} \cup \alpha)$$

$$\textit{meta}\text{-ASK}[n](\mathrm{KB}_{n,k}, \alpha) = \begin{cases} \textit{yes} & \text{if } \mathrm{KB}_{n,k} \vdash_{L_n} \alpha \\ \textit{no} & \text{if } \mathrm{KB}_{n,k} \nvdash_{L_n} \alpha \end{cases}$$

The operation meta-TELL[n]($KB_{n,k,a}$, a) takes $KB_{n,k} \subseteq L_n$, i.e. the *kth* KB at the nth level of the tower, and returns $KB_{n,k}$ with *a* included as a premise.

The operation meta-ASK[n]($KB_{n,k}$jt, *a)* takes $\mathrm{KB}_{n,k} \subseteq L_n$, i.e. the *kth* KB at the nth level of the tower, and returns *yes* or no, depending on whether *a* is provable or not provable from $KB_{n,k}$ in L_n, respectively. [2]

III. The First-Order Example

A. Inference and Procedural Reflection

The next problem to be discussed is where inference can be said to originate in the model, and how it is performed. By way of example, these issues are addressed for a tower based on first-order logic. As a starting guess, one can imagine taking a given tower and pairing each level with a 'black box', which would carry out all of the reasoning required for that level. However, the original motivation in this area was that sentences expressed as part of a meta-theory should be able to guide inference with respect to a theory, but so far no procedural connection between levels has been made.

To establish this connection, an approach described by Smith [1982] is considered, called *procedural reflection.* The main ideas of this approach may be summarized as follows. In procedural reflection, a processor for a language is not regarded as a 'black box', but is instead definable in terms of a program which describes it. All programs, therefore, are considered to be executed indirectly, by virtue of executing a program that represents the processor. A processor written in the same language it has been designed to implement is called *meta-circular:* 'meta', because of the property that the processor operates on formal program fragments, and 'circular', because an MCP does not, in itself, constitute a definition of the language.

The incentive for viewing computation in terms of a tower of MCPs is that it allows *reflective procedures,* written in the same language as the original program, to be included as part of the environment which executes the program. Hence, by changing the general description of the processor at level $k + 1$, for $k > 0$, the underlying program situated at level *k* can give rise to different processes. For present concerns, the analogous idea is to axiomitize the proof procedure found in the 'black box' at level *k* (used to carry out inference at level *k)* as part of the axioms of provability AP_{k+1} expressed by L_{k+1}. In fact, the idea is to axiomitize the proof procedure in such a way that the axioms themselves can be interpreted procedurally. Hence, inference performed on one level of the tower is viewed as being implemented by the level above. As with procedural reflection, where reflective procedures can be written to augment an MCP, this approach will allow sentences expressed in a meta-theory to determine how inference proceeds with respect to a theory.

The ideas outlined above appear in various forms for several Horn-Clause systems ([Bowen and Kowalski 1982], [Gallaire and Lasserre 1982], [Davis 1980], [Kramer 1985]). The novelty of this approach is the step up to full first-order logic; of particular interest is the fact that negation can be explicitly represented. This is important for control, since negation allows testing for properties that do not hold. It therefore becomes possible to block unwanted properties from entering into a resolution process. Also important is that resolution in the system presented is *non-clausal.* Therefore, the sentences used to define the MCP, express control strategies, as well as express the original theories over which these operate, can maintain much of their original form usually lost when everything must be reduced into clausal form.

B. A First-Order MCP

The (full) first-order MCP to be described is based upon the rule of Non-Clausal Resolution ([Murray 1982]). Like its clausal counterpart ([Robinson 1962]), NC-Resolution is a rule of *refutation*: given a set $A \subseteq L$ and a sentence $\alpha \in L$, if $\{A \cup \neg\alpha\}$ leads to a contradiction 'F' (falsehood), then α is said to follow from A. However, while clausal resolution resolves on unifiable, complimentary literals between clauses, NC-Resolution resolves on general, quantifier-free formulas containing unifiable atoms of opposite *polarity.* An important property of NC-Resolution, therefore, is that sets of formulas do not have to be converted into conjunctive normal form (although skolemization is still necessary); the standard connectives ($\wedge$, $\vee$, $\supset$, $\equiv$, $\neg$) are all available for resolution.

The first step in the design of the MCP is the selec-

[2] Only logics where the relation h is recursive are guaranteed to return an answers to arbitrary *ask* operations. Where h is r.e., but not recursive, answers are in general returned only in the case of a *yes.*

tion of a proof procedure. The one adopted combines NC-Resolution in a *connection graph* format. One of the attractive features of these graphs, from the point of view of obtaining an MCP, is that they provide a convenient way of organizing the sentences of a theory. Informally, a set of well-formed formulas *S* can be represented as a connection graph < V, *E* >, where each *vertex* vi, ε *V* denotes one of the formulas s, ε 5, and each *edge Ik* ε *E,* joining a pair of vertices in *V,* represents a potential resolvent. The connection graph refinement to Robinson's resolution principle was introduced by Kowalski [1975]. Each vertex, therefore, denotes a formula in clausal form, and edges occur between those vertices containing unifiable, complimentary literals. The combination of NC- and Connection Graph Resolution is described in [Stickel 1982]. In this case, *V* denotes a set of quantifier-free formulas, and *E* is a set of edges joining vertices of *V* that contain unifiable atoms of opposite polarity.

A resolution step in < *V, E* > consists of selecting one of the edges I_k E *E,* and forming the resolvent *ncr* associated with the ends of I_k. A new graph < *V'E'* > is then constructed by adjoining *ncr* to V, and by including among *E* the set of edges which *ncr* is said to 'inherit' from its two parent vertices. [3] It should be noted that there is no *a prion* constraints on how edges should be selected to obtain a proof, only that they should be selected such that the completeness of the rule is preserved.

The previous discussion can be summarized in the form of two meta-language definitions, considered as a partial specification of the MCP: [4]

(i) $\forall V, E, \alpha\ (\mathit{provable}(V, E, \alpha) \equiv$
$(\exists V', E', \alpha'\ (\mathit{skolem}(\mathit{not}(\alpha), \alpha') \wedge$
$\mathit{initialize\text{-}graph}(V, E, \alpha', V', E') \wedge$
$\mathit{cg\text{-}resolve}(V', E'))))$

(ii) $\forall V, E\ (\mathit{cg\text{-}resolve}(V, E) \equiv \mathit{icon}(V) \vee$
$(\exists l, ncr, V', E'\ (\mathit{select\text{-}edge}(V, E, l) \wedge$
$\mathit{nc\text{-}resolve}(l, ncr) \wedge \mathit{inherit\text{-}node}(ncr, V, V') \wedge$
$\mathit{inherit\text{-}edges}(l, ncr, E, E') \wedge$
$\mathit{cg\text{-}resolve}(V', E'))))$.

Note that *provable* defines the overall proof procedure: by way of separate definitions, *skolem* and *initialize-graph* describe the addition of the skolemized form of ->a to the graph < *V, E* > leading to the extended graph < *V',E'* >. Next, *cg-resolve* defines the steps involved in a deduction. The definition of *select-edge* represents the selection of an edge / from the set of edges *E.* With respect to *nc-resolve,* the term *ncr* denotes the NC-resolvent created by resolving on the atoms associated with the ends of /. The inclusion of *ncr* as a new vertex of the graph (mapping *V* to *V)* is represented by *inherit-node,* while the corresponding operation for new edges (mapping *E* to *E')* is represented by *inherit-edges.* Testing for a contradiction is represented by *icon*, which looks for 'F' among the vertices of *V*.

All further definitions required in order to give a complete specification of the MCP follow as sub-definitions of either *provable* or *cg-resolve.* All MCP sub-definitions are considered to be defined in an analogous way to those given above, e.g. $\forall \alpha, \alpha'\ (\mathit{skolem}(\alpha, \alpha') \equiv \ldots)$ or

$$\forall V, E, l\ (\mathit{select\text{-}edge}(V, E, l) \equiv \ldots).$$

C. Execution

The remaining question is how NC-Resolution on the MCP simulates the proof procedure. The approach uses a procedural interpretation on formulas and a form of parameter-passing similar to that found in many Horn-clause systems. That is, literals of an MCP definition are refuted in a left-to-right order, with ground terms instantiated for variables at each step. Some subtlety is involved, however, since the specialized form of reasoning is used only with respect to copies of the MCP; to ensure completeness, general first-order theories must be treated more generally.

To illustrate, an NC-Resolution process associated with the MCP is initiated by a resolution step on an atomic sentence of the form:

$$\neg \mathit{provable}(\ulcorner V \urcorner, \ulcorner E \urcorner, \ulcorner \alpha \urcorner),$$

where $\ulcorner V \urcorner$, $\ulcorner E \urcorner$, and $\ulcorner \alpha \urcorner$ are (ground) meta-language terms which designate a set of vertices V, a set of edges E, and an object-language formula α. This sentence might be said to correspond to the initial 'state' of the process, with successive resolvents corresponding to successive states. The only major rule for performing resolution steps is that successive resolvents (i.e. states) are always created by resolving the left-most literal of the current state against its corresponding MCP definition. Therefore, the first resolution step consists of resolving $\neg \mathit{provable}(\ulcorner V \urcorner, \ulcorner E \urcorner, \ulcorner \alpha \urcorner)$ against its definition in (i). Instantiation by a most general unifier $\{\ulcorner V \urcorner / V, \ulcorner E \urcorner / E, \ulcorner \alpha \urcorner / \alpha\}$ leads to the NC-resolvent (and next state):

$\neg(\mathit{skolem}(\mathit{not}(\ulcorner \alpha \urcorner), \alpha') \wedge$
$\mathit{initialize\text{-}graph}(\ulcorner V \urcorner, \ulcorner E \urcorner, \alpha', V', E') \wedge$
$\mathit{cg\text{-}resolve}(V', E'))$.

The result of the next series of resolution steps is the 'binding' of α' to $\ulcorner \alpha' \urcorner$, the term designating the formula(s) representing the skolemized form of $\neg\alpha$. This series is initiated by resolving $\neg \mathit{skolem}(\mathit{not}(\ulcorner \alpha \urcorner), \alpha')$ against its associated MCP definition. Once all states have been passed through to introduce and resolve away all of the atoms associated with the definition of *skolem*, the next intermediate state reached is

$\neg(\mathit{initialize\text{-}graph}(\ulcorner V \urcorner, \ulcorner E \urcorner, \ulcorner \alpha' \urcorner, V', E') \wedge$
$\mathit{cg\text{-}resolve}(V', E'))$.

Another series of resolution steps then results in an instantiation of $\ulcorner V' \urcorner$ for V' and $\ulcorner E' \urcorner$ for E', representing the addition of the skolemized form of $\neg\alpha$ to $< V, E >$.

[3] A property of edge inheritance is that the edge /* used to form the resolvent at each step is removed from the graph. However, since the interest in connection graphs is on the data structure they provide, rather than on defining a new rule of inference, this particular aspect is ignored.

[4] Note that quantifiers have been included for readability.

The single remaining literal,

$$\neg cg\text{-}resolve(\ulcorner V' \urcorner, \ulcorner E' \urcorner)$$

indicates the start of the deductive process on the graph $< V', E' >$. With respect to the MCP, this process is initiated by a resolution step between this literal and its definition in (ii).

The procedural interpretation placed on the MCP definitions corresponds to a form of depth-first search, so a question of completeness arises. In particular, it must be possible for the MCP to interact in a complete way with general object-language theories. Search strategies expressed by the MCP are found as part of the definition of *select-edge*, a sub-definition of *cg-resolve.* Note that *E* might be regarded in terms of an ordered list or *schedule:* selecting the first element of *E,* and inheriting new edges by mapping them onto the front of *E,* results in depth-first search (DFS). Similarly, keeping inheritance the same, and selecting edges from the back of *E,* results in breath-first search (BFS).

In the full tower, each logic L_{i+1}, $i > 0$, expresses the provability axioms AP_{i+1} by a copy of the MCP. Now, the kind of search strategy expressed by the MCP found at a particular level depends on whether resolution is associated with some general theory, or a theory representing another copy of the MCP. By default, all general theories are expressed by L_o, and completeness is maintained for them by having the version of *select-edge* included in L_1 satisfy the general strategy of BFS. For the rest of the tower L_k, $k > 1$, which by default operate on other copies of the MCP, *select-edge* in L_k satisfies a special form of DFS (corresponding to the proof procedure).

IV. Advancing Control

In this final section, a cursory overview is presented on two extensions to first-order reflective tower outlined in §111 to allow more specialized forms of control. For a more complete description, the reader is referred to [Lownie 1990].

A- Control Strategies

Edge schedules are often useful for implementing general domain-independent control strategies such as depth-first and breadth-first search. However, trying to design more complicated domain-dependent strategies based exclusively on ordering edges in a schedule can be difficult. A more structured approach is to allow control strategies, expressed for a copy of the MCP as part of *select-edge(V,E, l)*, to index the edges of a graph $< V, E >$ through its set of vertices *V.* Accordingly, the vertices v_i £ *V* act in the role of 'goals' within a deduction, while the atoms a^* £ v_i, act in the role of 'sub-goals'. Note that an edge *l* ε *E* can be described in terms of a 5-tuple:

$$l = [v_0|a_0|v_1|a_1|mgu],$$

where a_0 and a_1 occur with opposite polarity in v_0 and v_j respectively, and *mgu* is a most general unifier that unifies them. A new selection policy, therefore, can be based on indexing through the components of an edge as follows:

- select (in order) a combination of
  - (i) a *goal* $v_i \in V$, a *sub-goal* $a_k \in v_i$,
  - (ii) an edge $l \in E$ incident on v_i, a_k.

Of course, other ways of indexing edges are possible, e.g. selecting two goals followed by two sub-goals. Additionally, other structures can be used to extend this approach; for example, (i) a 'mini-scheduler' I, to allow control information to be communicated between resolution steps, e.g. in the form of complete edges or various components of edges, and (ii) a 'history' *H*, providing a record of the resolution steps performed, and used, for example, to implement selective backtracking, or to pursue multiple lines of reasoning.

B. Subsidiary Deduction Rules

A subsidiary deduction rule can be described as a property of the meta-theory in the role of an inference rule. For example, [Stickel 1986] presents a general (sound) deductive principle called *theory resolution* which can be used to test for inconsistency among selected sets of clauses or literals. Theory resolution allows specialized procedures to be included alongside general resolution to perform, for example, taxonomic reasoning or reasoning with partial-orders.

It is possible to express subsidiary deduction rules in the style set down for *cg-resolve*. Each subsidiary rule, therefore, comes equipped with its own set of edges (also extendible by a mini-scheduler and history) for expressing its own individual control strategies. All rules associated with a particular meta-theory (mKB), however, share the same set of vertices. Let $sdr_0, \ldots, sdr_p$ denote a set of $(p+1)$ subsidiary deduction rules contained in KB_k of the logic L_n $(n > 1)$, for $n, k, p \in \mathcal{N}$. Then sdr_i, $0 \leq i \leq p$, may be defined as follows:

$$\begin{aligned}\forall V, E_i\ (sdr_i(V, E_i) \equiv\ & icon_i(V) \vee \\ & (\exists l, res, V, E_i\ (select\text{-}edge_i(V, E_i, l) \wedge \\ & \quad sdr_i\text{-}resolve(l, res) \wedge inherit\text{-}node_i(res, V, V') \wedge \\ & \quad\quad inherit\text{-}edges_i(l, res, E_i, E_i') \wedge \\ & \quad\quad\quad sdr_i(V', E_i')))).\end{aligned}$$

To each object-language inference there corresponds the particular rule selected to perform it. Since any language L_i, $i \in \mathcal{N}$, can be isolated in terms of an object-language, this view naturally extends upward in the tower. To select and apply rules requires that a meta-language description of them be made available to the processor. A *rule sequence* RS has the form: $[\ \ulcorner V \urcorner^2 \mid cgr \mid rs_0 \mid \ldots \mid rs_p\ ]$, where each element *cgr* or rs_i is itself a 2-tuple: $(0 \leq i \leq p)$

$$[\ \ulcorner cg\text{-}resolve \urcorner \mid \ulcorner E_{cgr} \urcorner^2\ ] \quad \text{or} \quad [\ \ulcorner sdr_i \urcorner \mid \ulcorner E_i \urcorner^2\ ]$$

The first element of each 2-tuple designates the name of the rule, while the second element refers to its set of edges. (Note that $\ulcorner V \urcorner^2$ refers to two levels of designation above *V*; similarly for $\ulcorner E_i \urcorner^2$.)

With *cg-resolve* no longer the only rule available to a resolution process, it is no longer tied directly into the definition of the MCP:

$$\forall V, E, RS\ (mcp^{2+}(V, E, RS) \equiv icon(V) \vee (\exists S, V', E', RS'\ (select\text{-}rule(V, E, RS, S) \wedge cg\text{-}apply(V, E, RS, S, V', E', RS') \wedge mcp^{2+}(V', E', RS'))))).$$

Briefly, *select-rule* represents the selection S of one of the 2-tuples $[\ulcorner cg\text{-}resolve \urcorner \mid \ulcorner E_{cgr} \urcorner^2]$ or $[\ulcorner sdr_i \urcorner \mid \ulcorner E_i \urcorner^2]$ from RS, while *cg-apply* represents a single application of the selected rule, which results (in part) in the formation of a new rule sequence RS' given by either

$$[\ulcorner V' \urcorner^2 \mid [\ulcorner cg\text{-}resolve \urcorner \mid \ulcorner E'_{cgr} \urcorner^2] \mid \ldots]$$

or

$$[\ulcorner V' \urcorner^2 \mid \ldots [\ulcorner sdr_i \urcorner \mid \ulcorner E'_i \urcorner^2] \ldots].$$

Copies of map^{2+} are included into the tower by appending its definition as part of AP_i, for i > 2. In terms of upward compatibility in the tower, it is possible to view mop^{2+} as a rule which can be selected and applied like any other. Further consideration in this area involves: (i) a precise description how rules can be applied; (ii) 'cross inheritance', by which new edges for a rule are generated as a result of resolvents created by other rules; and (iii) an updated definition of provable which takes into account the the addition of copies of mcp^2+. The details, while lengthy, are mostly straightforward ([Lownie 1990]).

V. Summary

This paper outlined a reflective architecture for declarative languages, which includes Smith's procedural model ([1982]) as a special case for first-order languages based on Murray's NC-Resolution rule ([1982]). Its main contribution over existing approaches is how this leads to various forms of selective control for KB-systems based on (full) first-order logic.

Several other topics warrant further consideration; for example, implementations which minimize computational overhead incurred by reflective inference by explicitly representing only those parts of the processor required for control. These are left to the later paper ([Lownie 1990]). However, a good reference which can found in this area is [des Rivieres and Smith 1984].

Acknowledgments

The author would like to thank, at the University of Toronto, Hector Levesque, Jim des Rivieres, Ray Reiter, and John Mylopoulos, and at Queen's University, Peter O'Hearn, for their comments and suggestions regarding earlier drafts of this work.

References

[Bowen and Kowalski 1982] Bowen, K.A., and Kowalski, R., "Amalgamating Language and Metalanguage in Logic Programming," in Clark, K., and Tarnland, S. (eds.), Logic Programming, Academic Press, New York, pp. 153-172, 1982.

[Davis 1980] Davis, R., "Meta-Rules: Reasoning About Control," Artificial Intelligence 15 (3), pp. 179-222, 1980.

[de Rivieres 1984] des Rivieres, J., and Smith, B., "The Implementation of Procedurally Reflective Languages," Xerox PARC, Report ISL-1, Palto Alto, CA, 1984.

[Feferman 1962] Feferman, S., "Transfinite Recursive Progressions of Axiomatic Theories", Journal of Symbolic Logic 27, pp. 259-316, 1962.

[Gallaire and Lasserre 1982] Gallaire, H, and Lasserre, C, "Metalevel Control for Logic Programming," in Clark, K., and Tarnland, S. (eds.) Logic Programming, Academic Press, New York, pp. 173-185, 1982.

[Genesereth and Smith 1983] Genesereth, M.R., and Smith, D.E., "An Overview of Meta-Level Architecture," AAAI-83, Washington, D.C., pp. 119-124, 1983.

[Kowalski 1975] Kowalski, R., "A Proof Procedure Using Connection Graphs," J ACM (22) (4), pp. 572-595, 1975.

[Kramer 1984] Kramer, B.M., "Representing Control Strategies Using Reflection," CSCSI-84, London, Ontario, 1984.

[Levesque 1984] Levesque, H. J., "Foundations of a Functional Approach to Knowledge Representation," Artificial Intelligence 23 (2), pp. 155-212, 1984.

[Lownie 1990] Lownie, T.M., in preparation, 1990.

[Murray 1982] Murray, N.V., "Completely Non-Clausal Theorem Proving," Artificial Intelligence 18 (1), pp. 67-85, 1982.

[Robinson 1965] Robinson, J.A., "A Machine-Oriented Logic Based On The Resolution Principle," J A CM 12 (1), pp. 23-41, 1965.

[Smith 1982] Smith, B.C., "Reflection and Semantics in a Procedural Language," MIT/LCS/ TR-272, Massachusetts Institute of Technology, Cambridge, MA, 1982.

[Smith 1986] Smith, B.C., "Varieties of Self-Reference," in Theoetical Aspects of Reasoning about Knowledge, Morgan Kaufmann Publishers, Los Altos, CA, pp. 19-43, 1986.

[Stickel 1982] Stickel, M.E., "A Nonclausal Connection-Graph Resolution Theorem-Proving Program," AAAI-82, Pittsburgh, PA, pp. 229-233, 1982.

[Stickel 1986] Stickel, M.E., "Automated Deduction by Theory Resolution," Technical Note 340, SRI International, Menlo Park, CA, 1986.

[Weyhrauch 1980] Weyhrauch, R.W., "Prolegomena to a Theory of Mechanized Formal Reasoning," Artificial Intelligence 13 (1)(2), pp. 133-170, 1980.

[Wojcicki 1977] Wojcicki, R., "Strongly Finite Sentential Calculi," in Wojcicki, R., and Malinowski, G (eds.) Selected Papers on Lukasiewicz Sentential Calculi, Institute of Philosophy and Sociology, Polish Academy of Sciences, pp. 53-77, 1977.