The Restricted Language Architecture of a Hybrid Representation System

Marc Vilain

BBN Laboratories
10 Moulton St.
Cambridge, MA 02238

Abstract. Hybrid architectures have been used in several recent knowledge representation systems. This paper explores some distinctions between various hybrid representation architectures, focusing in particular on systems built around restricted representation languages This restricted language architecture is illustrated by describing KL-TWO, a hybrid reasoner based on the restricted representation facility RUP. The bulk of this paper discusses KL-TWO, its subcomponents, and the techniques used to interface them.

Many recent knowledge representation systems have been based on hybrid architectures. These systems are hybrids in that they do not attempt the entire knowledge representation task with a single inferential component. Instead, they combine several reasoners into a complex whole. The advantages to be gained by this hybrid approach vary from system to system, but often include increases in the system's computational efficiency, the coverage of its representation language, or the ease of expressing knowledge with the system.

The topic of this paper is a new logic-based hybrid representation system. This system, called KL-TWO, is one which I have been developing with my colleagues over the past two years. Although KL-TWO has features in common with other hybrid reasoners, it differs very significantly from other systems in the way the common features are used. The purpose of this paper is thus twofold, to describe the features that KL-TWO possesses, and to stress the differences in approach between KL-TWO and other hybrid reasoners.

Hybrid Representation Architectures

The hybrid representation scheme most familiar to the general AI audience is that embodied by such systems as KRYPTON (Brachman, Fikes, & Levesque, 1983) or the theorem-proving system of Schubert and his colleagues (Schubert, Papalaskaris, & Taugher, 1983). These systems start with a theorem prover for first-order logic (FOL), and augment it with special-purpose inference systems. These special-purpose components are designed to perform efficiently certain forms of reasoning that would be computationally expensive for the theorem prover to perform by itself. In essence, for their restricted domains, these augmentations bypass the normal proof procedure of the theorem prover, and cut out some of its combinatorial search. For example, KRYPTON augments Stickel's connection graph theorem prover with a device for terminological reasoning. Schubert and his colleagues added to their theorem prover components to reason about types, part-whole relationships, temporal ordering, and color. In this paper, I will refer to systems such as these as *theorem prover-based hybrid reasoners.*

One important characteristic of these theorem prover-based hybrids is that the hybrid architecture may enhance the overall efficiency of the system, but will not change its overall expressiveness. Anything which can be represented in the special-purpose augmentations could also be represented in the unadorned theorem prover (albeit less conveniently or efficiently). The augmentations don't extend the expressive power of the theorem prover as much as they circumvent its computational inefficiencies.

KL-TWO has a substantially different architecture from that of theorem prover-based systems. This alternative architecture was first suggested by McAllester (1980, 1982). He argued that one need not base a representation system on a full inference system for the predicate calculus. Indeed, given that full first-order logic is only semi-decidable, and that even the best theorem-provers are prone to time-consuming search, there is good reason to consider alternative representation schemes. McAllester proposed building knowledge representation systems around a fairly simple restricted device, his Reasoning Utility Package (RUP for short). RUP provides only a subset of the inferential power of a first-order theorem prover, (roughly the propositional subset) but is computationally much more efficient. To restore some of the missing representational power, one extends RUP with special-purpose reasoners tailored to one's application.

Other restricted languages have been proposed for this kind of hybrid system (for example, the relevance versions of propositional logic and FOL proposed by Levesque (1984) and Patel-Schneider (1985)). I use the term *restricted language hybrid reasoners* to refer to hybrids built around restricted systems such as those of McAllester, Levesque, or Patel-Schneider.

The advantages of the restricted language approach become apparent if the subcomponents that extend the central reasoner are themselves restricted but efficient. The resulting system will not have the full representational expressiveness of a first-order language, but in exchange it will have the potential for reasoning efficiently. This approach has a very attractive promise: if one can make do with the expressive limitations of a restricted language hybrid, one may have a very efficient system indeed.

[1]This restorer, wot tupporttd by the Advonct Rtttorch Projact Agency of tht Dtportment of Dtftntt. Under Contract No. NO0014-77-C-6378.

KLrTWO

KL-TWO it an experiment with the restricted language approach to hybrid reasoning. The KL-TWO hybrid is composed of two subreasoners: PENNI, a modified version of RUP, and NIKL, a terminological reasoner that descended from KL-ONE. In the pages that follow I will describe both PENNI and NIKL in some detail, and will explain how the two systems are interfaced. But first let me give a general description of KL-TWO for the reader to keep in mind through the discussion ahead.

KL-TWO'a subcomponents, PENNI and NIKL, are complimentary reasoners. The representation task is divided up between them, roughly as follows.

PENNI contains a database of propositions assertions. The language of PENNI is essentially the propositional subset of first order predicate calculus (a more precise definition follows). This language does not contain any quantification.

PENNI's simple propositional framework is extended by KL-TWO's second subreasoner, NIKL. NIKL is a terminological reasoner similar to KL-ONE (Brachman & Schmolze, 1985) or the terminological component of KRYPTON (Brachman et al.. 1983). As I will explain below, the terminological statements one can make in NIKL define a simple class of universally quantified sentences. These sentences can be applied in PENNI to extend PENNI's propositional language with a limited form of quantification.

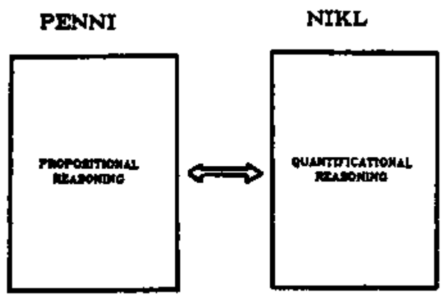

Figure 1: The architecture of KL-TWO

The architecture of KL-TWO can be summarized in this way: PENNI provides propositional reasoning and NIKL provides limited quantificational reasoning. This is illustrated by Figure 1.

PENNI

PENNI is a modified version of McAllester's RUP (1980; 1982). The features of PENNI I will describe in this section are thus originally features of RUP.

At the heart of PENNI is a database of propositions. The language of this database is the quantifier-free predicate calculus with equality. This language is defined as containing:

1. Expressions of the form (P a), (Q a b),... at in (MAN John) or (OFFSPRING John Mary);
2. Expressions of the form (= a b), as in (= (grade Bill) B+),
3. Boolean combinations of (1) and (2), art in (-> (FATHER John) (PERSON John)).

This language doesn't contain any quantifiers, I will say more about this below.

The database permits incremental assertions and retractions. Propositions can be added in any order; following each addition a number of deductions will be made. Retractions can also be performed at any time. The RUP database is built as a truth maintenance system (or TMS), and permits all the useful operations that have been associated with such systems. These include the ability to justify a deduction by returning the exact set of user-asserted propositions that entail it, the ability to retract efficiently the logical consequences of a proposition when that proposition is itself retracted, and the ability to perform fast dependency-directed backtracking. These features are extraordinarily useful: eloquent arguments for them have been made by McAllester (1980; 1982), Doyle (1978), and others.

I mentioned above that the language of PENNI does not include quantification. The decision not to include quantifiers was made by McAllester when he designed RUP. and follows from his view of logical reasoning as consisting of two separate processes, deduction and instantiation. Deduction is the process of deriving the consequences of a set of propositions, whereas instantiation is the process of applying quantified sentences to produce new propositions. McAllester (1980) argues that within certain constraints deduction can be performed relatively efficiently. Instantiation, on the other hand, is very difficult: often the choice of which sentences to instantiate, and when to do so, can only be determined efficiently on the basis of the specific reasoning domain. The approach embodied by RUP (and hence by PENNI) is to focus on providing a good efficient mechanism for deduction,[2] and to remain uncommitted as to how to perform instantiation.

In RUP, the choice of how to instantiate quantified sentences is left up to the user. This is facilitated by providing "hooks" into the database that allow the user to augment its language. The hooks are realized by the mechanism of demon invocation; RUP provides several varieties of IF-ADDED and IF-NEEDED demons. The version of RUP used in KL-TWO contains McAllester's demon invocation mechanisms. However, as I mentioned above, KL-TWO can provide a class of quantified inferences efficiently and automatically, without requiring the use of demons. These inferences are performed by NIKL, KL-TWO's second subreasoner.

NIKL

NIKL[3] is a terminological reasoner that descends from KL-ONE (Brachman & Schmolze 1985). As with KL-ONE, NIKL allows one to make statements about the *terms* that describe a domain.

NIKL distinguishes two kinds of terms, Concepts and Roles. Semantically. these correspond respectively to 1-place and 2-place relations. For example, the NIKL Concept PERSON corresponds to the predicate that is

[2]RUP's forward-chaining deduction algorithm is very efficient. It is incomplete, however: it fails to perform certain inferences, roughly those that require case analysis. RUP does provide another deduction algorithm which extends the first one to produce a complete proof procedure.

[3]By necessity, this description of NIKL is rather simplified and terse. For more details, see Moser (1983), and Schmolze & Israel (1983).

true of all entities which are persons The NIKL Role OFFSPRING corresponds to the relation that holds between any individual and its children

Primitive and Complex Terms

NIKL further distinguishes between primitive and complex terms. Primitive terms are simply terms for which no definition can be given. PERSON, for example, is a primitive Concept, since the criteria for being a person defy complete description. Similarly, OFFSPRING is a primitive Role.

In NIKL, complex terms are built out of other terms (which themselves can be simple or complex). To create complex terms, NIKL provides a set of term constructors which combine Concepts and Roles to produce new Concepts and new Roles. Several of these constructors, along with their intended semantics are summarized in Figure 2.

NIKL Expression, x	Semantics of x, $[\![x]\!]$
(CMeet C_1 C_2)	$\lambda x\ ([\![C_1]\!]\ x)\ \&\ ([\![C_2]\!]\ x)$
(CMin R n)	$\lambda x.\ \exists\ n$ distinct $y_i\ \&_i\ ([\![R]\!]\ x\ y_i)$
(CRestrict R C)	$\lambda x\ \forall\ y\ ([\![R]\!]\ x\ y) \rightarrow ([\![C]\!]\ y)$
(VRDiff R C)	$\lambda x.y.\ ([\![R]\!]\ x\ y)\ \&\ ([\![C]\!]\ y)$

Figure 2: Complex term constructors

For example, consider the complex Concept

(CMeet PARENT MAN) (1)

This Concept corresponds in turn to a complex predicate which can be notated with the λ-expression

$\lambda x.$ (PARENT x) & (MAN x)

This complex predicate holds true of everything for which the PARENT and MAN predicates hold true (e.g., it holds true of fathers). Similarly, the complex Role

(VRDiff OFFSPRING MALE) (2)

corresponds to the 2-place λ-abstracted relation

$\lambda x.y.$ (OFFSPRING x y) & (MALE y)

This relation holds between something and any of its OFFSPRING which are MALE (e.g., between something and its sons).

Naming terms

To make complex terms more convenient, NIKL allows them to be assigned names. For example, the complex terms I have just described (formulas 1 and 2) can be assigned their rightful names by means of the following NIKL statements.

(Let FATHER (CMeet PARENT MAN)) (3)
(Let SON (VRDiff OFFSPRING MALE)) (4)

Note that assigning a name to a complex term is tantamount to giving a definition to that name. This definition can be interpreted as a first-order sentence. For example, the assignments in formulas 3 and 4 correspond to the following definitions.

$\forall$ x (FATHER x) $\leftrightarrow$ (PARENT x) & (MAN x)
$\forall$ x,y (SON x y) $\leftrightarrow$ (OFFSPRING x y) & (MALE y)

For more examples of NIKL definitions, see Figure 3.

Subsumption and Classification

The semantics of NIKL lead naturally to a relation that holds between NIKL terms, that of *subsumption*. A term subsumes another term if the meaning of the second entails the meaning of the first. More precisely, in the case of Concepts, we say that a Concept C_1 subsumes a Concept C_2 iff

$$\models \forall\ x\ (C_2\ x) \rightarrow (C_1\ x)$$

Subsumption between Roles has a similar definition.

Considering the examples from figure 3, we can see that the Concept ANIMAL directly subsumes the Concepts HERBIVORE, CARNIVORE, and BIG-ANIMAL. There are other Concepts that ANIMAL subsumes in a derived way, for example BIG-CARNIVORE, or the Concept

(CMeet HERBIVORE CARNIVORE)

NIKL Let expression	Semantics of Concept term	Definitional interpretation
(Let HERBIVORE (CMeet ANIMAL (CMin (VRDiff EATS PLANT) 1)))	λx (ANIMAL x) & $\exists$ y (EATS x y) & (PLANT y)	$\forall$ x (HERBIVORE x) $\leftrightarrow$ (ANIMAL x) & $\exists$ y (EATS x Y) & (PLANT y)
(Let CARNIVORE (CMeet ANIMAL (CMin (VRDiff EATS ANIMAL) 1)))	λx (ANIMAL x) & $\exists$ y (EATS x y) & (ANIMAL y)	$\forall$ x (CARNIVORE x) $\leftrightarrow$ (ANIMAL x) & $\exists$ y (EATS x y) & (ANIMAL y)
(let BIG-ANIMAL (CMeet ANIMAL (CRestrict SIZE BIG)))	λx (ANIMAL x) & $\forall$ y (SIZE x y) $\rightarrow$ (BIG y)	$\forall$ x (BIG-ANIMAL x) $\leftrightarrow$ (ANIMAL x) & $\forall$ y (SIZE x y) $\rightarrow$ (BIG y)
(Let BIG-CARNIVORE (CMeet CARNIVORE BIG-ANIMAL))	$\lambda x.$ (CARNIVORE x) & (BIG-ANIMAL x)	$\forall$ x (BIG-CARNIVORE x) $\leftrightarrow$ (BIG-ANIMAL x) & (CARNIVORE x)

Figure 3: NIKL Examples

which would appropriately be named OMNIVORE. Stating these subsumption relations in terms of first-order sentences, we have

$\forall$ x (CARNIVORE x) $\rightarrow$ (ANIMAL x)
$\forall$ x (BIG-CARNIVORE x) $\rightarrow$ (ANIMAL x)
and so forth.

Subsumption relations in NIKL are computed in an operation known as classification. Given a NIKL term To and a database of (pre-classified) NIKL terms, the classifier locates all the database terms which subsume To. This computation effectively installs To in the NIKL database; to allow for efficient subsumption queries, the computation is then cached.

For example, classifying the OMNIVORE Concept (i.e. (CMeet HERBIVORE CARNIVORE)) with respect to the database in Figure 3 would install OMNIVORE in the database. It would be given as subsumers the Concepts HERBIVORE, CARNIVORE, ANIMAL, and all other Concepts that subsume these.

In addition to computing subsumption relations, the classifier performs a canonicalization operation on the term it is classifying. For example, the Concept

```
(CMeet ANIMAL (CRestrict SIZE BIG)
              (CMin (VRDiff EATS ANIMAL) 1))
```

would canonicalize to the pre-existing Concept

```
(CMeet BIG-ANIMAL CARNIVORE)
```

which is the Concept that has been named BIG-CARNIVORE.

The subsumption relation and the operation of classification provide NIKL with much of its representational strength. In KL-TWO, they also play a crucial role in the interface between NIKL and PENNI.

Interfacing PENNI and NIKL

As I described above, PENNI holds a database of grounded propositions, and NIKL contains definitions which can be read as universally quantified sentences. Terms in NIKL (such as the Concept HERBIVORE) correspond to the predicate names of propositions in PENNI (such as HERBIVORE in (HERBIVORE rabbit-1)). The KL-TWO interface exploits this correspondence: it operates by applying in PENNI sentences from NIKL.

Consider for example the database in Figure 3. Among the sentences derivable from that database is

$$\forall x\ (\text{HERBIVORE } x) \rightarrow (\text{ANIMAL } x) \qquad (5)$$

If (HERBIVORE rabbit-1) were asserted in PENNI, then by applying formula 5 in PENNI, KL-TWO could infer automatically (ANIMAL rabbit-1).

The key issue here is determining which NIKL sentences should be instantiated in PENNI, and when this instantiation should take place. In KL-TWO, these decisions are arrived at using a carefully controlled mixed regimen of forward and backward reasoning. The forward reasoning takes place when a new proposition is asserted of an individual: the forward reasoner is invoked to determine which NIKL sentences may as a result be applied to the individual. The backward reasoner is invoked to answer a query: it will apply sentences that have been selected as appropriate by the operation of the forward reasoner.

The Forward Reasoner

The forward reasoner is brought into action when new propositions are asserted of some PENNI individual I. Its job is to combine all that has been asserted of I, and determine from this which sentences can be applied to I (without actually applying them). It does so by creating for I a NIKL Concept called the MSG of I (for Most Specific Generalization of I). This Concept captures an abstraction of that which is known of I, and in turn determines which NIKL sentences can be applied to I. These sentences are simply those that encode the subsumption relations between the MSG of I and other Concepts in the NIKL database.

This is best clarified by an example. Consider again the NIKL database of Figure 3. Let's say now that the following propositions were added to PENNI.

```
(BIG-ANIMAL al)
(ANIMAL a2)
(EATS al a2)
```

This might correspond to the situation where you're in the African plains, and you've just noticed near you some big animal eating another animal

The forward reasoner abstracts these propositions into the Concept Caf, the MSG for al:

```
(CMeet BIG-ANIMAL
       (CMin (VRDiff EATS ANIMAL) 1))
```

To see that this abstraction is valid, note that semantically it corresponds to

$$\lambda x.\ (\text{BIG-ANIMAL } x)\ \&\ \exists y\ (\text{EATS } x\ y)\ \&\ (\text{ANIMAL } y)$$

Applying this λ-expression to al yields the valid proposition

(BIG-ANIMAL al) & 3 y (EATS al y)
& (ANIMAL y)

By constructing Ca1, the interface has effectively abstracted what is known of al. To determine which sentences may be applied to al, the interface next classifies Caf. The classifier in turn locates every existing Concept which subsumes Caf, in this case CARNIVORE. BIG-ANIMAL, ANIMAL (and any other subsumers of these Concepts not indicated in Figure 3). These subsumption relations encode the sentences

$\forall$ x (Caf x) $\rightarrow$ (CARNIVORE x)
$\forall$ x (Caf x) $\rightarrow$ (ANIMAL x)
and so forth

There is actually one more sentence which is applicable here, and it is determined by virtue of the classifier's canonicalizing Caf. Indeed, given the Figure 3 database, Caf canonicalizes to

```
(CMeet CARNIVORE BIG-ANIMAL)
```

which is the Concept named BIG-CARNIVORE. In essence the canonicalization operation revealed

$$\forall x\ (\text{BIG-CARNIVORE } x) \leftrightarrow (\text{Caf } x) \leftrightarrow \exists y\ (\text{EATS } x\ y)\ \&\ (\text{ANIMAL } y)\ \&\ (\text{BIG-ANIMAL } x)$$

The forward reasoner has now discovered all NIKL sentences applicable to al. At this point it chooses to instantiate only one of these, the one derived by canonicalization. This is done by asserting in PENNI

```
((BIG-ANIMAL al) & (EATS al a2) & (ANIMAL a2))
  --> (BIG-CARNIVORE al)
```

From this, the PENNI TMS will automatically infer

(BIG-CARNIVORE al)

We call this proposition the MSG predication of al.

Because the classifier caches subsumption relations, all other NIKL sentences applicable to al will be retrievable later (as needed), and need not be instantiated now. Additionally, the MSG predication being true in PENNI will allow demons that are indexed by the proposition's predicate name to fire (Including, for example, a demon encoding the rule that you ought to run away from a big carnivore!) Finally, asserting the MSG predication facilitates subsequent TMS deductions which are of importance to the backward reasoner.

The Backward Reasoner

The backward reasoning operation is used to answer queries, propositions for which a truth assignment is being sought. To determine the truth of a query about an individual I, the backward reasoner will try to determine whether any NIKL sentences applicable to I entail the truth of the queried proposition.

I will assume here that the query has form (C I), where the predicate name C corresponds to a Concept of the same name[4] To determine the truth of (C I), the backward reasoner compares the Concept C to *CI,* the MSG of I. If C subsumes *CI,* then the backward reasoner may infer (C I).

To see how this works, consider again our example from the African plains. Say the forward reasoner has found the MSG of the individual a1 to be BIG-CARNIVORE, and say we query

(CARNIVORE al)

To determine the truth of this query, the backward reasoner checks whether CARNIVORE subsumes BIG-CARNIVORE. Given the database in Figure 3. this subsumption relation does indeed hold, which establishes that

$$\forall x\ (\text{BIG-CARNIVORE}\ x) \rightarrow (\text{CARNIVORE}\ x)$$

This is precisely the NIKL sentence that is needed to answer the query, and the interface instantiates it by asserting

(BIG-CARNIVORE al) --> (CARNIVORE al)

In turn the PENNI TMS will infer

(CARNIVORE al)

Conclusions

In the view of KL-TWO I have given here, KL-TWO's principal achievement can be seen as having extended RUP with some form of quantification. However, the kind of quantification that KL-TWO provides is limited, the scope of NIKL's quantificational language is the class of terminological statements that are characteristic of KL-ONE and its various descendants. Many quantified statements can not be handled in this way. In KL-TWO, these statements that can't, be expressed in NIKL must be encoded with demons attached to the PENNI database. This observation also applies to other restricted language hybrids built on RUP, in particular the CAKE system of Rich (1982).

So how well does KL-TWO measure up to the promise of the restricted language approach to hybrid reasoning? Can it be both computationally efficient and representationally sufficient? The answer unfortunately is yes and no. Yes, the system is efficient (although it relies on some incomplete algorithms), and yes, it is more expressive than RUP alone. But, as might be predicted, KL-TWO still isn't sufficient unto itself. As RUP before it. KL-TWO mutt be extended for particular applications, albeit to a lesser extent than RUP.

However, this state of affairs is representative of the inherent trade-off between expressiveness and efficiency that is common to all knowledge representation systems. Out of the many possible ways of handling this trade-off, KL-TWO makes one particular choice, one which I think has been successful.

[4]For more details on how Role queries are hondled. Vilain (1985).

References

Brachman, R. J., Fikes, R. E., & Levesque, H. J. (1983). KRYPTON. A functional approach to knowledge representation *IEEE Computer, 16(10),* 87-73.

Brachman, R. J. & Schmolze, J. G. (1985). An Overview of the KL-ONE knowledge representation system. *Cognitive Science, 9(2).*

Doyle, J. (1978). *Truth maintenance systems for problem solving* (AI Technical Report 419). MIT AI Lab.

Levesque, H. J. (1984). A logic of implicit and explicit belief. In *Proceedings of AAAI-84,* 198-202.

McAllester, D. A. (1980). *An outlook on truth maintenance* (AI Memo 551). MIT AI Lab.

McAllester (1982). *Reasoning utility package user's manual* (AI Memo 667). MIT AI Lab.

Moser, M. G. (1983). An overview of NIKL, the new implementation of KL-ONE. In Sidner (1983). 7-26.

Patel-Schneider, P. (1985). A decidable first-order logic for knowledge representation. In *Proceedings of 1JCAI-85.*

Rich, C. (1982). Knowledge representation languages and the predicate calculus. How to have your cake and eat it too. In *Proceedings of AAA1-82,* 193-196.

Schmolze, J. G. & Israel, D. J. (1983). KL-ONE: Semantics and classification. In Sidner (1983), 27-39.

Schubert, L. K., Papalaskaris, M. A.. & Taugher, J. (1983). Determining type, part, color, and time relationships. *IEEE Computer, 16(10),* 53-80.

Sidner, C. L. (Ed.) (1983). *Research in knowledge representation and natural language understanding — Annual report, 1 September 1982 - 31 August 1983.* Bolt Beranek and Newman Report No. 5421.

Vilain, M. B. (1985). *An approach to hybrid reasoning.* BBN Laboratories Report (in preparation).