Inverting Resolution with Conceptual Graphs*

Maurice Pagnucco
email : morri@cs.su.oz.au

Norman Foo
email : norman@cs.su.oz.au

Knowledge Systems Group,
Basser Department of Computer Science,
Madsen Building F09,
University of Sydney,
NSW, 2006, Australia.
fax : +61-2-692-3838

Abstract. Methods for performing inductive inference have become very important in Artificial Intelligence, especially in the area of Machine Learning. One technique capable of performing induction is based on inverting the resolution process (with the help of an oracle). This is known as inverse resolution.
In this paper we investigate how inverse resolution can be performed using conceptual graphs. This is done by showing how the individual inverse resolution operators can be implemented using conceptual graphs. We show that the processes involved can actually be viewed as inverses of beta (and alpha) rules. Also, the operations can be seen as analogues to inverse resolution operators suggested, in the literature, for predicate calculus (e.g., absorption, identification, etc.).
The advantage of this approach is that it develops a technique for performing induction using conceptual graphs. In particular, two of the operators are capable of performing constructive induction through the introduction of new relations not present in the original graphs. We also claim that the use of conceptual graphs provides a natural way of performing these operations and that this leads to a better understanding of the processes involved.
Keywords: Inductive inference, inverse resolution, machine learning, constructive induction.

1 Introduction

Techniques for performing inductive inference have become an important and useful part of Artificial Intelligence, particularly within the sub-field of Machine Learning. The basic idea behind this process is to construct a general domain theory from particular facts. One such technique is called *Inverse Resolution* [1] [2] [3] which, as the name suggests, is based on inverting the resolution process.

This framework relies on the following notion of generality within first-order logic [4] : a formula A is more general than a formula B if A logically entails

* To appear in the Proceedings of the First International Conference on Conceptual Structures, Quebec, Canada, August 1993.

(or implies) B (i.e., $A \models B$). This leads us to the following characterisation of inductive inference (see also [4]). If we have a partial domain theory T and an example E not entailed by this theory (i.e., $T \not\models E$) then we attempt to construct a new domain theory T' from T and E such that it entails both the example ($T' \models E$) and the old domain theory ($T' \models T$).

Muggleton and Buntine [3] present an implementation called CIGOL which is capable of performing the above process with respect to SLD resolution ($A \vdash_{SLD} B$)[1]. This system is based on five operators which are shown to be inverses of the resolution process. The domain theory and example are represented by a series of Horn clauses. If we can determine a new domain theory T' in this way, we say that T' was induced from T and E via inverse resolution.

One of the interesting aspects of Inverse Resolution is that two of the operators are capable of performing constructive induction. That is, they can suggest the introduction of a new predicate into the language[2]. Therefore, in a certain sense, this form of induction is not restricted by the initial vocabulary.

The main aim of this paper is to show how the process of inverse resolution can be performed using conceptual graphs. This is done by examining how the individual operators can be replicated through operations on conceptual graphs. We also show that these operations can be viewed as inverses of the *alpha* and *beta rules* used to perform deduction using conceptual graphs.

The advantage of this approach is that it introduces a technique for performing inductive inference — in particular, constructive induction — using conceptual graphs. Also, it can give us an idea of how to extend this concept to considering inverses of particular alpha and beta rules thereby providing a more general technique for performing induction using conceptual graphs. We also claim that the operations involved are naturally modelled using conceptual graphs and that they help one to better understand how the process works.

The intent of this paper is not to express the complete formalism but to convey the essence of the above ideas, investigating their feasibility. We postpone the formal proofs and implementation details to a future paper.

In the following section we introduce some basic terminology and present the inverse resolution operators. Section 3 shows how the inverse resolution operations can be performed using conceptual graphs. In section 4 we present our conclusions and make some suggestions for future work.

2 Inverse Resolution

2.1 Terminology and Notation

In this section we briefly introduce some terminology that will be useful for the exposition presented in this paper. The terminology and notation used here and

[1] Note, using the notion $A \vdash_{SLD} B$ instead of $A \models B$ constitutes a specialisation of the process presented in the previous paragraph since an implementation will be subject to restrictions not present in the theoretical framework.

[2] In this sense, *constructive induction* is also known as *predicate invention*.

in the following sections on resolution and inverse resolution is mostly due to Muggleton and Buntine [3] (see also [4]).

Terms have their usual meaning. Atoms are predicate symbols applied to terms. A positive literal is an atom while a negative literal is a negated atom. A clause is a finite set of literals; a Horn clause is a clause with at most one positive literal, and; a definite clause is a clause with exactly one positive literal. A substitution θ is a finite set of pairs $\theta = \{v_1/t_1, ..., v_n/t_n\}$ where the v_i are variables and the t_i are terms. When applied to a term, substitution replaces each occurrence of the variable v_i with the term t_i. A unifier is a substitution θ such that $t_1\theta = t_2\theta$. The unifier θ is a most general unifier (mgu) of t_1 and t_2 if there is no other unifier σ such that $t_1\sigma$ is more general than $t_1\theta$. An inverse substitution θ^{-1} maps variables to terms in $t\theta$.

2.2 Resolution

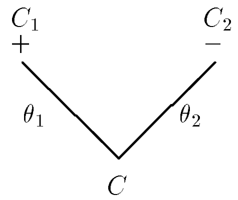
Resolution is a process which infers a clause C from the clauses C_1 and C_2. If C_1 and C_2 are two clauses with no common variables and L_1 and L_2 are literals in C_1 and C_2 respectively such that θ is the mgu of L_1 and L_2 then the resolved product (or resolvent) of C_1 and C_2 is denoted $C = C_1.C_2$ where

$$C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta. \tag{1}$$

We can also factor θ into two parts θ_1 and θ_2 such that $\theta = \theta_1\theta_2$ and $domain(\theta_i) \subseteq vars(C_i)$. This allows us to express the above as

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2. \tag{2}$$

We can visualise this process with the help of the following diagram.



The plus and minus sign denote that the literal resolved upon appears positive or negative (respectively) in that clause.

2.3 Inverse Resolution Operators

In this section we shall briefly introduce the five main inverse resolution operators. The interested reader is referred to [3] and [4] for more information.

V Operators (Absorption and Identification) The previous figure gives a "V"-shaped representation of a single resolution step. The two V operators in inverse resolution are capable of deriving one of the clauses at the arms of the V when given the clause at the other arm and the clause at the base. The *absorption* operator constructs C_2 given C_1 and C while the *identification* operator constructs C_1 given C_2 and C.

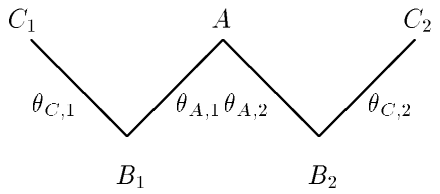
In applying these operators we have to find the inverse of a resolved product. To this end, we define the notion of the *resolved quotient* of C and C_1 as $C_2 = C/C_1$. A manipulation of equation 2 above for resolution gives us

$$C_2 = (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{L_2\} \tag{3}$$

This quotient is unique under the following *separability assumption*: the clauses $(C_1 - \{L_1\})\theta_1$ and $(C_2 - \{L_2\})\theta_2$ (see section 2.2 on resolution — in particular, equation 2) contain no common literals. Without this assumption the resolved quotient of two clauses may not be unique. We shall also adopt this assumption in this paper.

An important point to note is that, unlike resolution, the clauses produced by inverse resolution operators are not necessarily valid even though the initial clauses may be. To ensure validity of this process the notion of an *oracle* is needed. An oracle is an entity that accepts a clause if it is valid[3] and rejects it otherwise. In applying inverse resolution, the use of an oracle is essential in maintaining the consistency of the revised domain theory. Since the new domain theory entails both the old domain theory and the example, its closure is always expanding. If we accept any clause suggested by an inverse resolution operator then it may be the case that we are in fact over-generalising in the sense that there may be ground instances, of clauses within the domain theory, that are not true under the intended interpretation. We might like to think of the oracle as an abstraction of the intuitive notion of a verifying experiment. In an implementation of this system the oracle is usually the user.

W Operators (Intra-Construction and Inter-Construction) By combining two resolution "V"s we obtain the following form:



In this case we assume that a common literal L, within A, resolves with C_1 and C_2 producing B_1 and B_2. The two W operators construct clauses A, C_1 and C_2

[3] By validity in this context we mean truth in the intended model, not the usual meaning ascribed to the term validity. We shall continue with this use of the term throughout this paper.

from the clauses B_1 and B_2. If the literal resolved on, L, is negative, the operator is called *intra-construction* and if it is positive, it is called *inter-construction*. It is interesting to note that since the literal L in A is resolved away, then clauses A, C_1 and C_2 may contain a literal whose predicate symbol does not occur in B_1 or B_2.

Again, the oracle is consulted as to the validity of the constructed clauses and is also asked to name any newly introduced predicates if they are considered plausible (in the intended model).

Truncation The *truncation* operator actually results from the special case where the base of a V or W contains the empty clause. In fact, Muggleton and Buntine [3] show that the truncation of two literals may be constructed by taking their least-general-generalisation. Rouveirol and Puget [4] introduce a more general truncation operator which replaces terms by variables and drops literals from clauses. Again, the oracle is consulted as to the validity of the truncated clause.

3 Performing Inverse Resolution Using Conceptual Graphs

3.1 Resolution Using Conceptual Graphs

Many reasoning systems using propositional or predicate calculus as their representation language employ resolution in order to perform deductive inference. In conceptual graph theory, deduction is performed through the use of alpha rules (propositional case) and beta rules (first-order case) [5]. Sowa [5] notes that the following conceptual graph schema is analogous to resolution ($\neg[u\ v], \neg[\neg[u]\ w] \vdash_{\alpha\ or\ \beta\ rules} \neg[v\ w]$). A more general form of this schema, more useful in this exposition, may be represented by the following diagram[4] (see section 2.2 on resolution):

$\neg[\neg[u] w_1 ... w_n]$ $\qquad\qquad$ $\neg[u v_1 ... v_m]$

$$\neg[v_1 ... v_m w_1 ... w_n]$$

Sowa also points out that graph matching in conceptual graphs is the analogue of unification in first-order reasoning. In the following sections we use these two facts to perform inverse resolution using conceptual graphs.

3.2 Representing Formulae Using Conceptual Graphs

Current inverse resolution systems only consider theories and examples presented to them in the form of Horn clauses. Therefore, in this paper we only consider examples where the conceptual graphs represent Horn clauses[5] though the tech-

[4] For a proof of this and the previous schema, see appendix A.

[5] That is, conceptual graphs that map to Horn clauses under the ϕ operator.

niques involved should be applicable to more general graphs. A Horn clause represented as a conceptual graph assumes the following form

$$\neg[\neg[u]\ w_1 ... w_n]$$

where $\neg[u]$ represents the head and u is either a single concept or a relation connecting concepts with coreference links to concepts in the outer contexts and $w_1...w_n$ represents the tail. Due to this relationship, we shall refer to graphs of the form $\neg[u]$ (nested at level 1) as the head of the graph and graphs of the form $w_1...w_n$ (nested at level 1) as the tail of the graph. An example of such a graph is presented in figure 1.

We notice in the resolution schema presented in the previous section that a nested graph $\neg[u]$ in one graph resolves (matches) a nested graph u in another graph and is not present in the final graph. If u only matches a subgraph of another graph then this process cannot proceed. This notion is also very important when performing inverse resolution using conceptual graphs — we rely on matching graphs contained in the nested contexts of other graphs to construct a new graph. In so doing, we wish to maximise the number of graphs that can be matched and we shall be hindered if a graph only matches a subgraph of another graph. In fact, more work may be needed in this area to determine the significance of subgraph matching.

In order to maximise the amount of matching we introduce the following "splitting" operation:
coreferent split — any concept that is linked to n (> 1) relations may be "split" into a number of identical concepts ($\leq n$) and joined by coreference links. Any relations that were originally linked to any of these concepts are then linked to only one of the "split" concepts.

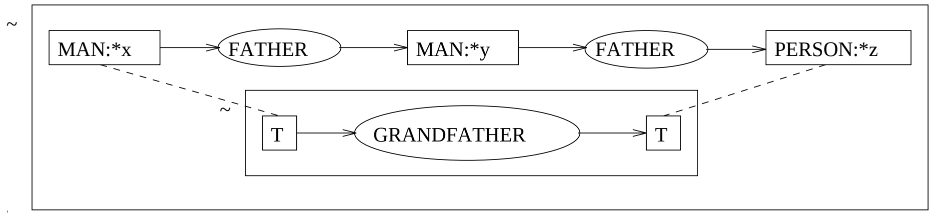


Fig. 1. Conceptual graph of the formula $\forall x \forall y \forall z((man(x) \wedge man(y) \wedge person(z) \wedge father(x, y) \wedge father(y, z)) \rightarrow grandfather(x, z))$ expressing a notion of grandfatherhood.

Figure 2 shows the graph in figure 1 after coreference splitting.

At first sight, this operation may seem strange but it actually corresponds to the inverse of the beta rule *coreferent join* and, as mentioned above, it will help us maximise the number of graphs matched, resulting in smaller graphs produced by the inverse resolution process. Recall the separability assumption

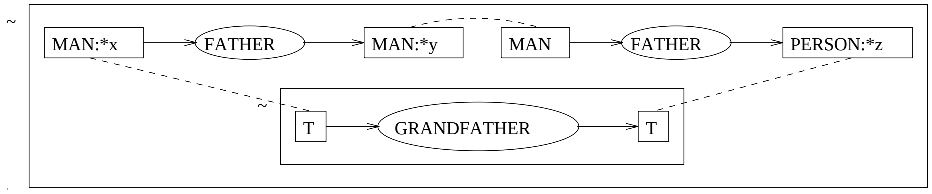


Fig. 2. Result of coreference split operation applied to conceptual graph in figure 1.

discussed in section 2.3. We also need to make this assumption here to ensure that a unique conceptual graph results from a V operation. This manifests itself as an assumption that none of the graphs $v_1 ... v_m$ in $\neg[u v_1 ... v_m]$ match any of the graphs $w_1 ... w_n$ in $\neg[\neg[u] w_1 ... w_n]$.

We are now in a position to introduce the inverse resolution operators for conceptual graphs. Together with the graph in figure 2, the graphs in figures 3 and 4 will represent the *original domain theory* while the graph in figure 5 will represent the *example* in our discussion of inverse resolution operators using conceptual graphs[6]. The graph in figure 2 represents one notion of grandfatherhood — x is the grandfather of z if x is the father of a man y and y is the father of z. Figure 3 represents a notion of fatherhood — x is the father of y if y is the child of x and x is male — and that in figure 4, a notion of motherhood — x is the mother of y if y is the child of x and x is female. The graph in figure 5 represents a particular instance of a family relationship — Tom is Liz's grandfather if Tom is the father of $Helen$, who is female, and Liz is the child of $Helen$.

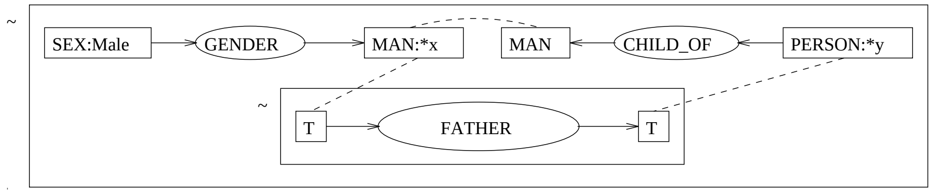


Fig. 3. Conceptual graph expressing notion of fatherhood: $\forall x \forall y((man(x) \wedge person(y) \wedge child\_of(y,x) \wedge gender(Male,x) \wedge sex(Male)) \rightarrow father(x,y))$.

3.3 V Operators[7]

[6] These examples are based on an example presented in [4].

[7] Note that in the following sections we do not present similar constructions to those in [3] since a simplifying assumption, that one of the arms of a V or W contains

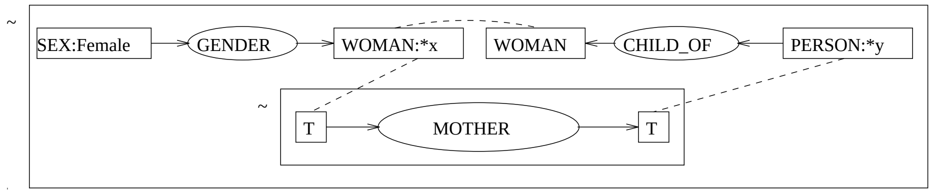


Fig. 4. Conceptual graph expressing notion of motherhood: $\forall x \forall y((woman(x) \wedge person(y) \wedge child\_of(y, x) \wedge gender(Female, x) \wedge sex(Female)) \rightarrow mother(x, y))$.

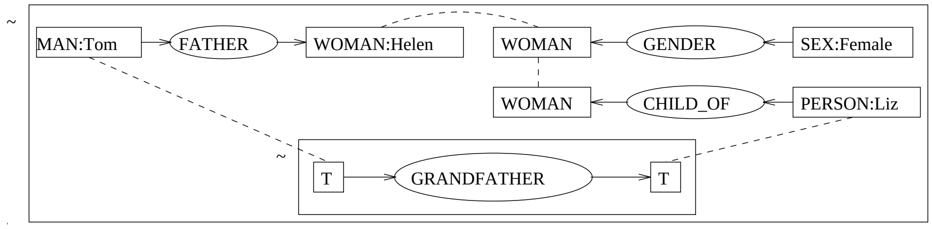


Fig. 5. Example graph: $(man(Tom) \wedge woman(Helen) \wedge person(Liz) \wedge father(Tom, Helen) \wedge child\_of(Liz, Helen) \wedge sex(Female) \wedge gender(Female, Helen)) \rightarrow grandfather(Tom, Liz)$.

In discussing the V operators we refer back to the resolution V schema for conceptual graphs presented above. Recall also that the absorption operator constructs a clause containing the negative form of the common literal. In our case, the nested graph $\neg[u]$ in the graph on the left arm of the V represents the positive form of the literal to be resolved on while the nested graph u in the right arm of the V represents the negative form. Therefore, the absorption operator using conceptual graphs will construct a graph of the form $\neg[u\ v_1 ... v_m]$ from graphs $\neg[\neg[u]\ w_1 ... w_n]$ (from the partial domain theory) and $\neg[w_1 ... w_n\ v_1 ... v_m]$ (example).

By just studying this schema we can suggest the following mechanism for performing absorption using conceptual graphs. Given two graphs G_1 (the absorbed graph — $\neg[w_1 ... w_n\ v_1 ... v_m]$ above) and G_2 (the absorbent graph — $\neg[\neg[u]\ w_1 ... w_n]$ above), if a subset $w_1 ... w_n$ of the graphs nested at level 1 of G_1 matches the body of the graph G_2 (i.e., all graphs except that denoting the head $\neg[u]$) then we construct a new graph G (the resolved quotient) containing, at level 1 of a negated context, the remainder of the graphs nested at level 1 of the absorbed graph G_1 and the non-negated head of the absorbent graph G_2. In this case, the resulting graph is obviously of the form $\neg[u\ v_1 ... v_m]$. Note also that coreference links, in the newly created graph, are placed between any

a unit clause, is used there and we wish to deal with the more general case as, for example, in [4].

concept c in graph u and a concept c' in $v_1...v_m$ if there was a coreference link between c' and a concept c'' in $w_1...w_n$ in graph $\neg[w_1...w_n\, v_1...v_m]$ and there was a coreference link between c and c'' in $\neg[\neg[u]\, w_1...w_n]$. A similar situation occurs with the other V and W operators. At this point, the oracle must be consulted to ensure validity of the resulting graph.

Figure 6 gives an example of this absorption operation. This new graph suggests an instance of the grandfatherhood notion — *Tom* is *Liz*'s grandfather if *Tom* is the father of a woman, *Helen*, who is the mother of *Liz*. This is an instance of a case not covered by our current domain theory.

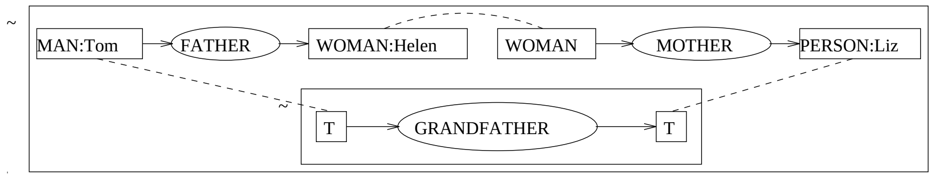


Fig. 6. Absorption of a specialisation of conceptual graph in figure 4[8] and figure 5.

This process can be justified by "inverting" the proof of $\neg[w_1...w_n v_1...v_m]$ (see appendix A). The proof of $\neg[w_1...w_n v_1...v_m]$ can be performed through the use of the beta (or alpha) rules *iteration*, *deiteration*, *erasure* and *double negation*. Using the graph $\neg[\neg[u]\, w_1...w_n]$ as a guide we can "invert" this proof by first inverting the *double negation* step to obtain $\neg[\neg[\neg[v_1...v_m]]w_1...w_n]$. We then proceed by inserting the graph u at level 2 (an even context — therefore an inverse of *erasure*) obtaining $\neg[\neg[u\neg[v_1...v_m]]w_1...w_n]$; *iteration* of u into level 3 (an odd context — the inverse of *deiteration*) obtaining $\neg[\neg[u\neg[u v_1...v_m]]w_1...w_n]$, and; finally, using $\neg[\neg[u]\ w_1...w_n]$ again as a guide, we delete the graph $\neg[u\ v_1...v_m]$ from level 2 (an even context) and assert it as a separate graph (inverse of *iteration* but note that it is not a *deiteration*). The interested reader is also referred to appendix B for a stepwise presentation of this justification.

An interesting (and important) point to note is that the process we have described is directly analogous to those proposed for absorption in [1] and [4] for the propositional and predicate calculus respectively. In these, if the body of a clause C_2 can be unified with part of the body of a clause C_1 then a new clause is constructed (from C_1) by replacing that part of the body of C_1 that unifies with the body of C_2 with the head of C_2.

A process for performing identification can be derived in the same way. In this case, a graph $\neg[\neg[u]\ w_1...w_n]$ is constructed from graphs $\neg[w_1...w_n\ v_1...v_m]$ and $\neg[u\, v_1...v_m]$. The justification of this process is very similar to that for absorption

[8] The graph in figure 4 was specialised by restricting the referent of the concept `[PERSON: *y]` to `[PERSON: Liz]`. Alternatively, we could have attempted a generalisation of the graph in figure 5 but we have tried to avoid generalising too soon here.

except that, in the final step, the inverse of the *iteration* is actually a *deiteration* (see appendix B). Again, the process involved resembles that proposed in [1] for identification.

3.4 Truncation

Rouveirol and Puget [4] defined a truncation operator that replaces terms by variables and drops literals in the tail of a clause. Based on this definition the truncation operator on conceptual graphs is capable of performing the following actions:

- remove graphs in an odd context (level 1 in the cases considered here)
- replace individual referents by generic referents
- replace types by supertypes[9].

The last of these could actually have been suggested as a separate generalisation operation to aid in maximising the number of graphs matched but it makes more sense to include it here because, together, these three operations are exactly the inverse of the alpha and beta rule *insertion*. In fact, this last operation, appears to be performing exactly the same task that inverse substitution performs in the first-order logic case. Figure 7 shows a truncation of the graph in figure 6. This represents a simple generalisation of the graph in figure 6 where terms have been replaced by variables and one concept type label (`WOMAN`) has been replaced by its supertype (`PERSON`). This gives another general graph for the notion of grandfatherhood — x is the grandfather of z if x is the father of a person y and y is the mother of z.

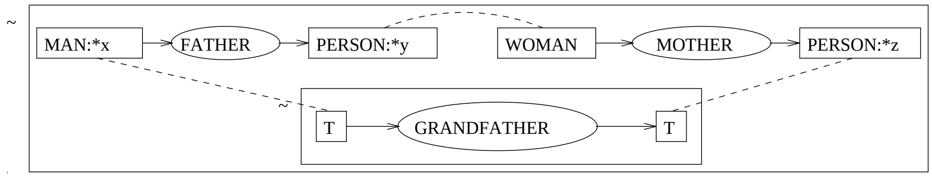


Fig. 7. Result of several truncation operations applied to conceptual graph in figure 6.

Again, the oracle must be consulted to determine whether the resulting graph is meaningful in the current situation.

3.5 W Operators

As mentioned previously, the two W operators are potentially the most interesting of all the inverse resolution operators because they are capable of introducing

[9] In fact, these latter two generalisations are considered generalisations by Sowa [5].

new predicate symbols into the vocabulary. In performing these operations using conceptual graphs this manifests itself as the introduction of new relations, not present in the original graphs, between concepts.

Recall that in intra-construction the common literal resolved on occurs negated in the clause at the middle vertex of the W. This fact motivates the following schema for performing intra-construction using conceptual graphs.

$$\neg[\neg[v]u_1 ... u_m] \qquad \neg[v w_1 ... w_n] \qquad \neg[\neg[v]s_1 ... s_l]$$

$$\neg[u_1 ... u_m w_1 ... w_n] \qquad \neg[s_1 ... s_l w_1 ... w_n]$$

This leads us to the following technique for intra-construction using conceptual graphs. Given two graphs G_1 (here, the graph $\neg[u_1 ... u_m w_1 ... w_n]$) and G_2 (here, the graph $\neg[s_1 ... s_l w_1 ... w_n]$) such that a subset $w_1 ... w_n$ of the graphs nested at level 1 of G_1 match a subset of the graphs nested at level 1 of G_2 we construct three new graphs in the following way. The first graph ($\neg[\neg[v]u_1 ... u_m]$) is a negated context containing (as its body) graphs $u_1 ... u_m$ from level 1 of graph G_1 that did not feature in the match with G_2 and (as its head) a new graph v (in a negated context at level 1) consisting of a new relation not present in the original graphs. The new relation has links to as many concepts as appear in the outer context (i.e., in graphs $u_1 ... u_m$) and coreference links between each of these concepts and a concept in the outer context (level 1)[10]. A second graph ($\neg[\neg[v]s_1 ... s_l]$) can be constructed in a similar way except that the body contains graphs at level 1 in G_2 that did not match those in G_1 (i.e., graphs $s_1 ... s_l$ above). The final graph is constructed by placing the matched graphs $w_1 ... w_n$ at level 1 of a negated context along with a new graph consisting of the newly introduced relation together with its concepts (graph v). In constructing this last graph, if there were any coreference links between concepts in the u_i's (or s_k's) and the w_j's in the original graphs then coreference links are also placed between concepts in the w_j and concepts in v that had coreference links to the respective u_i (or s_k) in the previous two graphs constructed. At this point, the oracle should be consulted to determine whether the new graphs and the new relation are meaningful and, if so, name the relation.

Figure 8 gives an example of an intra-construction operation using conceptual graphs. Here, the larger graph represents a single notion of grandfatherhood (replacing the two notions represented in figures 2 and 7) — x is the grandfather of z if x is the father of a person y and y is *newp* to z. The two smaller graphs

[10] In fact, since the concepts in graphs $u_1 ... u_m$ may already be linked by coreference links due to coreference split operations, the number of concepts linked to the new relation can be reduced. We only need to include one concept in v for every set of concepts in $u_1 ... u_m$ linked together by coreference links. Since the concepts in v are to be linked to other concepts via coreference links we can simply make them the general type (i.e., [$\top$]) if desired.

describe the introduced *newp* relation — x is *newp* to z if x is the father of z and x is *newp* to z if x is the mother of z. In this case the new relation corresponds to the *parent* relation (i.e., (PARENT)).

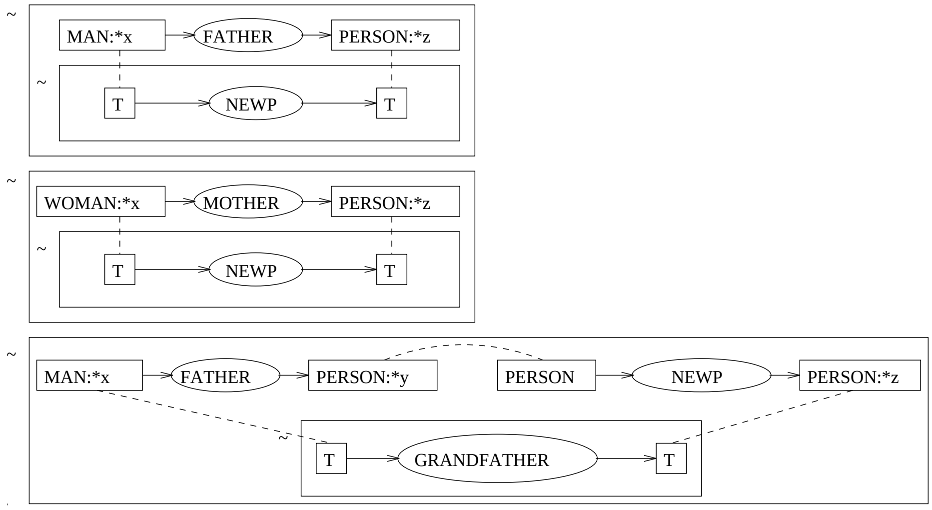


Fig. 8. Intra-construction of graph in figure 7 and truncation of graph in figure 2[11].

A justification for this process in terms of inverses of beta (alpha) rules can be derived in a similar fashion to the section on V operators (see appendix B). In this case, the two graphs at the base of the W can be used to guide "inverse" proofs of the two individual V's (see appendix B). Again, the process above is analogous to that proposed in [1] and [4] for intra-construction in propositional and first-order predicate calculus using Horn clauses.

The schema for inter-construction becomes

$$\neg[vu_1 ... u_m] \qquad \neg[\neg[v]w_1 ... w_n] \qquad \neg[vs_1 ... s_l]$$

$$\neg[u_1 ... u_m w_1 ... w_n] \qquad \neg[s_1 ... s_l w_1 ... w_n]$$

since the common literal (v) appears positive in the middle vertex of the W in this case. A process for performing inter-construction can be derived and justified in a similar way to that for intra-construction above.

An interesting point to note is that both the schema for intra-construction and inter-construction have similar graphs at the base of the W schema and,

[11] The concept `[MAN:*y]` was truncated to `[PERSON:*y]`.

therefore, it would appear impossible to determine when one operation should be performed in favour of the other. This does not occur in predicate calculus cases discussed because they deal with a more restricted case where all clauses are definite. In this situation, intra-construction corresponds to having two base graphs $\neg[\neg[t]u_1...u_m w_1...w_n]$ and $\neg[\neg[t]s_1...s_l w_1...w_n]$ having matching heads which produce a graph $\neg[\neg[t]v w_1...w_n]$ at the middle vertex of the W (the other graphs retain their current form). Inter-construction corresponds to having two base graphs $\neg[\neg[t]u_1...u_m w_1...w_n]$ and $\neg[\neg[r]s_1...s_l w_1...w_n]$ with differing heads, producing the graph $\neg[\neg[t]v u_1...u_m]$ on one arm and $\neg[\neg[r]v s_1...s_l]$ on the other arm of the W (the other graph retains its current form).

As a final note, observe that in our examples we have used graphs that have been subjected to a coreference split operation and that graphs resulting from inverse resolution operations are also "split". These graphs can of course be joined by using the beta rule *coreference join* to produce more readable graphs.

4 Conclusions and Future Work

In this paper we have shown how to perform the process of inverse resolution using conceptual graphs. This provides us with a technique for performing inductive inference using conceptual graphs. In particular, two of the operators (intra-construction and inter-construction) furnish us with a method for performing constructive induction — in this case, the introduction of new relations, between concepts, that were not present in the original graphs.

We have also seen how the mechanisms developed for performing inverse resolution using conceptual graphs are analogous to the operations for performing inverse resolution on Horn clauses. In fact, we claim that viewing these operations as graph operations is more natural and provides one with a clearer picture of the processes involved.

A very important point is that the operations developed in this paper can be viewed as inverses of particular beta (or alpha) rules or groups of beta (alpha) rules. This gives us an idea of how we can perform generalisation by inverting the proof theory of a system. In future work it would be interesting to define a notion of generality based on alpha or beta rules (i.e., with respect to $A \vdash_\alpha B$ or $A \vdash_\beta B$) and to investigate inverses of these individual operators as a method for performing (constructive) induction using conceptual graphs. We suspect that, in order to reduce the computational complexity of a such a system and the number of accesses to the oracle, more domain information will need to be taken into account.

Thus far, applications using inverse resolution appear to been restricted to classification problems (see, for example, [1] [2]). Recall that the main aim in applying inverse resolution operators is to refine a particular domain theory. Therefore, we can imagine that the operators developed here may be particularly well suited to evolving domain theories represented in terms of conceptual graphs. A particular example would be the development of a rule base for a conceptual graph expert system. Inverse resolution could also be used as a technique in scientific theory formation where the domain theory represents the current state of scientific knowledge and the oracle takes on the role of verifying experiments. Again, in our case conceptual graphs would be used to represent scientific knowledge. The advantage in using inverse resolution over other learning techniques is inverse resolution's ability to perform constructive induction or, in our case, introduce new relations not previously present.

5 Acknowledgements

The authors would especially like to thank Peter Eklund and the anonymous referees whose comments have proved invaluable in producing the final version of this paper. We would also like to thank our colleagues in the Knowledge Systems Group whose comments have improved the content and presentation in this paper. The first author is supported by an Australian Postgraduate Research Award.

References

1. Stephen Muggleton. Duce, an oracle based approach to constructive induction. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 287–292. Morgan Kaufman, 1987.
2. Stephen Muggleton. Inverting the resolution process. In J. E. Hayes and D. Michie, editors, *Machine intelligence 12*. 1989.
3. Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Machine Learning Workshop*, pages 339–352. Morgan Kaufman, 1988.
4. Celine Rouveirol and Jean Francois Puget. Beyond inversion of resolution. In *Proceedings of the 7th International Machine Learning Workshop*, pages 122–130. Morgan Kaufman, 1990.
5. John F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, Reading, MA, 1984.

A Resolution Using Conceptual Graphs

In this section we present a proof of the following sequent:
$\neg[\neg[u]w], \neg[uv] \vdash_{\alpha rules} \neg[vw]$

1	$\neg[uv]$	HYP
2	$\neg[\neg[u]w]$	HYP
3	$\neg[\neg[u\neg[uv]]w]$	1,2 iteration
4	$\neg[\neg[u\neg[v]]w]$	3 deiteration (u)
5	$\neg[\neg[\neg[v]]w]$	4 erasure (u)
6	$\neg[vw]$	5 double negation

This of course can be generalised to a proof of the following sequent which is of more use in this presentation
$\neg[\neg[u]w_1 ... w_n], \neg[uv_1 ... v_m] \vdash_{\alpha rules} \neg[v_1 ... v_m w_1 ... w_n]$

1	$\neg[uv_1...v_m]$	HYP
2	$\neg[\neg[u]w_1...w_n]$	HYP
3	$\neg[\neg[u\neg[uv_1...v_m]]w_1...w_n]$	1,2 iteration
4	$\neg[\neg[u\neg[v_1...v_m]]w_1...w_n]$	3 deiteration (u)
5	$\neg[\neg[\neg[v_1...v_m]]w_1...w_n]$	4 erasure (u)
6	$\neg[v_1...v_m w_1...w_n]$	5 double negation

B Justification of Conceptual Graph Inverse Resolution Operators

In this section we attempt to justify the inverse resolution operations for conceptual graphs by showing how they can be considered an inverse of a series of alpha and beta rules.

We can justify the absorption process using the following "inverse proof".

1	$\neg[\neg[u]w_1...w_n]$	Domain Theory
2	$\neg[v_1...v_m w_1...w_n]$	Example
3	$\neg[\neg[\neg[v_1...v_m]]w_1...w_n]$	2 double negation (double negation^{-1})
4	$\neg[\neg[u\neg[v_1...v_m]]w_1...w_n]$	3 insert u at L2 (erasure^{-1})
5	$\neg[\neg[u\neg[uv_1...v_m]]w_1...w_n]$	4 iteration (u) (deiteration^{-1})
6	$\neg[uv_1...v_m]$	1, 5 deletion + assert (iteration^{-1})

We can justify identification in a similar way:

1	$\neg[uv_1...v_m]$	Domain Theory
2	$\neg[v_1...v_m w_1...w_n]$	Example
3	$\neg[\neg[\neg[v_1...v_m]]w_1...w_n]$	2 double negation (double negation^{-1})
4	$\neg[\neg[u\neg[v_1...v_m]]w_1...w_n]$	3 insert u at L2 (erasure^{-1})
5	$\neg[\neg[u\neg[uv_1...v_m]]w_1...w_n]$	4 iteration (u) (deiteration^{-1})
6	$\neg[\neg[u]w_1...w_n]$	1, 5 deiteration (iteration^{-1})

Justifications for intra-construction and inter-construction are slightly more complex but follow the same idea. For intra-construction we have the following justification:

1	$\neg[w_1...w_n u_1...u_m]$	Domain Theory
2	$\neg[w_1...w_n s_1...s_l]$	Domain Theory
3	$\neg[\neg[\neg[w_1...w_n]]u_1...u_m]$	1 double negation (double negation^{-1})
4	$\neg[\neg[\neg[w_1...w_n]]s_1...s_l]$	2 double negation (double negation^{-1})
5	$\neg[\neg[v\neg[w_1...w_n]]u_1...u_m]$	3 insert v at L2 (erasure^{-1})
6	$\neg[\neg[v\neg[w_1...w_n]]s_1...s_l]$	4 insert v at L2 (erasure^{-1})
7	$\neg[\neg[v\neg[vw_1...w_n]]u_1...u_m]$	5 iteration (v) (deiteration^{-1})
8	$\neg[\neg[v\neg[vw_1...w_n]]s_1...s_l]$	6 iteration (v) (deiteration^{-1})
9	$\neg[vw_1...w_n]$	7, 8 delete + assert (iteration^{-1})
10	$\neg[\neg[v]u_1...u_m]$	7, 9 deiteration (iteration^{-1})
11	$\neg[\neg[v]s_1...s_l]$	8, 9 deiteration (iteration^{-1})

Inter-construction can be justified in similar fashion:

1	$\neg[u_1...u_m w_1...w_n]$	Domain Theory
2	$\neg[s_1...s_l w_1...w_n]$	Domain Theory
3	$\neg[\neg[\neg[u_1...u_m]]w_1...w_n]$	1 double negation (double negation^{-1})
4	$\neg[\neg[\neg[s_1...s_l]]w_1...w_n]$	2 double negation (double negation^{-1})
5	$\neg[\neg[v\neg[u_1...u_m]]w_1...w_n]$	3 insert v at L2 (erasure^{-1})
6	$\neg[\neg[v\neg[s_1...s_l]]w_1...w_n]$	4 insert v at L2 (erasure^{-1})
7	$\neg[\neg[v\neg[vu_1...u_m]]w_1...w_n]$	5 iteration (v) (deiteration^{-1})
8	$\neg[\neg[v\neg[vs_1...s_l]]w_1...w_n]$	6 iteration (v) (deiteration^{-1})
9	$\neg[vu_1...u_m]$	7, 8 delete + assert (iteration^{-1})
10	$\neg[vs_1...s_l]$	7, 8 delete + assert (iteration^{-1})
11	$\neg[\neg[v]w_1...w_n]$	7, 9 deiteration (iteration^{-1})