Rule Based Computation of Updates to Terminologies

S.Modgil
Biomedical Informatics Unit,
Eastman Institute for Oral Health Care Sciences,
University College London, England
Email: S.Modgil@eastman.ucl.ac.uk

Abstract

In this paper we formalise compilation of the conjunctive bodies of a restricted class of Horn rules into updates on terminologies. This involves a pre-processing of the graphs representing connections between terms in a rule body. We motivate and illustrate application of this work in hybrid Description Logic/Rule based frameworks for medical plan specification.

1 Introduction and Motivation

A number of works (e.g., [5, 3]) have investigated reasoning in mixed rule based and Description Logic knowledge bases. In this paper we describe a novel treatment of the interaction between Horn rules and Description Logic terminologies; viz.a.vie., we formalise compilation of the conjunctive bodies of a restricted class of Horn rules into updates on terminologies, so as to exploit the ability of Description Logics to maintain incomplete information about individuals. We motivate application and benefits of this work in the context of incremental design tasks in which: a) rule based reasoning over a partially specified design is used to suggest addition of design components; b) there is a requirement to represent and reason with incomplete specifications of these design components; c) domain knowledge is encoded in a Description Logic terminology.

To illustrate, consider the incremental task of designing (authoring) a medical plan. In [9] we describe the development and use of a medical plan authoring tool. Use of the tool results in generation of both a symbolic representation of the plan for export to plan execution software, and a textual description of the plan for inclusion in a protocol document. Linked to the authoring tool is a "plan advisor" [8] which contains rules that can be queried to suggest updates to a plan being designed, e.g.,

$$\textit{safety(Action, E1, Prevent\_Act)} \leftarrow \textit{plan(Action), effect(Action,E1)}, \textit{hazard(E1),action(Prevent\_Act),effect(Prevent\_Act,E2),prevent(E2,E1)} \quad \textbf{(1)}$$

Suppose a partially designed medical plan containing the *Action drug_A* which has the hazardous effect $E1 = dehydration$, and the query *? safety(Action, E1, Prevent_Act).* Successful evaluation of the query will bind *Prevent_Act* to an action that has an effect $E2$ which prevents *dehydration*, prompting the plan author to update the symbolic plan representation with this preventative action. However, the user can interact with the plan advisor to *partially* evaluate (unfold) a natural language translation of (1) to:

safety(drug_A,dehydration,Prevent_Act)$\leftarrow$ *action(Prevent_Act),effect(Prevent_Act,E2),*
prevent(E2,dehydration) **(2)**

The body of (2) is then used to generate the text *"execute an action that has an effect which prevents dehydration"*, which is added to (updates) the textual protocol describing the plan. This text represents an incompletely specified new action described in terms of its *intentions* or *goals*. The ability to model planned actions in terms of their intentions has a number of important benefits [11]. In particular it allows for flexibility in the detailed specification of an action at plan execution time, at which point the specifics of a suggested action can be checked for compliance with the intentions (one can think of intentions as constraints). However, (2) does not constitute a corresponding declarative representation of the incompletely described action, for inclusion in the symbolic representation of the plan. Now, assuming a Description Logic encoding of the medical domain knowledge (of which there are an increasing number of examples e.g., [10]), then the body of (2) can be compiled into a new concept definition; $prevent\_dehydration\_act \doteq action \sqcap \exists effect.prevent : dehydration$, and a new fact, $prevent\_dehydration\_act(a2)$, can be asserted. The intentions of the "place-holder" action $a2$ are encoded in the description, so that at plan execution time, classification services can check for compliance of a suggested specific action (checking whether it has the intended effect). Furthermore, at plan design time, the action $a2$ can be reasoned with as part of the symbolic plan (e.g., ordered temporally) in the same way as other concrete actions (e.g., $drug\_A$). Also, properties of $a2$ can be reasoned about, via hybrid reasoning of the type described in [5] (e.g., $safety(drug\_A, dehydration, a2)$ will be a hybrid entailment).

In the remainder of this paper we formalise the above update procedure in the context of mixed Horn Rule and Description Logic knowledge bases. This requires defining compilation of the conjunctive bodies of a restricted class of Horn rules to Description Logic concepts. Compilation involves an initial pre-processing of the graphs representing connections between terms in a rule body. Note that the central role of conjunctive queries in database and knowledge representation has motivated previous work on compilation of conjunctive *queries* into concept descriptions [2, 4]. The class of conjunctions considered in this work is wider than those considered in [2], but close to those considered in [4]. Our approach necessarily differs from the latter given that we update, rather than query, terminologies.

2 Preliminaries

In this work, a knowledge base (kb) is a tuple $(\mathsf{R}, \mathsf{T}, \mathsf{A})$, where R is a set of non-recursive Horn rules $H(\overline{Y}) \leftarrow B_1(\overline{X}_1), \ldots, B_n(\overline{X}_n)$, where $\overline{Y}, \overline{X}_1, \ldots, \overline{X}_n$ are tuples of variables or constants, and the rule is *safe*, i.e., any variable in $\overline{Y}$ must also appear in $\bigcup_{i=1}^{n} \overline{X}_i$. A is a set of ground facts, and T is an acyclic terminology in a language that is a superset of $\mathcal{ERIB}$ (concept conjunction; role conjunction; existential quantification; fills $(R : a)$; inverse roles (R^{-1})). Our primary concern is to show how the body of a rule in R can be compiled into a $\mathcal{ERIB}$ concept definition. However, this work should be viewed in the context of hybrid reasoning on mixed knowledge bases. In [5], hybrid reasoning is described for a CARIN kb $\triangle = (\mathsf{R}, \mathsf{T}, \mathsf{A})$, where predicates in a rule body can be concept or role names, or *ordinary* predicates of arbitrary *arity* that do not

appear in T. T is encoded in some subset of $\mathcal{ALCNR}$ (no inverse roles or fills). A sound and complete decidable reasoning procedure is defined for determining whether $\triangle \models q(\overline{a})$, where q is a concept, role or ordinary predicate, and $\overline{a}$ a tuple of constants. A key feature of this procedure is the existential entailment algorithm (based on constraint systems). Propagation rules are applied to generate completions of the models of T and the ground concept and role atoms in A. Extensions of the ordinary predicates are then computed by evaluating the Horn rules using a traditional Horn rule reasoning algorithm. The CARIN authors have indicated (in private communications) that their reasoning procedure can be extended (straightforwardly) for fills, and (with difficulty) for inverse roles.

The following definitions establish the notion of a computable rule, i.e., a rule whose body can be compiled into a $\mathcal{ERIB}$ expression. The concluding lemma will be of use in the following section.

Definition 1 The *binding graph* $\mathcal{G}(B)$, of a Horn rule $H \leftarrow B$, is a set of labelled edges $(\alpha, \beta, \{r_1, \ldots, r_n\})$, where $(\alpha, \beta, \{r_1, \ldots, r_n\}) \in \mathcal{G}(B)$ if α and/or β is a variable in B, and for $i = 1 \ldots n, r_i(\alpha, \beta)$ is a predicate in B.

We say that $predecessor((\alpha, \beta, R)) = \alpha$ and $successor((\alpha, \beta, R)) = \beta$. A *node* η is in a set of edges Σ, if η is the predecessor or successor of an edge $e \in \Sigma$. Also, an edge $e = (\alpha,\beta,R)$ can be reversed to $e^{-1} = (\beta,\alpha,R^{-1})$, where $R^{-1} = \{r^{-1} | r \in R\}$. Note that if $\Sigma = \{e_1, \ldots, e_n\}$ then $\Sigma^{-1} = \{e_1^{-1}, \ldots, e_n^{-1}\}$.

Definition 2 Let α and β be two nodes in a set of edges $\Sigma = \{e_1, \ldots, e_n\}$. Then,
• connected(α, β) if **a)** $(\alpha, \beta, R) \in \Sigma$ or $(\beta, \alpha, R) \in \Sigma$, or **b)** $(\alpha, \gamma, R) \in \Sigma$ or $(\gamma, \alpha, R) \in \Sigma$, and connected$(\gamma, \beta)$
• Σ is a route from α to β iff $predeccessor(e_1) = \alpha$, $successor(e_n) = \beta$, and for $i = 1 \ldots n-1$, $successor(e_i) = predecessor(e_{i+1})$. Σ is a **cyclical** route if $\alpha = \beta$.
• $\Sigma*$ denotes a set obtained by reversing some subset Σ' of edges in Σ, i.e., $\Sigma* = (\Sigma - \Sigma') \cup \Sigma'^{-1}$
• A set Σ' of edges is **distinct** from Σ if $\neg\exists\ \Sigma'*$ such that $\Sigma'* = \Sigma$

Definition 3 Let G be a binding graph.
• Σ is a subgraph of G if $\Sigma \subseteq G$, $\forall \eta, \eta' (\eta \neq \eta')$ in Σ, connected(η, η'), and Σ is maximal under set inclusion. Σ is a **variable subgraph** of G if Σ is a subgraph of G', where G' is the set of edges in G with variable predecessors and successors
• Σ is a cycle in G if $\Sigma \subseteq G$ and there exists a cyclical route $\Sigma*$

Definition 4 A Horn rule $H \leftarrow B$ and the binding graph $\mathcal{G}(B)$ are said to be **computable**, iff
- $n \leq 2$ for any non-ground n-arc predicate in the body B[1].
- If $\{\Sigma_1, \ldots, \Sigma_n\}$ is the set of cycles in $\mathcal{G}(B)$, then for $i = 1 \ldots n$, there exists a constant node in Σ_i.

Lemma 1 Let Σ and Σ' be two sets of edges such that there exists two distinct routes, $\Sigma*$ from β to δ, and $\Sigma'*$ from β to δ. Then $\Sigma \cup \Sigma'$ contains a cycle

[1]Excluding non-ground predicates of arity > 2 is not as restrictive as might first appear, since any n-arc predicate can be re-expressed as a conjunction of binary predicates ([1])

PROOF. Since $\Sigma*$ and $\Sigma'*$ are distinct, then $\exists e \in \Sigma*$ ($\Sigma'*$) such that neither e nor e^{-1} are in $\Sigma'*$ ($\Sigma*$). It is sufficient to prove there is a cycle in the case that $\Sigma*$ and $\Sigma'*$ differ by only one edge (and its reverse). Let $\Sigma* = \{e_1, \ldots, e_{k-1}, (\alpha, \gamma, R), e_{k+1}, \ldots, e_n\}$ and $\Sigma'* = \{e_1, \ldots, e_{k-1}, (\alpha, \gamma, R'), e_{k+1}, \ldots, e_n\}$ (i.e., two distinct edges from nodes α to γ). Hence there exists a $\Upsilon \subseteq \Sigma \cup \Sigma'$ such that there is a cyclical route $\Upsilon* = \{(\alpha, \gamma, R), (\gamma, \alpha, R'^{-1})\}$, i.e., $\Sigma \cup \Sigma'$ contains a cycle. QED

3 Compiling Updates from Rule Bodies

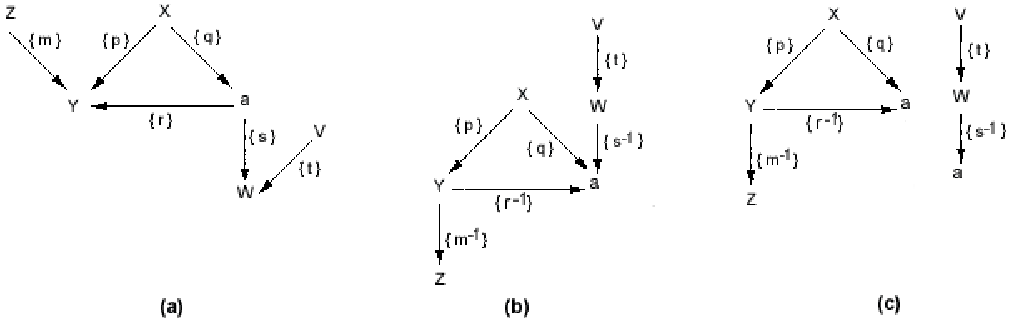

Figure 1: a) Binding graph for body of rule (3); (b) binding graph *transformed* by definition 6; c) partitioned graphs defined by definition 7

Fig.1a) shows the binding graph for the example rule:

$$h(X) \leftarrow i(X), m(Z,Y), p(X,Y), q(X,a), r(a,Y), s(a,W), t(V,W) \quad \textbf{(3)}$$

A binding graph must be processed before compilation of updates. This initially involves transforming the binding graph so that no variable is the successor of two edges. Firstly, we define a procedure for transforming a set of edges to a tree (the term *root node* describes a node that is not the successor of any other node).

Definition 5 Let Σ be a set of edges and α a root node in Σ. Then $trans(\Sigma) = subtrans(\alpha, \emptyset)$, where for any node η in Σ, $subtrans(\eta, \mathbf{P})$ is defined as follows: (read "**P**" as "processed nodes"):

Let $\{(\eta, \beta_1, R_1), \ldots, (\eta, \beta_n, R_n)\}$ be the set of all edges in Σ with predecessor η. Let $\{(\gamma_1, \eta, R'_1), \ldots, (\gamma_l, \eta, R'_l)\}$ be the set of all edges in Σ with successor η, such that for $k = 1 \ldots l$ $\gamma_k \notin \mathbf{P}$. Then,

$subtrans(\eta, \mathbf{P}) = \bigcup_{j=1}^{n}\{\ \{(\eta, \beta_j, R_j)\} \cup subtrans(\beta_j, \mathbf{P} \cup \{\eta\})\ \} \cup$
$\bigcup_{k=1}^{l}\{\ \{(\eta, \gamma_k, R_k'^{-1})\} \cup subtrans(\gamma_k, \mathbf{P} \cup \{\eta\})\ \}$

Note, if Σ is a variable subgraph of a computable graph, then Σ contains no cycles and so at least one root node. We now show that the procedure $trans(\Sigma)$ terminates and generates a tree, and subsequently define the first step in the processing of a binding graph.

Theorem 2 *If G is a computable binding graph and Σ a variable subgraph of G, then $trans(\Sigma)$ terminates and generates a tree.*

PROOF. Let $trans(\Sigma) = subtrans(\rho, \emptyset)$. By def.5, at any stage in the computation, the (partially) computed $trans(\Sigma)$, is the union of sets of edges $T_1, \ldots, T_n$ where each T_i is of the form $\Omega* \cup subtrans(\delta, \mathbf{P})$, where $\Omega \subseteq \Sigma$. By def.5:

$$\Omega* \text{ is a route from } \rho \text{ to } \delta, \text{ and } \forall\ \eta, \eta \neq \delta,\ \eta \text{ is in } \Omega* \text{ if } \eta \in \mathbf{P}. \qquad \mathbf{2.1}$$

Given that Σ is finite, proof of termination is shown if $\delta \neq \delta'$ for any $T = \Omega* \cup subtrans(\delta, \mathbf{P})$, $T' = \Omega'* \cup subtrans(\delta', \mathbf{P}')$. Suppose $\delta = \delta'$. Then there exist two distinct computed routes $\Omega*$ from ρ to δ, and $\Omega'*$ from ρ to δ. By lemma1, $\Omega \cup \Omega'$ contains a cycle. Hence, a subset of Σ contains a cycle, contradicting the assumption that G is computable. To show $trans(\Sigma)$ generates a tree, we show that for any $T = \Omega* \cup subtrans(\alpha, \mathbf{P})$, if $(\gamma_1, \alpha, R'_1), \ldots, (\gamma_l, \alpha, R'_l)$ are edges with successor α, then there exists at most one $\gamma_k \in \mathbf{P}$, i.e., all but one of the edges with successor α will be reversed. Suppose $\gamma_j \in \mathbf{P}$ and $\gamma_k \in \mathbf{P}$. Then by 2.1, $\Omega* = \{e_1, \ldots, e_n, (\gamma_k, \alpha, R'_k)\}$ where γ_j is a node in $\{e_1, \ldots, e_n\}$. Hence there is a route from γ_j to α that is a subset of $\Omega*$. By assumption, there is a distinct route $\{(\gamma_j, \alpha, R'_j)\}$, and so by the same reasoning as above, some subset of Σ contains a cycle, contradicting the assumption that G is computable. QED

Definition 6 Let G be a computable binding graph and $\{\Sigma_1, \ldots, \Sigma_n\}$ the set of all variable subgraphs of G. Let $H = \bigcup_{i=1}^{n} \Sigma_i$ and $H' = \bigcup_{i=1}^{n} trans(\Sigma_i)$. Let $I = \{(\alpha, \beta, R) | (\alpha, \beta, R) \in G$, and α is a constant$\}$. Then,

$$transform(G) = (G - H - I) \cup (H' \cup I^{-1})$$

As a result of applying *transform* to a binding graph, no variable is the successor of two or more edges, and no constant is the predecessor of an edge. Fig.1b) shows the results of applying *transform* to the binding graph for (3). The second step involves partitioning each subgraph in transform(G) into a set of graphs (fig.1c). Each partitioned graph has exactly one root node, and no two graphs share a variable node. We give a declarative definition of the partitioned graphs ($\mathcal{PW}$ denotes "powerset"), then give a final definition for processing a binding graph, and subsequently prove properties of the graphs obtained which will be referenced in the theorem concluding this section.

Definition 7 For a set of edges Σ, let τ be a set $\{\Sigma'_1, \ldots, \Sigma'_k\}$, such that:
1) $\tau \subseteq \mathcal{PW}(\Sigma)$,
2) $\Sigma = \bigcup_{i=1}^{k} \Sigma'_i$, and $\bigcap_{i=1}^{k} \Sigma'_i = \emptyset$,
3) each Σ'_i has exactly one root node,
4) if α is a variable in Σ'_i then $\neg\exists\Sigma'_j$, such that $j \neq i$ and α is in Σ'_j

If Σ is a singleton set $\{(\alpha, \beta, R)\}$ then $partition(\Sigma) = \{\Sigma\}$, else:
$partition(\Sigma)$ is a set τ, such that τ satisfies 1)-4) above, and $\forall\tau'$ such that τ' satisfies 1) - 4), $|\tau| \leq |\tau'|$.

Definition 8 Let G be a computable binding graph, and $\{\Sigma_1, \ldots, \Sigma_n\}$ the set of all subgraphs of $transform(G)$. Then, $process(G) = \bigcup_{i=1}^{n} partition(\Sigma_i)$.

Lemma 3 Let G be a computable binding graph. Then: **i)** No variable node in $process(G)$ has more than one predecessor; **ii)** no constant in $process(G)$ is a predecessor node; **ii)** If a constant α is the successor of n edges in a subgraph $\Sigma \in process(G)$, then Σ contains n distinct routes from the root node η (in Σ) to α

PROOF. **i)** follows from theorem 2 and def.6. It is sufficient to prove **ii** for the case n = 2, i.e., $(\gamma, \alpha, R) \in \Sigma$, $(\beta, \alpha, R') \in \Sigma$, $\gamma \neq \beta$. By def.1 stating that no edge in G has a constant predecessor and successor, and by reversal of all edges with constant predecessor in def.6, γ and β must be variables, and a constant can only be a terminal node (proving **ii**). Hence, by **i** and assumption of a single root node, there must exist two distinct routes (containing only variables) from η to γ (hence α), and η to β (hence α). QED

Note that it must be possible to partition a graph Σ to obtain graphs with a single root node such that no two graphs share a variable. Firstly, it cannot be the case that Σ has no root nodes. Suppose otherwise. Then some subset Σ' of Σ must be a cyclical route $\{(\alpha, \beta_1, R), (\beta_1, \beta_2, R_1), \ldots, (\beta_n, \alpha, R_n)\}$. By def.6, no node in Σ' can be a constant, contradicting the assumption of computability. Secondly, suppose Σ has two or more root nodes, and it is not possible to partition. This will be the case if a variable node is the successor or more than one edge, since one cannot then retain all edges in the partitioned graphs (def.7-2)) while ensuring that no variable is common to any two graphs. But then this contradicts lemma 3i) (which holds true of graphs obtained by *transform* (def.8) prior to partitioning).

We now define compilation of $\mathcal{ERIB}$ expressions, and computation of updates from a rule body. In the following, if $\mathsf{r} = H \leftarrow B$, then $\forall \alpha \in \overline{B}$, let $Unary(\mathsf{r}, \alpha) = \{A \mid A(\alpha)$ is a unary predicate in $B\}$, and $Unary(\mathsf{r}, \alpha) = \top$ if there exists no unary predicate in B. Also, let $ground(B)$ denote the ground predicates in B, and $isolated(B)$ denote the non-ground predicates in B whose variables do not appear in any binary predicate (all predicates in $isolated(B)$ will be unary).

Definition 9 Let $\mathsf{r} = H \leftarrow B$, and β a node in a subgraph $\Sigma \in process(\mathcal{G}(B))$. Let

$$\{(\beta, \gamma_1, R_1), \ldots, (\beta, \gamma_l, R_l), (\beta, \gamma_{l+1}, R_{l+1}), \ldots, (\beta, \gamma_n, R_n)\}$$

be the set of all edges in Σ with predecessor β, where each R_i is a set $\{r_i^1, \ldots, r_i^k\}$, and for $i = 1 \ldots l$, γ_i is a variable, for $i = l+1 \ldots n$, γ_i is a constant. Then,

$$compile(\beta) = \bigwedge Unary(\mathsf{r}, \beta) \wedge \bigwedge_{i=1}^{l} (\exists r_i^1 \sqcap \cdots \sqcap r_i^k.\ compile(\gamma_i)) \wedge \bigwedge_{i=l+1}^{n} (r_i^1 \sqcap \cdots \sqcap r_i^k : \gamma_i)$$

Definition 10 Let $\triangle = (\mathsf{R}, \mathsf{T}, \mathsf{A})$, $\mathsf{r} = H \leftarrow B$ a computable rule, and let $\{\gamma_1, \ldots, \gamma_m\}$ be the variables in $isolated(B)$. Then,

$$\mathsf{A}'' = \bigcup_{i=1}^{m} \{A(b_i) \mid A \in Unary(\mathsf{r}, \gamma_i)\}\ \text{, where each } b_i \text{ is a fresh constant}$$

Let $process(\mathcal{G}(B)) = \{\Sigma_1, \ldots, \Sigma_n\}$, and for each Σ_i, let α_i denote the root node for Σ_i. Then,

$$\mathsf{T}' = \bigcup_{i=1}^{n} \{c_i \doteq compile(\alpha_i)\}\ ,\ \mathsf{A}' = \bigcup_{i=1}^{n} \{c_i(a_i)\}$$

where each a_i is a fresh constant and each c_i a fresh concept name

$\triangle$ *updated via* B is defined as $(\mathsf{R},\ \mathsf{T} \cup \mathsf{T}',\ \mathsf{A} \cup \mathsf{A}' \cup \mathsf{A}'' \cup ground(B))$

The updates defined for rule (3), on the basis of the graphs in fig.1c), are (we omit writing $\top$ if $\top$ is one among a number of conjuncts):

$c1 \doteq i \sqcap (\exists p.(r^{-1} : a) \sqcap (\exists m^{-1}.\top)) \sqcap q : a$, $c1(a1)$, $c2 \doteq \exists t.s^{-1} : a$, $c2(a2)$

Referring to the medical planning example of section 1, assuming a knowledge base $\triangle = (\mathsf{R}, \mathsf{T}, \mathsf{A})$, where R is the set of plan advisor rules, T a medical terminology encoded in a superset of $\mathcal{ERIB}$, and A contains facts about a plan being designed and facts about the medical domain, then def. 10 will update T and A with $prevent\_dehydration\_act \doteq action \sqcap \exists effect.prevent : dehydration$, and $prevent\_dehydration\_act(a2)$. Note that the properties of $a2$ can be reasoned about, via hybrid reasoning of the type described in [5]. For example, hybrid reasoning will determine the entailment $\triangle \models safety(drug\_A, dehydration, a2)$. Indeed, to facilitate rapid re-implementation and thus demonstrate proof of concept, we chose to simulate hybrid reasoning of the above type, by translating T to a definite program T^*, thus obtaining the definite program $\triangle^* = (\mathsf{R} \cup \mathsf{T}^* \cup \mathsf{A})$ (we refer the reader to [7] for details of the translation and re-implementation). At plan execution time the place holder action $a2$ can be "instantiated" by a specific action which can be checked to determine that it is an instance of $prevent\_dehydration\_act$; effectively checking for compliance with the intentions encoded in the concept definition.

Finally, to demonstrate "correctness" of the procedure for update computation, the following theorem states that given a rule r, then a ground instance of the head of r is entailed by r and the *skolemised* first order translation of the updates computed on the basis of the body of r. Firstly, if d is an $\mathcal{ERIB}$ concept expression, and α a constant or variable, then the first order translation $fol(d, \alpha)$ is defined as follows:

1) if d is atomic (a concept name) then $fol(d, \alpha) = d(\alpha)$
2) if $d = d_1 \wedge d_2$ then $fol(d, \alpha) = fol(d_1, \alpha) \wedge fol(d_2, \alpha)$
3) if $d = \exists R.d'$ then $fol(d, \alpha) = fol(R, \alpha, Y) \wedge fol(d', Y)$, Y is a fresh variable
4) if $d = R : b$ then $fol(d, \alpha) = fol(R, \alpha, b)$,

where $fol(R, \alpha, \beta)$ is defined as follows: Let R denote a role expression [2] $p_1 \sqcap \ldots \sqcap p_j \sqcap q_{j+1}^{-1} \sqcap \ldots \sqcap q_n^{-1}$. Then, $fol(R, \alpha, \beta) = \bigwedge_{i=1}^{j} p_i(\alpha, \beta) \wedge \bigwedge_{k=j+1}^{n} q_k(\beta, \alpha)$.

Correctness of the above translation should be obvious given the first order semantics for the operators. Hence, $c \doteq d$ is equivalent to $\forall X c(X) \leftrightarrow fol(d, X)$, and so for each updated pair $(c_i \doteq d_i,\ c_i(a_i))$, $\forall X c_i(X) \leftrightarrow fol(d_i, X), c_i(a_i) \models fol(d_i, a_i)$. Substituting a skolem constant for each fresh variable introduced by 3), we thus obtain the skolemised first order entailment $skol(fol(d_i, a_i))$, where $skol(fol(d_i, a_i))$ is a conjunction of ground unary and binary predicates.

Theorem 4 *Let* r *be a computable rule* $H(\overline{Y}) \leftarrow B_1(\overline{X}_1), \ldots, B_n(\overline{X}_n)$. *Let the updates computed from the body of* r, *as given by def.10, be:*

$$\mathsf{T}' = \{c_1 \doteq d_1, \ldots, c_n \doteq d_n\},\ \mathsf{A}' = \{c_1(a_1), \ldots, c_n(a_n)\},\ \mathsf{A}'',\ ground(B).$$

Let $\Gamma = \bigcup_{i=1}^{n} \{skol(fol(d_i, a_i))\} \cup \mathsf{A}'' \cup ground(B)$. *Then:* $\Gamma, \mathsf{r} \models H(\overline{a})$.

Proof. It is sufficient to show that Γ entails ground instances of every predicate in B with a valid substitution of constants for variables in B. $ground(B) \subseteq \Gamma$, and $B_i(a_i) \in \mathsf{A}''$ if $B_i(X_i)$ is in $isolated(B)$ (X_i is not a variable in any binary). Let B'' denote the remaining non-ground unary and binary predicates (those not in $ground(B)$ or $isolated(B)$). Each binary predicate p_b in B'' labels an edge in $\mathcal{G}(B)$. We show that p_b

[2] We can assume any role expression to be of this form since role conjunction is associative and commutative, and inverse role is distributive

or p_b^{-1} labels an edge in some Σ_i in $process(\mathcal{G}(B)) = \{\Sigma_1, \ldots, \Sigma_n\}$. This follows from def.5 (where every node in a variable subgraph is connected and so will be processed by *subtrans*), and definitions 6, 7-2) and 8. In def.10, every p_b / p_b^{-1} will compile to a role in $c_i \doteq (compile(\alpha_i) = d_i)$, since (referring to def.9) when compiling a variable γ_i, γ_i has no predecessor other than β (lemma 3i). Compiling a constant γ_i, γ_i is not the predecessor of any edge (lemma 3ii), and if there is an e =(β',γ_i,R) ($\beta' \neq \beta$), then by lemma 3ii) there is a route from α_i to γ_i that includes e, and so all predicates in R will be compiled. Every unary predicate in B'' shares a variable with some binary p_b in B'', and so (by def.9) will compile to a concept in some d_i. We thus have that every predicate in B'' is compiled to a role or concept in some d_i, and hence appears as a ground predicate in $\bigcup_{i=1}^{n}\{skol(fol(d_i, a_i))\}$. Since by def.7-4) no variable is common to any two Σ_i, Σ_j, then substitution of variables by skolem constants constitutes a valid substitution on B''. QED

4 Conclusions

Our work formalises compilation of the conjunctive bodies of a restricted class of Horn rules into updates on terminologies. We have illustrated application and benefits of this work in a medical planning context, but believe that it can be generalised to other incremental design tasks in which incomplete specifications of design components need to be represented and reasoned with (e.g., in configuration management [6]). An immediate goal is to link the plan advisor and plan authoring tool to established large scale medical terminologies (e.g., [10]), and further develop and implement the update procedures described here. Our work on compilation of conjunctions to concept descriptions is related to existing works on compilation of conjunctive queries [2, 4]. In [2] conjunctions are compiled into $\mathcal{ALEN}$ concept descriptions [2]. The class of conjunctions considered do not include constants, and the binding graphs are required to define a tree from the outset. In [4], compilation is defined for conjunctions with constants, and whose binding graphs include cycles and variables converged on by two or more edges. Cycles exclusively containing variables are handled by nondeterministically substituting a variable in the cycle with an individual name (constant) occurring in the ABox. Variables converged on by two or more edges are similarly substituted. However, these techniques are only suitable when compiling conjunctive *queries* and would clearly not be appropriate when updating compiled descriptions and associated instances.

Acknowledgements: Warm thanks to Maarten Marx for his valuable comments on this paper

References

[1] A. Deliyanni and R. Kowalski, Logic and Semantic Networks. In: Commun. ACM, (22,3), 184-192, Mar. 1979.

[2] F. Goasdou and M. Rousset, Compilation and Approximation of Conjunctive Queries by Concept Descriptions. In: Proceedings of the 15th European Conference on Artificial Intelligence, ECAI 2002.

[3] B. N. Grosof, I. Horrocks, R. Volz, S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In: Proceedings of 12th International Conference on the World Wide Web (WWW-2003), Budapest, Hungary, May 20-23, 2003.

[4] I. Horrocks and S. Tessaris, A Conjunctive Query Language for Description Logic Aboxes. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000), 399-404, 2000.

[5] A. Y. Levy and M. Rousset, Combining Horn Rules and Description Logics in CARIN. In: Artificial Intelligence 104 (1-2), 165-209, 1998.

[6] D. McGuinness, Description Logic for Configuration. In: eds., F. Baader, D. McGuinness, D. Nardi, and P. Patel-Schneider, The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2002.

[7] S. Modgil, Linking Rules to Terminologies and Applications in Medical Planning. To appear in: 9th Conference on Artificial Intelligence in Medicine in Europe, Cyprus, 2003 (http://www.eastman.ucl.ac.uk/%7Edmi/publications.html)

[8] S. Modgil and P. Hammond, Generating Symbolic and Natural Language Partial Solutions for Inclusion in Medical Plans. In: Proc. 8th Conf. on Artificial Intelligence in Medicine in Europe, (LNAI 2101, Springer-Verlag), 239-248, 2001.

[9] S. Modgil and P. Hammond, Decision Support Tools for Clinical Trial Design, In: Artificial Intelligence in Medicine, 27(2), 181-200, 2003.

[10] A. Rector et. al., The GRAIL concept modelling language for representing medical terminology. In: Artificial Intelligence in Medicine, (9), 139-171, 1997.

[11] Y. Shahar, S. Miksch, P. Johnson, The Asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines, In: Artificial Intelligence in Medicine, 14(1-2), 29-51,1998.