Decomposition of Declarative Knowledge Bases with External Functions*

Thomas Eiter and **Michael Fink** and **Thomas Krennwallner**
Institute of Information Systems, Vienna University of Technology
Favoritenstrasse 9–11, A-1040 Vienna, Austria
{eiter,fink,tkren}@kr.tuwien.ac.at

Abstract

We present a method to decompose a declarative knowledge base, given by a logic program under Answer Set Semantics with access to external sources. It overcomes the ineffectiveness of current methods due to a lack of structural information about these sources, viewed as black boxes, by exploiting independency information in accesses to them. To this end, we develop a generic notion of domain independence that allows to restrict the evaluation domain and, as a consequence, to prune unnecessary dependency assumptions between atoms. This leads to increased decomposability; we demonstrate this by an evaluation method for HEX-programs based on program rewriting, which may yield large performance gains. While developed for a particular formalism, the notions and ideas of this paper might be adapted to related formalisms as well.

1 Introduction

In the last years, there has been a steady increase of the desire to integrate different information sources and software into applications, in order to provide enhanced usability. To address this need, some declarative knowledge representation formalisms have been extended with the capability to access external sources. Often, this is realized via an interface in the style of an API; examples of rule based such formalisms are various extensions of ASP solvers like DLV, XSB,[1] Agent Programs [Subrahmanian *et al.*, 2000], dl-programs [Eiter *et al.*, 2008], defeasible description logic [Wang *et al.*, 2004], Prolog engines with an interface for external functions, etc.

However, the enhanced capabilities of such formalisms come at a performance price. Indeed, structural dependency information, which is usually exploited for efficient evaluation, is blurred or completely lost by access through external sources. This is an inevitable consequence of treating these sources as *black boxes*, which causes one to cautiously assume that numerous dependencies exists, even if this not true in reality. This may in particular be detrimental in case of mutual dependency between the information in the knowledge base and the external source, which can lead to almost universal dependency assumptions. As a result, decomposing the knowledge base into smaller, local parts for evaluation (like strongly connected components or splitting sets [Lifschitz and Turner, 1994]) is hindered, and large parts have to be evaluated in an uninformed generate and test manner.

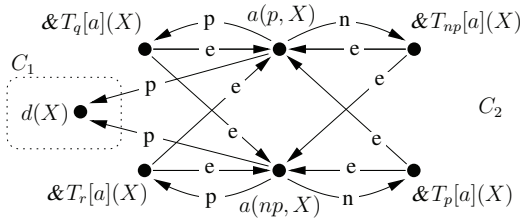


Figure 1: Dependency graph of P_N

We illustrate this on the canonical Nixon Diamond, couched in HEX-programs [Eiter *et al.*, 2006a], which extend Answer Set Programming (ASP) with access to external sources via *external atoms*, in a setting where default rules should be imposed on top of an external ontology.

Example 1 (Nixon Diamond) Suppose $T = \{r(n), q(n), \forall x.\, np(x) \equiv \neg p(x)\}$ contains the facts that Nixon was a republican ($r(n)$) and a quaker ($q(n)$); we further know that (i) quakers are normally pacifists, while (ii) republicans are normally not pacifists. In spirit of dl-programs [Eiter *et al.*, 2008], the HEX-program P_N below encodes (i) and (ii) in the rules (2) and (3), resp.; there, $a(p, X)$ and $a(np, X)$ encode the (reified) assumptions that X is a pacifist resp. is not a pacifist. The program accesses T via external atoms of form $\&\, T_\chi[a](X)$, which evaluate to true if, after adding all facts $\pi(c)$ to T such that $a(\pi, c)$ is in the model of the program, the atom $\chi(X)$ is true in T ($d(X)$ is a simple domain predicate):

$$d(n). \quad (1)$$

$$a(p, X) \leftarrow d(X), \&\, T_r[a](X), \text{not } \&\, T_{np}[a](X). \quad (2)$$

$$a(np, X) \leftarrow d(X), \&\, T_q[a](X), \text{not } \&\, T_p[a](X). \quad (3)$$

Without explaining the semantics here in detail, P_N has two models, $\{d(n), a(p, n)\}$ and $\{d(n), a(np, n)\}$, corresponding to the views that Nixon is a pacifist resp. is not a pacifist.

Possible dependencies between atoms are shown in the dependency graph of P_N in Figure 1, where an arc from α to

*This work has been supported by the Austrian Science Fund (FWF) projects P20840 and P20841.

[1] See www.dlvsystem.com, xsb.sourceforge.net

β stands for a (possible) dependency of α from β, and "n" means dependency through negation ("not"). All atoms except $d(X)$ mutually depend on each other, and form one component, C_2, for evaluation, which depends on the (singleton) component C_1. Now suppose we have m clones of Nixon and $d(n_1), \ldots, d(n_m)$ instead of $d(n)$ in (1). Then the respective program P_N^m instantiates to m copies, and we have 2^m models that contain $d(n_1), \ldots, d(n_m)$ and one of $a(p, n_i)$, $a(np, n_i)$, for each $i = 1, \ldots, m$. However, the dependency graph of P_N^m's grounding, which traditionally is the basis for efficient evaluation, is *not* the union of k copies of the dependency graph of P_N's grounding, as intuitively expected: there is, e.g., an arc $\&\, T_p[a](n_1) \rightarrow a(p, n_2)$, reflecting that $a(p, n_2)$ could impact the query $p(n_1)$ to T with update a. In fact, according to the dependency graph of P_N^m's grounding, all atoms except the $d(n_i)$ are on negative cycles, in particular all external atoms; a standard evaluation method thus has to consider all these atoms at once, along with (exponentially many) different guesses for possible input a to them. This results in long execution times. □

Due to the black box view, a traditional evaluation of P_N^m is unaware that each external atom actually depends *only on two facts*, e.g., $\&\, T_p[a](n_i)$ on $a(p, n_i)$, $a(np, n_i)$, and that (as by intuition) indeed a splitting of the P_N^m into k disjoint copies is feasible which can be evaluated independently.

This calls for refined methods to decompose a declarative knowledge base with access to external sources. In this paper, we address this issue and make the following contributions.

- We consider *domain independence information* as a basis for more effective decomposition of programs with access to external sources. Domain independence means that the evaluation of an external atom depends, for a given set of arguments, *only on a subset of the domain*, and that other elements in the domain can be safely ignored. Building on this, the idea is to split the domain in different parts such that external atoms can be evaluated independently in different parts. This is formalized as *greatest local split* of a set of ground atoms, which is algebraically defined in Section 3.
- Using greatest local splits, we refine the traditional method of evaluating programs using the ground dependency graph, by pruning unnecessary dependency assumptions of atoms caused by external functions (Section 4).
- We develop a decomposition method for HEX-programs that uses program rewriting on the non-ground level, based on independency information (Section 4). The evaluation of programs resorts then to standard technology, such that the method can be realized with rather little effort (Section 5).

Some experiments with an implementation, in particular on the program in Example 1 above, show that the rewriting may lead to large performance gains and scalability.

2 Preliminaries

Partial orders and partitions. Recall that a poset $(S, \preceq)$ has a binary relation $\preceq$ on a set S that is reflexive, anti-symmetric, and transitive. For any $X \subseteq S$, an element $u \in S$ is an *upper bound of* X, if $b \preceq u$ for all $b \in X$, and is the *least upper bound of* X, if $u \preceq u'$ for all upper bounds u' of X.

A *partition* π on a set S is collection of nonempty disjoint subsets of S such that $S = \bigcup \pi$.

Syntax of HEX-Programs. Let $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ be mutually disjoint sets whose elements are called *constant names*, *variable names*, and *external predicate names*, respectively. Unless explicitly specified, elements from $\mathcal{X}$ (resp., $\mathcal{C}$) are denoted with first letter in upper case (resp., lower case), while elements from $\mathcal{G}$ are prefixed with the "$\&$" symbol. We note that constant names serve both as individual and predicate names.

Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms*. A *higher-order atom* (or *atom*) is a tuple $(Y_0, Y_1, \ldots, Y_n)$, where $Y_0, \ldots, Y_n$ are terms; $n \geq 0$ is the *arity* of the atom. Intuitively, Y_0 is the predicate name; we thus use the more familiar notation $Y_0(Y_1, \ldots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant.

An *external atom* is of the form

$$\&\, g[Y_1, \ldots, Y_n](X_1, \ldots, X_m) \tag{4}$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called *input* and *output* lists, respectively), and $\&\, g \in \mathcal{G}$ is an external predicate name. We assume that $\&\, g$ has fixed lengths $in(\&\, g) = n$ and $out(\&\, g) = m$ for input and output lists, respectively. Intuitively, an external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates. For any set of atoms A, we denote by $\&\, A$ the set of external atoms in A.

A *rule* r is of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_m, \mathrm{not}\, \beta_{m+1}, \ldots, \mathrm{not}\, \beta_n \tag{5}$$

where $m, k \geq 0$, each α_i is an atom, and each β_j is an atom or an external atom. We let $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ and $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$. Rule r is a *constraint*, if $H(r) = \emptyset$ and $B(r) \neq \emptyset$, and a *fact*, if $B(r) = \emptyset$ and $H(r) \neq \emptyset$.

A HEX-*program* (or *program*) is a finite set P of rules; it is *ordinary*, if all its rules are *ordinary*, i.e., it contains only ordinary atoms.

Example 2 The program P_N for the Nixon Diamond above consists of one fact and two rules that use negation "not", and they have external atoms $\&\, T_r[a](X)$, $\&\, T_{np}[a](X)$, etc. □

Semantics of HEX-Programs. The *Herbrand base* HB_P of a HEX-program P is the set of all possible ground instances of atoms and external atoms in P using constants from $\mathcal{C}$. The grounding of a rule r, $grnd(r)$, is analogous, and the grounding of P is given by $grnd(P) = \bigcup_{r \in P} grnd(r)$. By default, $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ are implicitly given by P.

An *interpretation* of P is any subset $I \subseteq HB_P$ that contains only ordinary atoms; we say that I is a *model* of

– an atom $\alpha \in HB_P$, denoted $I \models \alpha$, if $\alpha \in I$.
– a ground external atom $\alpha = \&\, g[\mathbf{c_2}](\mathbf{c_1})$ (denoted $I \models \alpha$), if $f_{\&g}(I, \mathbf{c_2}, \mathbf{c_1}) = 1$, where $f_{\&g}$ is a (fixed) $(n+m+1)$-ary Boolean function for $\&\, g$, where $n = in(\&\, g)$, $m = out(\&\, g)$, $I \subseteq HB_P$, and $\mathbf{c_2} \in \mathcal{C}^n$, $\mathbf{c_1} \in \mathcal{C}^m$.
– a ground rule r ($I \models r$), if $I \models H(r)$ or $I \not\models B(r)$, where (i) $I \models H(r)$, if $I \models \alpha$ for some $\alpha \in H(r)$ and (ii) $I \models B(r)$, if $I \models \alpha$ for all $\alpha \in B^+(r)$ and $I \not\models \alpha$ for all $\alpha \in B^-(r)$.
– a program P ($I \models P$), iff $I \models r$ for all $r \in grnd(P)$.

The *FLP-reduct* [Faber *et al.*, 2004] of a program P wrt. an interpretation I is $fP^I = \{r \in grnd(P) \mid I \models B(r)\}$; then I is an *answer set of* P, if I is a $\subseteq$-minimal model of fP^I. For ordinary P, this amounts to answer sets as in [Gelfond and

Lifschitz, 1991]. We denote by $AS(P)$ the collection of all answer sets of P.

Example 3 In the Nixon example, the semantics of external atoms may be formally given as $f_{\&T_\chi}(I, c', c) = 1$ iff $T \cup \{d(d') \mid c'(d, d') \in I\} \models \chi(n)$, for all $c, c' \in \mathcal{C} = \{n, p, np, a\}$. Then, $I = \{d(n), a(p, n)\}$ is an answer set of P_N: clearly, $I \models fP_N{}^I$, and $fP_N{}^I$ includes (1) and the instance of (2) for $X{=}n$; hence, each model $J \subseteq I$ of $fP_N{}^I$ fulfills $J \supseteq I$.

In the rest of this paper, we assume w.l.o.g. that all input terms in external atoms are viewed as predicate symbols.

We next recall the notions of dependency graph and global splitting sets of a program [Eiter *et al.*, 2006a], which provides a basis for efficient HEX-program evaluation.

For a program P and atoms α, β occurring in P, we say

(a) α *depends positively on* β ($\alpha \to_p \beta$), if either

(i) some rule $r \in P$ has $\alpha \in H(r)$ and $\beta \in B^+(r)$;

(ii) there are rules $r_1, r_2 \in P$ such that $\alpha \in B(r_1)$ and $\beta \in H(r_2)$ and there exists a partial substitution θ of variables in α such that either $\alpha\theta = \beta$ or $\alpha = \beta\theta$; or

(iii) some rule $r \in P$ has $\alpha, \beta \in H(r)$.

(b) α *depends externally on* β ($\alpha \to_e \beta$), if α is is of form $\&g[X_1, \ldots, X_n](\mathbf{Y})$, β is of form $p(\mathbf{Z})$, and $X_i = p$ for some $i \in \{1, \ldots, n\}$.

(c) α *depends negatively on* β ($\alpha \to_n \beta$), if there is some rule $r \in P$ such that either $\alpha \in H(r)$ and $\beta \in B^-(r)$ or β is a nonmonotonic external atom (i.e., $I \subseteq I'$ and $I \models \beta$ does not entail $I' \models \beta$).

We say that α *depends on* β, if $\alpha \to \beta$, where $\to$ is the union of $\to_p$, $\to_e$, and $\to_n$.

The *dependency graph* $G_P = (V_P, E_P)$ of P has as vertices V_P all atoms occurring in P and as edges E_P all dependency relations $\to_p$, $\to_e$, and $\to_n$ for P

A *global splitting set* for a HEX-program P is a set of atoms A occurring in P such that $\alpha \in A$, β occurs in P, and $\alpha \to \beta$ implies $\beta \in A$. The *bottom* of P wrt. a set of atoms A is the set of rules $b_A(P) = \{r \in P \mid H(r) \cap A \neq \emptyset\}$.

Example 4 Figure 1 shows the dependency graph of P_N; note that all external atoms are monotonic, and $C_1 \cup C_2$ (all atoms) and C_1 are global splitting sets. □

3 Domain Independence

In this section, we develop our notion of domain independence. The idea is to partition the elements of the domain into disjoint sets, such that the evaluation of an external atom only depends on the elements that are in the sets associated with the arguments of the atom. We call a suitable such partitioning a *basis*. For program decomposition, we need to take the structure of the rules into account, respecting syntactic dependency of atoms and arguments according to the dependency graph, assuming that elements in the same partition are mutually dependent. This leads us to a refinement of the basis (i.e., sets are further partitioned) to a unique *greatest local split* of a set of atoms. The latter is defined algebraically.

Notation. We start with fixing some notation. In what follows, let $D = \mathcal{C}$ (for *domain*).

Let $(\mathcal{D}, \preceq)$ be the poset of the lattice of partitions of D, where for any two partitions $\pi_1, \pi_2 \in \mathcal{D}$, $\pi_1 \preceq \pi_2$ iff $\forall x \in \pi_1\ \exists y \in \pi_2 \colon x \subseteq y$ (i.e., $\preceq$ is the usual refinement order).

For any set $S \subseteq D$ and m-ary predicate symbol p, we let $p[S] = \{p(\mathbf{c}) \mid \mathbf{c} \in S^m\}$, and for a list $\mathbf{p} = p_1, \ldots, p_k$ of predicate symbols, we let $\mathbf{p}[S] = p_1[S] \cup \cdots \cup p_k[S]$.

We now define the notion of a basis for an external atom.

Definition 1 (Basis) *Let $B \subseteq D$, let $\alpha = \&g[\mathbf{p}](\mathbf{c})$ be a ground instance of an external atom over D with Boolean function $f_{\&g}$. We call B a* basis for α, *if for all interpretations I and $B' \supseteq B$, $f_{\&g}(I \cap \mathbf{p}[B], \mathbf{p}, \mathbf{c}) = f_{\&g}(I \cap \mathbf{p}[B'], \mathbf{p}, \mathbf{c})$.*[2]

Example 5 Let D be a domain, $I = \{p(a), p(b), q(a)\}$ be an interpretation, and $\mathbf{p} = p, q$. Let $B_1 = \{a\}$ and $B_2 = \{b, c\}$ be two subsets of D. Then, $I \cap \mathbf{p}[B_1] = \{p(a), q(a)\}$ and $I \cap \mathbf{p}[B_2] = \{p(b)\}$. Now suppose $\alpha = \&g[p](a)$ is such that for all interpretations I, $f_{\&g}(I, p, a) = 1$ iff $p(a) \in I$. Then, B_1 is a basis for α, and $f_{\&g}(I, p, a) = f_{\&g}(I \cap p[B_1], p, a)$.

Given an external atom α, define for any $a, b \in D$ that $a \sim_\alpha b$ iff $a, b \in B$ for basis B for α, and $c \sim_\alpha c$ for every $c \in D$. Then $\sim_\alpha$ is an equivalence relation, whose quotient set $D/\!\sim_\alpha$ forms a partition of D.

The notion of greatest local split is now defined as follows.

Definition 2 (Greatest Local Split) *For any set of ground atoms A, the* greatest local split of A, *denoted $gls(A)$, is the partition $\pi \in \mathcal{D}$ which is the least upper bound of $\mathcal{B} = \{D/\!\sim_\alpha \mid \alpha \in \&A\}$ in $(\mathcal{D}, \preceq)$ (recall $\&A$ from above).*

Obviously, $gls(A)$ is unique and always exists.

Example 6 Let $\sim_\alpha = \{(a,a), (b,b), (c,c), (d,d)\}$ and $\sim_\beta = \{(a,a), (a,b), (b,a), (b,b), (c,c), (d,d)\}$ be equivalence relations over $D = \{a, b, c, d\}$ for α and β, respectively. Then, $D/\!\sim_\alpha = \{\{a\}, \{b\}, \{c\}, \{d\}\}$ and $D/\!\sim_\beta = \{\{a, b\}, \{c\}, \{d\}\}$. Now let A be a set of ground atoms such that $\alpha, \beta \in A$. We obtain $gls(A) = D/\!\sim_\beta = \{\{a, b\}, \{c\}, \{d\}\}$, whenever all possible $\sim$ of external atoms in A are equal to $\sim_\alpha$ or to $\sim_\beta$. □

The following result then, whose proof is not difficult, forms the basis for exploiting joint dependency information.

Proposition 1 *Let $\&g[\mathbf{p}](\mathbf{c})$ be an external atom, let A be a set of atoms such that $\&g[\mathbf{p}](\mathbf{c}) \in grnd(A)$, and let I be an interpretation. Then, for every partition π such that $gls(A) \preceq \pi$, there exists some $B' \in \pi$ such that*

$$f_{\&g}(I, \mathbf{p}, \mathbf{c}) = f_{\&g}(I \cap \mathbf{p}[B'], \mathbf{p}, \mathbf{c}).$$

By this proposition, we can partition the domain, at different levels of granularity, in a way such that we can evaluate the atoms in A in different sets of the partition.

Note that computing the greatest local split is inexpensive.

Proposition 2 *Given a set of atoms A and $\mathcal{B}$ as in Def. 2, we can compute $gls(A)$ in linear time in the size of A and $\mathcal{B}$.*

Indeed, we can compute $gls(A)$, e.g., as connected components of a graph that we construct from $\mathcal{B}$ in linear time.

4 Rewriting with Domain Independence

In this section, we apply the results obtained in the previous section in order to improve the evaluation of HEX-programs

[2] As $f_{\&g}$ only depends on $\mathbf{p}$, $f_{\&g}(I \cap \mathbf{p}[B], \mathbf{p}, \mathbf{c}) = f_{\&g}(I, \mathbf{p}, \mathbf{c})$.

by taking dependency information of external atoms into account. The idea is to first prune unnecessary ground dependencies from the dependency graph G_P of a given program P, like for $\&\, T_p[a](n_1) \rightarrow a(p, n_2)$ in the grounding of P_N^m with m Nixon clones. In a second step, we rewrite the program into an equivalent program, such that the evaluation of strongly connected components is restricted to relevant subdomains whenever possible. Intuitively, this step accounts for unnecessary dependencies between non-ground atoms.

In this section, we assume that the domain D is finite; note that even for a basically infinite set $\mathcal{C}$, P may be equivalent to its grounding over a finite subset of $\mathcal{C}$; e.g., domain-expansion safe programs [Eiter *et al.*, 2006a] have this property.

Step 1: Dependency Graph Pruning. We assume that we are given a HEX-program without higher order atoms—note that w.l.o.g. higher order atoms can be removed by partial grounding—and that every ground external atom α in the program has an associated basis B_α in D. Furthermore, for a proper treatment of the dependency between input and output of a ground external atom, we assume bases B_α, such that all elements of the output $\mathbf{c}$ of α are in B_α.

We first reduce the dependency graph G_P to a graph G'_P as follows. For any graph G, let $SCC(G)$ be the set of strongly connected components in G. Consider any component $C \in SCC(G_P)$ and $gls(grnd(C))$ (which is uniquely determined by the bases of external atoms in $grnd(C)$). We remove every arc $\&\, g[\mathbf{p}](\mathbf{c_1}) \rightarrow p(\mathbf{c_2})$ from C, such that $\mathbf{c_1}$ and $\mathbf{c_2}$ are variable-free and there is an element in $\mathbf{c_2}$ which does not belong to the partition of $\mathbf{c_1}$ in $gls(grnd(C))$. Let G'_P denote the resulting graph. Then we can evaluate P using G'_P instead of G_P. More formally, this is detailed as follows.

Let $Comp = \{C_1, \ldots, C_n\}$ be a partitioning of the atoms occurring in P such that all $grnd(C_i)$ are pairwise disjoint, and let P_{C_i} contain all rules $r \in grnd(P)$ such that $H(r) \cap grnd(C_i) \neq \emptyset$, $i = 1, \ldots, n$. A partial ordering $\mathcal{E}(P) = (Comp, \sqsubseteq)$ is then an *evaluation ordering* for P, if $H(r) \subseteq grnd(C_i)$, for each $r \in P_{C_i}$, $i = 1, \ldots, n$, and the following property (6) holds. Let $P_{C_i}{\downarrow} = \bigcup_{C_j \sqsubseteq C_i} P_{C_j}$; then

$$AS(P_{C_i}{\downarrow}) = \bigcup_{\substack{M_j \in AS(P_{C_{i_j}}{\downarrow}),\\ j=1,\ldots,n_i}} AS(P_{C_i} \cup \bigcup_{j=1}^{n_i} M_j); \quad (6)$$

where $\{C_{i_1}, \ldots, C_{i_{n_i}}\} = \{C_j \sqsubset C_i \mid \forall C_{j'} : C_j \sqsubseteq C_{j'} \sqsubset C_i \Rightarrow C_j = C_{j'}\}$ are the maximal sets C_j smaller than C_i. That is, the answer sets of the rules associated with C_i and all C_j below are obtained as the answer sets of the rules associated with C_i to which the answer sets of the rules associated with C_j and all C'_j below are added.

Then, given that $C_{i_1}, \ldots, C_{i_l}$ are the maximal elements of $\mathcal{E}(P)$ and P_{con} are all constraints in P, it holds that

$$AS(P) = \left\{ N = \textstyle\bigcup_{j=1}^{l} M_{i_j} \mid M_{i_j} \in AS(P_{C_{i_j}}{\downarrow}), \; 1 \leq j \leq l, N \models P_{con} \right\}.$$

It is well-known that $SCC(G_P)$ is an evaluation ordering for P, cf. [Eiter *et al.*, 2006a]. We similarly have:

Proposition 3 *$SCC(G'_P)$ is an evaluation ordering for P.*

Intuitively, the strongly connected components of G'_P refine those of G_P by removing unnecessary input dependencies of external atoms. We note that coarser evaluation orderings can be obtained e.g. by collapsing suitable components in $SCC(G_P)$ resp. $SCC(G'_P)$. As for us the precise evaluation ordering is immaterial, we work in the sequel with a generic given evaluation ordering $\mathcal{E}(P)$.

Step 2: Program Rewriting. We proceed by introducing a rewriting of P wrt. $\mathcal{E}(P)$ in order to further improve the evaluation of non-ground programs. The idea is to restrict the applicability of rules with external atoms to relevant ground instances given respective domain independence information. To avoid inhibiting the application of relevant ground instances, and to keep the rewriting general and simple, we consider here the maximal number $k = k(P)$ of distinct variables in a rule $r \in P$ as a bound for potential dependencies.[3]

In the following, we let for every $p \in \mathcal{C}$ be p^s a fresh symbol with arity of p. For a set of atoms A, let

$$const(A) = \{c_1, \ldots, c_n \mid p(c_1, \ldots, c_n) \in A \setminus \& A\} \cap \mathcal{C}.$$

The *i-restricting substitution in A* is the partial substitution

$$\sigma_A^i = \{p/p^i \mid p(\mathbf{t}) \in A\},$$

which maps predicates p of every ordinary atom $p(\mathbf{t})$ in A to p^i. An *i-restriction of A in P*, short P_A^i, is the set of rules

$$\{H(r)\sigma_A^i \leftarrow B(r)\sigma_A^i, dom^i(X_1), \ldots, dom^i(X_r) \mid r \in b_A(P)\},$$

where dom^i is a unary predicate symbol and $X_1, \ldots, X_r$ are all output variables which occur in some external atoms in $B(r)$. Furthermore, let

$$\pi_\cup^k(A) = \{S_1 \cup \cdots \cup S_k \cup const(A) \mid S_1, \ldots, S_k \in \pi\}$$

be the k-fold union of a partition π augmented with $const(A)$.

Definition 3 (Decomposition $d_C(P)$) *Let $C \in \mathcal{E}(P)$ and $gls(grnd(C))_\cup^k(C) = \{D_1, \ldots, D_\ell\}$ for $k = k(P)$. The* decomposition of P wrt. C, *denoted $d_C(P)$, is defined as*

$$\textstyle\bigcup_{1 \leq i \leq \ell} P_C^i \cup \{\alpha \leftarrow \alpha^i \mid \alpha \in H(r), r \in b_C(P)\} \cup dom^i[D_i]\,.$$

Example 7 Reconsider the cloned version P_N^m of Example 1 and an evaluation order with the components C_1, C_2 in Figure 1, where $C_1 \sqsubset C_2$. Note that this is the evaluation order given by the strongly connected components of G_P (which is not affected by the pruning step). Given that $\&\, T_\chi[a](n_i)$ only depends on $a(p, n_i)$, $a(np, n_i)$, for $1 \leq i \leq m$, we obtain that $gls(grnd(C_2))_\cup^1(C_2) = \{\{n_i, p, np\} \mid 1 \leq i \leq m\}$.

Thus, the decomposition of P_N^m wrt. C_2 contains m rewritings of rule (2), e.g., for $i = 1$ given by

$$a^1(p, X) \leftarrow d(X), \&\, T_r[a^1](X), \text{not} \,\&\, T_{np}[a^1](X), dom^1(X),$$

where dom^1 ranges over $D_1 = \{n_1, p, np\}$. Similarly for rule (3). This reduces the applicability of the two rules to the corresponding domain D_i of the respectively rewritten rules, which intuitively amounts to removing any dependencies to ordinary ground atoms (inputs) in other subdomains. □

We denote by r and r^i ground rules from $grnd(b_C(P))$ and $grnd(d_C(P))$, resp., such that there is a rule $r' \in b_C(P)$ and a ground substitution θ yielding $r = r'\theta$ and $r^i = H(r')\sigma_C^i\theta \leftarrow B(r')\sigma_C^i\theta, dom^i(X_1)\theta, \ldots, dom^i(X_r)\theta$. Moreover, r^i is called *D_i-applicable* iff $c \in D_i$ for

[3] This is a cautious overestimate, and often k may be smaller.

every $dom^i(c) \in B(r^i)$. Given $C \in \mathcal{E}(P)$, with $gls(grnd(C))^k_\cup(C) = \{D_1, \ldots, D_\ell\}$ for $k = k(P)$, and a set $N \subseteq HB_P$ of ground ordinary atoms, let $\mathcal{M}(N)$ be the set

$$\mathcal{M}(N) = N \cup \bigcup_{i=1}^{\ell}(N^i \cup dom^i[D_i]),$$

where $N^i = \{p^i(\mathbf{c}) \mid p(\mathbf{c}) \in N \wedge (\varphi_a \vee \varphi_b)\}$, with conditions

φ_a = $\exists\, p(\mathbf{c}') \in grnd(C) \wedge p(\mathbf{c}) \notin grnd(C)$, and
φ_b = $\exists\, r^i \in grnd(d_C(P)) : r^i$ is D_i-applicable $\wedge\ p^i(\mathbf{c}) \in H(r^i) \cup B(r^i)\ \wedge\ N \models B(r) \setminus (grnd(C) \cup \&B(r)$.

Furthermore, for any set M of ground ordinary atoms, $M|_{\mathcal{C}} = \{p(\mathbf{c}) \in M \mid p \in \mathcal{C}\}$. Obviously, if M is of the form $\mathcal{M}(N)$, then $N = M|_{\mathcal{C}}$.

Informally, given a set N of ground ordinary atoms, in addition to the original atoms and a respective domain closure (i.e., $dom^i[D_i]$ atoms) $\mathcal{M}(N)$ contains a rewritten atom $p^i(c)$ for some $p(c) \in N$: If $p(c)$ is an atom that is modified by the rewriting and appears in a potentially applicable rule in P^i_C (Condition φ_b), or if $p(c)$ is an input atom, i.e., one that is not modified by the rewriting, such that the predicate p is subject to rewriting, however (Condition φ_a). The latter guarantees that it is taken into account as (potential) input for rewritten external atoms (as it is for the respective unmodified external atoms). Towards correctness of the rewriting technique, we first establish the following lemma.

Lemma 4 *Let $C \in \mathcal{E}(P)$, let $r \in grnd(b_C(P))$, and let $r^i \in grnd(d_C(P))$ such that r^i is D_i-applicable. Suppose $N \subseteq HB_P$ consists of ordinary ground atoms such that, for every $\&g[\mathbf{p}](\mathbf{c}) \in B(r) \cap grnd(C)$ and ground atom α, it holds that $\alpha \in N \cap \mathbf{p}[D_i]$ iff $\alpha\sigma^i_C \in \mathcal{M}(N) \cap \mathbf{p}\sigma^i_C[D_i]$. Then, $N \models B(r)$ iff $\mathcal{M}(N) \models B(r^i)$.*

Note that the condition on α could be weakened exploiting information on external atoms in $b_C(P)$, e.g., monotonicity.

For the simple rewriting to be applicable, we require the following condition. Given $b_C(P)$ and a set of ground ordinary (input) atoms I, such that $I \cap grnd(C) = \emptyset$, we call $d_C(P)$ a *unique split wrt. I*, iff for every ordinary atom $p(\mathbf{c}) \in grnd(C)$ the following holds: If $p^i(\mathbf{c}) \in H(r^i) \cup B(r^i)$ for some D_i-applicable rule $r^i \in d_C(P)$, and $I \models B(r^i) \setminus (grnd(C)\sigma^i_C \cup \&B(r^i) \cup dom^i[D_i])$, then (i) $p^j(\mathbf{c}) \in H(r^j) \cup B(r^j)$ for some D_j-applicable rule $r^j \in d_C(P)$, and $I \models B(r^j) \setminus (grnd(C)\sigma^i_C \cup \&B(r^j) \cup dom^j[D_j])$ implies $i = j$, and (ii) $p(\mathbf{c}) \in p[D_j]$ implies $i = j$. In this condition, (i) warrants that there is a unique index i of rewritten rules that can potentially infer a rewritten atom $p^i(c)$, and (ii) ensures that such atoms can constitute relevant input for external atoms in P^i_C only. This could be weakened by more advanced rewriting techniques, taking into account (i) dependencies introduced by the rules in $b_C(P)$ rather than the projective rules $\alpha \leftarrow \alpha^i$, and (ii) further (monotonicity) information concerning the external atoms in $b_C(P)$.

For the setting studied in this paper a 1-to-1 correspondence between the answer sets of $b_C(P)$ and the answer sets of its decomposition wrt. C, $d_C(P)$, holds for unique splits. More precisely, the correctness of the rewriting in this regard is expressed formally as follows.

Proposition 5 *Let $C \in \mathcal{E}(P)$, and let I be any set of ground ordinary atoms such that $I \cap grnd(C) = \emptyset$. If $d_C(P)$ is a unique split wrt. I, then $M \in AS(d_C(P) \cup \mathcal{M}(I))$ implies that $M|_{\mathcal{C}} \in AS(b_C(P) \cup I)$ and $M = \mathcal{M}(M|_{\mathcal{C}})$.*

On the other hand, the rewriting technique is complete for unique splits, i.e., given an answer set of N of $AS(b_C(P) \cup I)$, if $d_C(P)$ is a unique split wrt. I, then $\mathcal{M}(N)$ is an answer set of $d_C(P) \cup \mathcal{M}(I)$.

Proposition 6 *Let $C \in \mathcal{E}(P)$, and let I be any set of ground ordinary atoms such that $I \cap grnd(C) = \emptyset$. If $d_C(P)$ is a unique split wrt. I, then $N \in AS(b_C(P) \cup I)$ implies $\mathcal{M}(N) \in AS(d_C(P) \cup \mathcal{M}(I))$.*

Applying the global splitting theorem of [Eiter *et al.*, 2006a] and the previous results, we can compute answer sets of P iteratively. Given an evaluation ordering $\mathcal{E}(P) = (Comp, \sqsubseteq)$, let $E_1, \ldots, E_d$ be pairwise disjoint sets of components, such that $\bigcup_{i=1}^{d} E_i = Comp$ and for every $C \in E_i$, $C' \in E_j$, and $i < j$, it holds that $C' \not\sqsubset C$. For an iterative evaluation, we recursively define sets $\mathcal{I}$ of sets of ground ordinary atoms as follows: $\mathcal{I}_0 = \{\emptyset\}$, and $\mathcal{I}_{j+1} = \bigcup_{I \in \mathcal{I}_j}\{N \mid N = \bigcup_{C \in E_{j+1}} N_C(I)\}$, where $N_C(I) = M|_{\mathcal{C}}$ if $M \in AS(d_C(P) \cup \mathcal{M}(I))$ and $d_C(P)$ is a unique split wrt. I, and $N_C(I) = M$ for some $M \in AS(b_C(P) \cup I)$, otherwise. We eventually obtain the following correspondence.

Theorem 7 *Let P be a* HEX*-program. Then $N \in AS(P)$ if and only if $N \in \mathcal{I}_d$ and $N \models \{r \in P \mid H(r) = \emptyset\}$.*

This result establishes the correct application of the rewriting technique for the evaluation of a program.

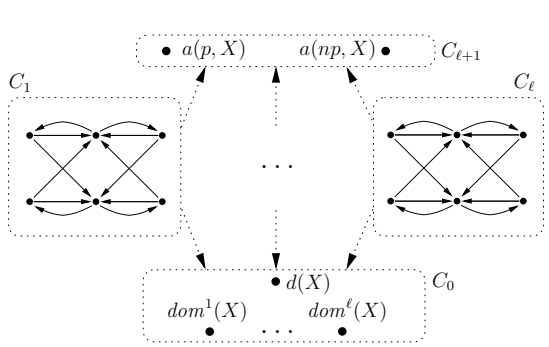
Example 8 Reconsider the cloned version P^m_N of Example 1 with the evaluation order $C_1 \sqsubset C_2$. For C_1, which is a set of facts, we obtain a single answer set of $d_{C_1}(P^m_N) \cup \emptyset$: $N_{C_1}(\emptyset) = \{d(n_i) \mid 1 \leq i \leq m\}$. Since $d_{C_2}(P^m_N)$ is a unique split wrt. $N_{C_1}(\emptyset)$, the we can compute the answer sets of P^m_N as the sets $M|_{\mathcal{C}}$, where $M \in AS(d_{C_2}(P^m_N) \cup \mathcal{M}(N_{C_1}(\emptyset)))$. □

Observe that for ground programs P, $k(P) = 0$, i.e., rule application cannot be further constrained by domain restrictions. This is reflected in the rewriting, which decomposes P into a single copy for every component without any dom predicates in the body. However, for ground programs, the pruning of dependencies as described in Step 1 is most effective, and captures the cases otherwise handled by the rewriting.

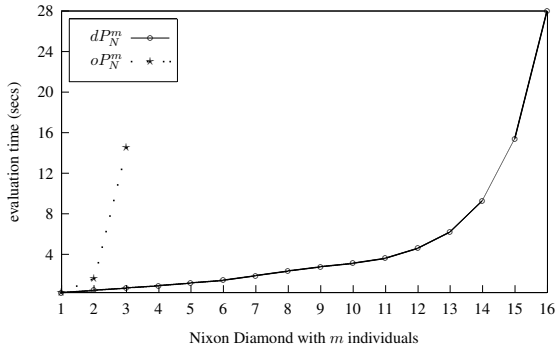
On the other hand, for non-ground programs, the evaluation may be improved by partial grounding, which reduces $k(P)$ (thus the number of domain partitions to be joined in a k-fold union). We leave a more fine-grained analysis of the program in order to reduce $k(P)$ as further work for optimization.

5 Realization and Experimentation

The standard algorithm to evaluate a given HEX-program P was shown in [Eiter *et al.*, 2006b]. Its principal idea is to break up the computation into smaller tasks, and, in spirit of [Lifschitz and Turner, 1994], to perform a bottom-up evaluation of the using the strongly connected components of G_P. The procedure uses a subroutine *eval*(P, C, I), which helps creating the models wrt. a component C and an interpretation I: *eval* keeps track of cycles in C and chooses the appropriate solving strategy for it, and generates models compliant



(a) Schematic view of dP_N^m for $m = \ell$



(b) oP_N^m vs. dP_N^m (all answer sets)

Figure 2: Nixon decomposition and experiment

Algorithm 1: *decomp-eval*(P, C, S, I): Compute the answer sets of a component C and greatest local split S in P wrt. I

Input: HEX-program P, component C, $S = gls(grnd(C))$, interpretation I
Result: collection of answer sets $\mathcal{N}$

(a) Compute $S^k_\cup(C) = \{D_1, \ldots, D_\ell\}$
$P' = I$

(b) **for** $j := 1, \ldots, \ell$ **do**
  Set $\sigma^j_C = \{p/p^j \mid p(\mathbf{t}) \in C\}$
  $P' = P' \cup P^j_C \cup \{\alpha \leftarrow \alpha\sigma^j_C \mid \alpha \in H(r), r \in b_C(P)\} \cup dom^j[D_j]$
end

(c) Evaluate P' and store the answer sets of it in $\mathcal{N}$
return $\mathcal{N}$

with the external atoms in C (see [Eiter *et al.*, 2006b] for the details).

We realize the evaluation of our decomposition method by adapting the standard evaluation algorithm. For that, we introduce the refined component evaluation subroutine *decomp-eval*(P, C, S, I) shown in Algorithm 1. The major changes for the overall refined HEX-program evaluation are:

(1) after creating the dependency graph G_P, we create from $SCC(G_P) = \{C_1, \ldots, C_n\}$ the greatest local splits $S_i = gls(grnd(C_i)), i = 1, \ldots, n$.

(2) we generate the pruned dependency graph G'_P and perform the bottom-up evaluation wrt. $SCC(G'_P)$;

(3) then, while evaluating the components, we check for each C_i whether $d_{C_i}(P)$ is a unique split wrt. I and call then

  (i) *decomp-eval*(P, C_i, S_i, I) instead of *eval* in order to evaluate a component C_i with greatest local split S_i in a HEX-program P wrt. an interpretation I, and take, transparently to the evaluation procedure, dependency information into account; otherwise,

  (ii) if $d_{C_i}(P)$ is not a unique split wrt. I, we just call *eval*(P, C_i, I).

Regarding (2), we prune G_P to G'_P by stripping ground dependencies in C_i wrt. S_i as described in Section 4. The resulting dependency graph G'_P then consists of components $Comp'$. The further evaluation of P, which is kept as is, will be performed with $Comp'$.

After preprocessing the components of P outlined as above, the refined components will be evaluated in the usual way. To this end, we may call *decomp-eval*(P, C_i, S_i, I) as described in (3i), provided that $d_{C_i}(P)$ is a unique split wrt. I (see step (3)). Alternatively, we fall back to the standard HEX-program evaluation procedure *eval*(P, C, I), as done in (3ii).

Let $C \in SCC(G'_P)$, $S = gls(grnd(C))$, and I be an interpretation. In *decomp-eval*(P, C, S, I) from above, we first generate the k-fold union of S in step (a), create $d_C(P)$ in step (b), and then evaluate the decomposed program in step (c). The result is stored in $\mathcal{N}$. The following states the correctness.

Proposition 8 *Given a* HEX*-program P, component C, greatest local split $S = gls(grnd(C))$, and interpretation I, we have $N \in AS(b_C(P) \cup I)$ iff $\mathcal{M}(N) \in \mathcal{N}$, where $\mathcal{N}$ is the output of Algorithm 1.*

For experimental evaluation, we adapted the development version of the dlvhex prototype system,[4] which is an open source reasoner for HEX-programs.

We have realized our running example, the Nixon diamond, using external atoms that query a description logic knowledge base (DL-KB) in spirit of dl-atoms in description logic programs [Eiter *et al.*, 2008]. To this end, we encoded T as DL-KB. The external atoms $\&T_\chi[a](c)$ are then realized as instance retrieval queries over T using a DL-reasoner (RacerPro 1.9.2beta).[5] All tests have been conducted on a P4 3GHz Linux system with 1GB of main memory. For a program P, we write short oP for the standard HEX-program evaluation strategy, whereas dP denotes our new decomposition evaluation strategy using *decomp-eval*.

We fixed $a \sim_\alpha a$ for all individuals a and all ground external atoms α for P^m_N's. In the decomposed program evaluation of P^m_N we have $m+2$ components (see Figure 2a for a schematic view). We computed all 2^m answer sets of the test programs, and the experimental results are displayed in Figure 2b. They show the effectiveness of the method; we

[4] www.kr.tuwien.ac.at/research/systems/dlvhex/

[5] www.racer-systems.com/

Table 1: Nixon experiments, avg. running time in seconds

m	1	2	3	4	8	16	32	64
oP_N^m	0.21	1.63	14.5	—	—	—	—	—
dP_N^m	0.22	0.43	0.65	0.91	2.33	28.12	—	—
oP_{nra}^m	0.21	1.53	14.2	—	—	—	—	—
dP_{nra}^m	0.23	0.41	0.61	0.85	2.12	5.81	16.27	49.83

gain an exponential speed-up with our decomposed evaluation strategy dP_N^m.

We have created another example by slightly adapting Example 1. If we add to P_N^m the additional rule

$$a(p, X) \leftarrow d(X), \text{not } \&\, T_{nra}[](X).$$

we obtain a program P_{nra}^m that disables the applicability of rule (3) and forces all Nixons to be pacifists. As above, we have fixed $a \sim_\alpha a$ for all individuals and all ground external atoms α for P_{nra}^m's. The evaluation with our decomposed programs then consists again of $m+2$ components. We computed the single answer set of each of these test programs.

The detailed outcome of both Nixon test series is listed in Table 1, where "—" means timeout (1 min). The results show that both oP_N^m and oP_{nra}^m evaluation have an early breakdown (m=4) while all runs of the dP_N^m evaluation up to m=16, and dP_{nra}^m up to m=64, finished in time.

The check for uniqueness of the split $d_C(P)$ was fast (a few msecs) on our examples by employing a non-recursive datalog program.

Due to space limitations, we only report here the Nixon example. We have looked at similar encodings, like the Tweety example, and the experiments show a similar picture.

To get a better understanding of the improvements of the evaluation, notice that implementations that feature external functions usually adopt a guess and check strategy when an external atom $\alpha = \&g[\mathbf{y}](\mathbf{x})$ appears in a (negative) cycle in the dependency graph. A guess represents a model of the cyclic program component and this model has to comply with the function $f_{\&g}$ for α, which may be expensive to evaluate. The guesses must be made wrt. a domain of individuals (in the worst case, all constants). Using our theory of domain independence, we can rewrite the program such that components range over smaller domains. This not only effects that fewer guesses are made for the external atoms, but also that the number of (costly) compliance checks decreases.

6 Discussion and Conclusion

We have presented a method to decompose declarative knowledge bases with access to external sources, formalized as HEX-programs, by using domain independence information. The method may lead to a large (exponential) improvement in performance, as evidenced by examples. While our method has been developed for HEX-programs, the underlying ideas and principles are generic and may be adapted for other similar KR formalisms (cf. Introduction).

To the best of our knowledge, program optimization based on domain independence about external sources, with possible non-monotonic negation, has not been addressed before. Note that domain independence is well-known in databases [Abiteboul *et al.*, 1995]. It is related to active domain semantics and usually enforced by safety of queries and rules. Domain decomposition techniques are also used in constraint processing [Cohen and Jeavons, 2003], but in a different setting.

Several issues remain for further investigation. One issue is a more involved analysis of dependencies, to make the set of atoms for which greatest local splits are computed smaller, e.g., by respecting unifiability over transitive arcs or respecting syntactic conditions (cf. [Costantini, 2006]). Also the notions of greatest local split and k-fold union may be refined. An orthogonal direction is *predicate splitting*, where input predicates p are removed from external atoms $\&g[\ldots, p, \ldots](\mathbf{X})$ whose values are independent of p wrt. relevant interpretations; this may be applied as a post processing after domain splitting.

Another issue is how to obtain dependency information. Clearly, this depends on the nature of the external sources, and there is no general recipe apart from (expensive) verification of the definitions. However, often the knowledge engineer has insight in dependency information that she can supply, like in the Nixon diamond example. To some extent, (in)dependency information may also be obtained by (semi-)automated analysis of external sources. For logic knowledge bases, notions of relevance for reasoning might be exploited, and for others (e.g., wordnet ontologies, URL lists) specific methods developed.

References

[Abiteboul *et al.*, 1995] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Cohen and Jeavons, 2003] D. Cohen and P. Jeavons. *Constraint Processing*, R. Dechter (ed.), chapter Tractable Constraint Languages. Morgan Kaufmann, 2003.

[Costantini, 2006] S. Costantini. On the existence of stable models of non-stratified logic programs. *Theory Pract. Log. Program.*, 6(1–2):169–212, 2006.

[Eiter *et al.*, 2006a] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Effective Integration of Declarative Rules with external Evaluations for Semantic Web Reasoning. In *ESWC'06*, pp. 273–287. Springer, 2006.

[Eiter *et al.*, 2006b] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Towards efficient evaluation of HEX programs. In *NMR'06*, pp. 40–46. 2006.

[Eiter *et al.*, 2008] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, 172(12-13):1495–1539, 2008.

[Faber *et al.*, 2004] W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: semantics and complexity. In *JELIA'04*, pp. 200–212. Springer, 2004.

[Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Gener. Comput.*, 9:365–385, 1991.

[Lifschitz and Turner, 1994] V. Lifschitz and H. Turner. Splitting a logic program. In *ICLP'94*, pp. 23–37. MIT Press, 1994.

[Subrahmanian *et al.*, 2000] V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, and R. Ross. *Heterogeneous Agent Systems: Theory and Implementation*. MIT Press, 2000.

[Wang *et al.*, 2004] K. Wang, D. Billington, J. Blee, and G. Antoniou. Combining description logic and defeasible logic for the Semantic Web. In *RuleML'04*, pp. 170–181. Springer, 2004.