Realizing Default Logic over Description Logic Knowledge Bases*

Minh Dao-Tran, Thomas Eiter, and Thomas Krennwallner

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{dao,eiter,tkren}@kr.tuwien.ac.at

Abstract. We consider a realization of Reiter-style default logic on top of description logic knowledge bases (DL-KBs). To this end, we present elegant transformations from default theories to conjunctive query (cq-)programs that combine rules and ontologies, based on different methods to find extensions of default theories. The transformations, which are implemented in a front-end to a DL-reasoner, exploit additional constraints to prune the search space via relations between default conclusions and justifications. The front-end is a flexible tool for customizing the realization, allowing to develop alternative or refined default semantics. To our knowledge, no comparable implementation is available.

1 Introduction

Ontologies are very important for representing terminological knowledge. In particular, description logics (DLs) have proved to be versatile formalisms with far-reaching applications like expressing knowledge on the Semantic Web; the *Ontology Web Language* (OWL), which builds on DLs, has fostered this development.

However, well-known results from the literature show that DLs have limitations: they do not allow for expressing default knowledge due to their inherent monotonic semantics. One needs nontrivial extensions to the first-order semantics of description logics to express exceptional knowledge.

Example 1. Take, as an example, a bird ontology expressed in the DL-KB $L = \{Flier \sqsubseteq \neg NonFlier,\ Penguin \sqsubseteq Bird, Penguin \sqsubseteq NonFlier, Bird(tweety)\}$. Intuitively, L distinguishes between flying and non-flying objects. We know that penguins, which are birds, do not fly. Nevertheless, we cannot simply add the axiom $Bird \sqsubseteq Flier$ to L to specify the common view that "*birds normally fly,*" as this update will make L inconsistent. From our bird ontology, we would like to conclude that Tweety flies; and if we learn that Tweety is a penguin, the opposite conclusion would be expected.

Hence, the simple ontology L from above cannot express exceptional knowledge. Default logic, a prominent formalism for expressing nonmonotonic knowledge in first-order logic, was introduced in the seminal work by Reiter [1]. To allow for nonmonotonicity in L, an extension of the semantics of terminological knowledge was given in [2], which is an early attempt to support default logic in the domain of description logics.

* This research has been supported by the Austrian Science Fund (FWF) project P20841 and P20840, and the EU research project IST-2009-231875 (ONTORULE).

C. Sossai and G. Chemello (Eds.): ECSQARU 2009, LNAI 5590, pp. 602–613, 2009.

Several other attempts to extend DLs with nonmonotonic features have been made, based on default logics [3,4], epistemic operators [5,6], circumscription [7,8], or argumentative approach [9]. They all showed that gaining both sufficient expressivity and good computational properties in a nonmonotonic DL is non-trivial.

However, reasoning engines for expressive nonmonotonic DLs are still not available (cf. Section 7). This forces users needing nonmonotonic features over DL-KBs to craft ad hoc implementations; systems for combining rules and ontologies (see [10]) might be helpful in that, but bear the danger that a non-standard semantics emerges not compliant with the user's intention. In fact, the less versatile a user is in KR formalisms, the higher is the likelihood that this will happen, even if just simple default rules of the form: *"If A is true and B can be consistently assumed, then conclude C,"* should be captured.

With the aim to offer a user-friendly reasoner over ontologies, we consider default reasoning on top of ontologies based on cq-programs [11], which integrate rules and ontologies. They slightly extend dl-programs [12] and allow for bidirectional communication between logic programs and DL-KBs by means of updates and powerful (unions of) conjunctive queries. Our main contributions are as follows.

- We consider a realization of Reiter-style default logic on top of DL-KBs, which amounts to a decidable fragment of Baader and Hollunder's terminological default logic [2], using cq-programs. A realization using dl-programs is discussed in [12], but is complex and more of theoretical interest than practical.
- We present two novel transformations of default theories into cq-programs, which are based on different principles and significantly improve over a similar transformation of default theories into dl-programs [12], both at the conceptual and the computational level. The former is apparent from the elegant formulation of the new transformations, while the latter is evidenced by experimental results. Furthermore, we present optimization methods by pruning rules, which are tailored specifically for the new translations.
- We describe a front-end as a new part of the dl-plugin [11] for the dlvhex engine implementing cq-programs. In this front-end, users can specify input comprising default rules in a text file and an ontology in an OWL file, run a command and get (descriptions of) the extensions of the default theory. Importantly, expert users in logic programming (LP) can exploit the front-end also at lower levels and customize the transformations, by adding further rules and constraints. In this way, alternative or refined semantics (e.g., preferences) may be realized, or the search space pruned more effectively.

Our front-end approach provides a simple and intuitive way to encode default knowledge on top of terminological KBs, relieving users from developing ad hoc implementations (which is rather difficult and error-prone). Furthermore, besides the benefit that special constructs of logic programs like weak constraints or aggregates can be utilized in customizations, the dlvhex implementation also offers the possibility to combine DL-KBs with other knowledge sources like, e.g., RDF KBs on a solid theoretical basis.

2 Preliminaries

Description Logics. We assume familiarity with DLs (cf. [13]), in particular any extension of $\mathcal{ALC}$ which can be recast to first-order logic w.r.t. CQ-answering (conceptually,

we can apply Reiter-style default logic on top of any such DLs).[1] A *DL-KB* L is a finite set of axioms (TBox) and factual assertions (ABox) α in the respective DL, which are formed using disjoint alphabets of atomic concepts, roles, and individuals, respectively. By $L \models \alpha$ we denote logical consequence of an axiom resp. assertion α from L.

Default Logic. In this paper, we restrict Reiter's default logic [1] and consider only conjunctions of literals in default rules. We adjust default theories in a way such that the background theory, given by a DL-KB L, represents an ontology.

A *default* δ (*over* L) has the form $\frac{\alpha(\boldsymbol{X}):\beta_1(\boldsymbol{Y_1}),\ldots,\beta_m(\boldsymbol{Y_m})}{\gamma(\boldsymbol{Z})}$, where $\alpha(\boldsymbol{X}) = \alpha_1(\boldsymbol{X_1}) \wedge \cdots \wedge \alpha_k(\boldsymbol{X_k})$, $\beta_i(\boldsymbol{Y_i}) = \beta_{i,1}(\boldsymbol{Y_{i,1}}) \wedge \cdots \wedge \beta_{i,\ell_i}(\boldsymbol{Y_{i,\ell_i}})$, $\gamma(\boldsymbol{Z}) = \gamma_1(\boldsymbol{Z_1}) \wedge \cdots \wedge \gamma_n(\boldsymbol{Z_n})$, and $\boldsymbol{X}$, $\boldsymbol{Y_i}$, and $\boldsymbol{Z}$ are all variables occurring in the respective conjuncts. We allow c to be c^* or $\neg c^*$ for every $c \in \{\alpha_i, \beta_{i,j}, \gamma_i\}$, where each $\alpha_i^*, \beta_{i,j}^*, \gamma_i^*$ is either an atomic concept or role name in L. A *default theory over* L is a pair $\Delta = \langle L, D \rangle$, where L is a DL-KB and D is a finite set of defaults over L.

Example 2. Consider the DL-KB L in Ex. 1 and $D = \left\{ \frac{Bird(X):Flier(X)}{Flier(X)} \right\}$, then $\Delta = \langle L, D \rangle$ is a default theory over L.

Given that L is convertible into an equivalent first-order formula $\pi(L)$, we can view Δ as a *Reiter-default theory* $T = \langle W, D \rangle$ over a first-order language $\mathcal{L}$, where $W = \{ \pi(L) \}$, and apply concepts from [1] to Δ; we recall some of them in the sequel.

The semantics of $T = \langle W, D \rangle$ is given in terms of its *extensions*, which we recall next; intuitively, they are built by applying defaults in D as much as possible to augment the definite knowledge in W with plausible conclusions.

Suppose $T = \langle W, D \rangle$ is *closed* (i.e., defaults have no free variables). Then for any set of sentences $S \subseteq \mathcal{L}$, let $\Gamma_T(S)$ be the smallest set of sentences from $\mathcal{L}$ such that (i) $W \subseteq \Gamma_T(S)$; (ii) $\Gamma_T(S)$ is deductively closed, i.e., $Cn(\Gamma_T(S)) = \Gamma_T(S)$; and (iii) if $\frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma} \in D$, $\alpha \in \Gamma_T(S)$, and $\neg\beta_1, \ldots, \neg\beta_m \notin S$ then $\gamma \in \Gamma_T(S)$. Here, $\vdash$ denotes classical derivability and $Cn(\mathcal{F}) = \{\phi \mid \mathcal{F} \vdash \phi \text{ and } \phi \text{ is closed}\}$, for every set $\mathcal{F}$ of closed formulas. For an arbitrary $T = \langle W, D \rangle$, Γ_T is applied to its closed version $cl(T)$, i.e., each default in D is replaced by all its grounded instances w.r.t. $\mathcal{L}$. Then, a set of sentences $E \subseteq \mathcal{L}$ is an extension of T, iff $E = \Gamma_T(E)$ [1].

cq-Programs. Informally, a cq-program consists of a DL-KB L and a disjunctive program P that may involve queries to L. Roughly, such a query may ask whether a specific conjunction of atoms or union of such conjunctions is entailed by L or not.

Syntax. A *conjunctive query* (CQ) $q(\boldsymbol{X})$ is an expression $\{\boldsymbol{X} | Q_1(\boldsymbol{X_1}), \ldots, Q_n(\boldsymbol{X_n})\}$, where each Q_i is a concept or role expression and each $\boldsymbol{X_i}$ is a list of variables and individuals of matching arity; $\boldsymbol{X} \subseteq Vars(\boldsymbol{X_1}, \ldots, \boldsymbol{X_n})$ are its *distinguished* (or *output*) variables, where $Vars(\boldsymbol{X_1}, \ldots, \boldsymbol{X_n})$ is the set of variables appearing in $\boldsymbol{X_1}, \ldots, \boldsymbol{X_n}$. Intuitively, $q(\boldsymbol{X})$ is a conjunction $Q_1(\boldsymbol{X_1}) \wedge \cdots \wedge Q_n(\boldsymbol{X_n})$ of concept and role expressions, which is true if all conjuncts are satisfied, and then projected on $\boldsymbol{X}$.

A *union of conjunctive queries* (UCQ) $q(\boldsymbol{X})$ is a disjunction $\bigvee_{i=1}^{m} q_i(\boldsymbol{X})$ of CQs $q_i(\boldsymbol{X})$. Intuitively, $q(\boldsymbol{X})$ is satisfied, whenever some $q_i(\boldsymbol{X})$ is satisfied.

[1] This includes inverse roles ($\mathcal{I}$), qualified number restrictions ($\mathcal{Q}$), nominals ($\mathcal{O}$), and role hierarchy ($\mathcal{H}$), where a DL-KB L is convertible into an equivalent first-order formula $\pi(L)$ [13]; role transitivity ($\mathcal{S}$) may occur in L as well, but is disallowed in defaults.

A *cq-atom* α is of form $\mathrm{DL}[\lambda; q(\boldsymbol{X})](\boldsymbol{X})$, where $\lambda = S_1\ op_1\ p_1, \ldots, S_m\ op_m\ p_m$ ($m \geq 0$) is an (input) list of expressions $S_i\ op_i\ p_i$, each S_i is either a concept or a role name, $op_i \in \{\uplus, \cup\}$, p_i is an (input) predicate symbol matching the arity of S_i, and $q(\boldsymbol{X})$ is a (U)CQ. Intuitively, $op_i = \uplus$ increases S_i by the extension of p_i, while $op_i = \cup$ increases $\neg S_i$. If $m = 1$, α amounts to a *dl-atom* $\mathrm{DL}[\lambda; Q](\boldsymbol{t})$ as in [12] where $\boldsymbol{X} = \mathit{Vars}(\boldsymbol{t})$.

A *literal* l is an atom p or a negated atom $\neg p$. A *cq-rule* r is an expression of the form $a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \mathrm{not}\ b_{m+1}, \ldots, \mathrm{not}\ b_n$, where every a_i is a literal and every b_j is either a literal or a cq-atom. We define $H(r) = \{a_1, \ldots, a_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*.

A *cq-program* $KB = (L, P)$ consists of a DL-KB L and a finite set of cq-rules P.

Example 3. Let L be from Ex. 1 and $P = \{\mathit{flies}(\mathit{tweety}) \vee \mathit{nflies}(\mathit{tweety});\ \mathit{bird}(X) \leftarrow \mathrm{DL}[\mathit{Flier} \uplus \mathit{flies}; \mathit{Flier}(X) \vee \mathit{NonFlier}(X)](X)\}$. Then, $KB = (L, P)$ is a cq-program. The body of the rule defining *bird* is a cq-atom with the UCQ $q(X) = \mathit{Flier}(X) \vee \mathit{NonFlier}(X)$ and an input list $\lambda = \mathit{Flier} \uplus \mathit{flies}$. In this cq-atom we update the concept Flier in L with the extension of *flies* before asking for the answers of $q(X)$.

Semantics. Given a cq-program KB = (L, P), the *Herbrand base* of P, denoted HB_P, is the set of all ground literals with predicate symbols in P and constant symbols in a (predefined) set $\mathcal{C}$. An *interpretation* I relative to P is a consistent subset of HB_P. We say I is a *model* of $l \in HB_P$ under L, or I *satisfies* l under L, denoted $I \models_L l$, if $l \in I$.

For any CQ $q(\boldsymbol{X}) = \{\boldsymbol{X} \mid Q_1(\boldsymbol{X_1}), \ldots, Q_n(\boldsymbol{X_n})\}$, let $\phi_q(\boldsymbol{X}) = \exists \boldsymbol{Y} \bigwedge_{i=1}^{n} Q_i(\boldsymbol{X_i})$, where $\boldsymbol{Y}$ are the variables not in $\boldsymbol{X}$, and for any UCQ $q(\boldsymbol{X}) = \bigvee_{i=1}^{m} q_i(\boldsymbol{X})$, let $\phi_q(\boldsymbol{X}) = \bigvee_{i=1}^{m} \phi_{q_i}(\boldsymbol{X})$. Then, for any (U)CQ $q(\boldsymbol{X})$, the set of *answers of* $q(\boldsymbol{X})$ *on* L is the set of tuples $\mathit{ans}(q(\boldsymbol{X}), L) = \{\boldsymbol{c} \in \mathcal{C}^{|\boldsymbol{X}|} \mid L \models \phi_q(\boldsymbol{c})\}$.

An interpretation I *satisfies* a ground instance $a(\boldsymbol{c})$ of $a(\boldsymbol{X}) = \mathrm{DL}[\lambda; q(\boldsymbol{X})](\boldsymbol{X})$ (i.e., all variables in $q(\boldsymbol{X})$ are replaced by constant symbols from $\mathcal{C}$), denoted $I \models_L a(\boldsymbol{c})$, if $\boldsymbol{c} \in \mathit{ans}(q(\boldsymbol{X}), L \cup \lambda(I))$, where $\lambda(I) = \bigcup_{i=1}^{m} A_i(I)$ and (i) $A_i(I) = \{S_i(\boldsymbol{e}) \mid p_i(\boldsymbol{e}) \in I\}$, for $op_i = \uplus$, and (ii) $A_i(I) = \{\neg S_i(\boldsymbol{e}) \mid p_i(\boldsymbol{e}) \in I\}$, for $op_i = \cup$.

I satisfies a ground cq-rule r, denoted $I \models_L r$, if $I \models_L H(r)$ whenever $I \models_L B(r)$, where $I \models_L H(r)$ if $I \models_L a$ for some $a \in H(r)$, and $I \models_L B(r)$ if $I \models_L a$ for all $a \in B^+(r)$ and $I \not\models_L a$ for all $a \in B^-(r)$.

I is a *model* of (or *satisfies*) a cq-program $KB = (L, P)$, denoted $I \models KB$, if $I \models_L r$ for all $r \in \mathit{ground}(P)$. The (strong) answer sets of KB, which amount to particular models of KB, are then defined like answer sets of an ordinary disjunctive logic program using the Gelfond-Lifschitz reduct P^I of P w.r.t. I, where cq-atoms are treated like ordinary atoms; I is then a *(strong) answer set* of KB, if I is a minimal model (w.r.t. set inclusion) of (L, P^I) (cf. also [12]).

Example 4 (cont'd). The strong answer sets of KB in Ex. 3 are $M_1 = \{\mathit{flies}(\mathit{tweety}), \mathit{bird}(\mathit{tweety})\}$ and $M_2 = \{\mathit{nflies}(\mathit{tweety})\}$. The answer set M_1 updates L in such a way that we can infer $q(\mathit{tweety})$ from $L \cup \lambda(M_1)$, thus $\mathit{bird}(\mathit{tweety}) \in M_1$, whereas in M_2, $L \cup \lambda(M_2) \not\models q(\mathit{tweety})$, and so $\mathit{bird}(\mathit{tweety}) \notin M_2$.

3 Transformations from Default Theories to cq-Programs

In the sequel, assume that we have a default theory $\Delta = \langle L, D \rangle$ over L.

We first revisit the transformation in [12], which we call Π. For each default of form $\frac{\alpha(\boldsymbol{X}):\beta_1(\boldsymbol{Y_1}),\ldots,\beta_m(\boldsymbol{Y_m})}{\gamma(\boldsymbol{Z})}$ (where the β_i and γ are just literals), it uses the following rules:

$$in\_\gamma(\boldsymbol{Z}) \leftarrow \text{not } out\_\gamma(\boldsymbol{Z}); \qquad out\_\gamma(\boldsymbol{Z}) \leftarrow \text{not } in\_\gamma(\boldsymbol{Z}) \tag{1}$$

$$g(\boldsymbol{Z}) \leftarrow \mathrm{DL}[\lambda;\alpha_1](\boldsymbol{X_1}),\ldots,\mathrm{DL}[\lambda;\alpha_k](\boldsymbol{X_k}), \text{not } \mathrm{DL}[\lambda';\neg\beta_1](\boldsymbol{Y_1}),\ldots,\text{not } \mathrm{DL}[\lambda';\neg\beta_m](\boldsymbol{Y_m}) \tag{2}$$

$$fail \leftarrow \mathrm{DL}[\lambda';\gamma](\boldsymbol{Z}), out\_\gamma(\boldsymbol{Z}), \text{not } fail \tag{3}$$

$$fail \leftarrow \text{not } \mathrm{DL}[\lambda;\gamma](\boldsymbol{Z}), in\_\gamma(\boldsymbol{Z}), \text{not } fail \tag{4}$$

$$fail \leftarrow \mathrm{DL}[\lambda;\gamma](\boldsymbol{Z}), out\_\gamma(\boldsymbol{Z}), \text{not } fail \tag{5}$$

where λ' contains for each default δ an update $\gamma^* \uplus in\_\gamma$ if $\gamma(\boldsymbol{Z})$ is positive, and an update $\gamma^* \uplus in\_\gamma$ if $\gamma(\boldsymbol{Z})$ is negative; λ is similar with g in place of in_γ.

Π is based on a guess-and-check approach: the rules (1) guess whether the conclusion $\gamma(\boldsymbol{Z})$ belongs to an extension E or not. If yes, a ground atom with auxiliary predicate in_γ, which is used in the input list λ' to update L, is inferred. Intuitively, $L \cup \lambda'(I)$ represents E. Next, the rule (2) imitates the Γ_Δ operator to compute $\Gamma_\Delta(E)$. The outcome is stored in an auxiliary predicate g, which is used in a second input list λ to update L (independent from λ'); intuitively, $L \cup \lambda(I)$ represents $\Gamma_\Delta(E)$, Finally, the rule (3) checks whether the guess for E is compliant with L, and the rules (4) and (5) check whether E and $\Gamma_\Delta(E)$ coincide. If this is the case, then E is an extension of Δ.

A natural question is whether we can have a simpler transformation; in particular, with fewer and more homogeneous cq-atoms, in the sense that the update lists are similar; this would help to reduce communication between the rules and L, such that the evaluation of the transformation is more effective.

We give a positive answer to this question and present two novel transformations, called Ω and Υ, which are based on different ways of computing extensions, inspired by algorithms *select-default-and-check* and *select-justification-and-check* that were earlier mentioned in [14]. In fact, in both of them a *single* input list λ is sufficient for *all* cq-atoms. Furthermore, by the use of UCQs, we can easily handle also defaults with conjunctive justifications and conclusions.

The transformations Ω and Υ are compactly presented in Table 1, where we use the following notation. Given a default $\frac{\alpha(\boldsymbol{X}):\beta_1(\boldsymbol{Y_1}),\ldots,\beta_m(\boldsymbol{Y_m})}{\gamma(\boldsymbol{Z})}$, for $\Psi \in \{in, cons, \overline{cons}\}$ and $e(\boldsymbol{W}) \in \{\gamma(\boldsymbol{Z}), \gamma_i^*(\boldsymbol{Z_i}), \beta_i(\boldsymbol{Y_i}), \beta_{i,j}^*(\boldsymbol{Y_{i,j}})\}$, we use $\Psi(e(\boldsymbol{W}))$ to denote $\Psi_e(\boldsymbol{W})$ (where Ψ_e is a predicate name).

Transformation $\boldsymbol{\Omega}$. The main idea of Ω is to use only one update λ instead of both λ and λ' in Π, hence only one type of auxiliary predicates is needed, namely $in(\gamma(\boldsymbol{X}))$.

This transformation is quite intuitive and follows exactly the usual way of evaluating extensions in default theories: *"If the prerequisites can be derived, and the justifications can be consistently assumed, then the conclusion can be concluded."*

Intuitively, in the rule with head $in(\gamma(\boldsymbol{X}))$, we apply the Γ_Δ operator to find out whether the whole consequent $\gamma(\boldsymbol{Z})$ is in the extension E or not. If this is the case, then each $\gamma_i(\boldsymbol{Z_i})$ in $\gamma(\boldsymbol{Z})$ will also be concluded to be in E by rules in $\mathcal{R}$. In order to check

Table 1. Transformations Ω/Υ of default theory Δ to cq-program $KB_\Omega(\Delta)/KB_\Upsilon(\Delta)$

For $\Delta = \langle L, D\rangle$ and $X \in \{\Omega, \Upsilon\}$, let $KB_X(\Delta) = (L, P_X)$, where $P_X = \bigcup_{\delta\in D} X(\delta)$ and

$$\Omega(\delta) = \mathcal{R} \cup \{ in(\gamma(\boldsymbol{Z})) \leftarrow \mathrm{DL}[\lambda; \alpha(\boldsymbol{X})](\boldsymbol{X}), \\ \text{not } \mathrm{DL}[\lambda; d(\beta_1(\boldsymbol{Y_1}))](\boldsymbol{Y_1}), \ldots, \text{not } \mathrm{DL}[\lambda; d(\beta_m(\boldsymbol{Y_m}))](\boldsymbol{Y_m}) \}$$

$$\Upsilon(\delta) = \mathcal{R} \cup \{ in(\gamma(\boldsymbol{Z})) \leftarrow \mathrm{DL}[\lambda; \alpha(\boldsymbol{X})](\boldsymbol{X}), cons(\beta_1(\boldsymbol{Y_1})), \ldots, cons(\beta_m(\boldsymbol{Y_m}))\} \cup \\ \left\{ \begin{array}{ll} fail \leftarrow cons(\beta_i(\boldsymbol{Y_i})), \mathrm{DL}[\lambda; d(\beta_i(\boldsymbol{Y_i}))](\boldsymbol{Y_i}), \text{not } fail; & \\ fail \leftarrow \overline{cons}(\beta_i(\boldsymbol{Y_i})), \text{not } \mathrm{DL}[\lambda; d(\beta_i(\boldsymbol{Y_i}))](\boldsymbol{Y_i}), \text{not } fail; & \\ cons(\beta_i(\boldsymbol{Y_i})) \leftarrow \text{not } \overline{cons}(\beta_i(\boldsymbol{Y_i})); & \\ \overline{cons}(\beta_i(\boldsymbol{Y_i})) \leftarrow \text{not } cons(\beta_i(\boldsymbol{Y_i})) & \mid 1 \le i \le m \end{array} \right\}$$

where $\lambda = (\gamma_i^* \uplus in_{\gamma_i}, \gamma_i^* \uplus in_{\neg\gamma_i} \mid \delta \in D)$, $d(\beta_i(\boldsymbol{Y_i})) = \neg\beta_{i,1}(\boldsymbol{Y_{i,1}}) \vee \cdots \vee \neg\beta_{i,\ell_i}(\boldsymbol{Y_{i,\ell_i}})$, and $\mathcal{R} = \{in(\gamma_i(\boldsymbol{Z_i})) \leftarrow in(\gamma(\boldsymbol{Z})) \mid 1 \le i \le n\}$.

the satisfaction of the prerequisite, we use a CQ, while consistency of a justification is checked by a UCQ. In case the prerequisite or a justification is just a literal, the query amounts to instance checking (which is more efficient).

Example 5. Consider default theory Δ in Ex. 2. Since the prerequisite, justification and conclusion of the default in D are just literals, $\mathcal{R}$ can be simplified to $\emptyset$ and the cq-atoms to instance checks. Therefore, P_Ω consists only of the rule

$$in_{Flier}(X) \leftarrow \mathrm{DL}[\lambda; Bird](X), \text{not } \mathrm{DL}[\lambda; \neg Flier](X) \ ,$$

where $\lambda = Flier \uplus in_{Flier}, Flier \uplus in_{\neg Flier}$. The single answer set of $KB_\Omega(L, D)$ is $I_\Omega = \{in_{Flier}(tweety)\}$ which corresponds to the single extension.

Transformation Υ. In this transformation, we make use of the *Select-justifications-and-check* algorithm. The definition of $\Upsilon(\delta)$ in Table 1 is explained as follows. The first rule emulates the Γ_Δ operator to find the set of consequences under a consistency assumption for the default justifications $\beta_i(\boldsymbol{Y_i})$ with the extension E; like above, with $\gamma(\boldsymbol{Z})$ also each $\gamma_i(\boldsymbol{Z_i})$ is concluded by the rules in $\mathcal{R}$.

The assumptions for all justifications $\beta_i(\boldsymbol{Y_i})$ are guessed with the last two rules, and they are checked with two constraints: the first prevents cases in which we guess that $\beta_i(\boldsymbol{Y_i})$ is consistent with E but we can in fact derive $\neg\beta_i(\boldsymbol{Y_i})$. Similarly, the second constraint eliminates all models in which $\beta_i(\boldsymbol{Y_i})$ is guessed to be inconsistent with E but we cannot derive its negation.

We can see that transformation Υ involves less communication with the DL-KB than Ω; instead, it has explicit guessing on the logic program side. If the number of justifications is small, we may expect better performance.

Example 6. For the default theory Δ in Ex. 2, P_Υ consists of the following rules:

$$cons_{Flier}(X) \leftarrow \text{not } \overline{cons}_{Flier}(X); \qquad \overline{cons}_{Flier}(X) \leftarrow \text{not } cons_{Flier}(X)$$
$$in_{Flier}(X) \leftarrow \mathrm{DL}[\lambda; Bird](X), cons_{Flier}(X)$$
$$fail \leftarrow cons_{Flier}(X), \mathrm{DL}[\lambda; \neg Flier](X), \text{not } fail$$
$$fail \leftarrow \overline{cons}_{Flier}(X), \text{not } \mathrm{DL}[\lambda; \neg Flier](X), \text{not } fail$$

where $\lambda = Flier \uplus in_{Flier}, Flier \uplus in_{\neg Flier}$. The single answer set of $KB_\Upsilon(L, D)$ is $I_\Upsilon = \{in_{Flier}(tweety), cons_{Flier}(tweety)\}$ which corresponds to the single extension.

The following theorem shows the correctness of our transformations.

Theorem 1. *Let $\Delta = \langle L, D\rangle$ be a default theory over L, and $X \in \{\Omega, \Upsilon\}$. Then:*

(i) For each extension E of Δ, there exists a (unique) strong answer set M of $KB_X(\Delta)$, such that $E = Cn(L \cup \lambda(M))$.
(ii) For each strong answer set M of $KB_X(\Delta)$, the set $E = Cn(L \cup \lambda(M))$ is an extension of Δ.

Note that in general, answering UCQs over expressive DL-KBs may be undecidable; in our case variables range effectively only over known individuals in the KB (i.e., constants in $\mathcal{C}$). We also mention that the further transformation of $KB_X(\Delta)$ into HEX-programs [15] for execution in dlvhex requires rules to be domain-expansion safe; this is easily accomplished by introducing a domain predicate dom, adding to the body of each rule for each variable Y the atom $dom(Y)$, and appending a fact $dom(a)$ to P_X for each individual a in the KB (see [15] for details).

4 Optimization

This section introduces pruning rules to reduce the search space in model computation. In what follows, we consider defaults δ_1 and δ_2, where

$$\delta_i = \frac{\alpha_{i,1}(\boldsymbol{X_{i,1}}) \wedge \cdots \wedge \alpha_{i,k_i}(\boldsymbol{X_{i,k_i}}) : \beta_{i,1}(\boldsymbol{Y_{i,1}}), \ldots, \beta_{i,m_i}(\boldsymbol{Y_{i,m_i}})}{\gamma_{i,1}(\boldsymbol{Z_{i,1}}) \wedge \cdots \wedge \gamma_{i,n_i}(\boldsymbol{Z_{i,n_i}})} \ .$$

Based on the interaction of δ_1 and δ_2, we can add the following rules. Let $\gamma_i(\boldsymbol{Z_i})$ be short for $\gamma_{i,1}(\boldsymbol{Z_{i,1}}) \wedge \cdots \wedge \gamma_{i,n_i}(\boldsymbol{Z_{i,n_i}})$, where $\boldsymbol{Z}_i = \bigcup_{1 \le j_i \le n_i} \boldsymbol{Z_{i,j_i}}$, for $i = 1, 2$.

Forcing other defaults to be out. The well-known Nixon Diamond example motivates a shortcut in dealing with defaults whose conclusions are opposite. In this example, the conclusion of one default blocks the other. To prune such cases, we can add

$$fail \leftarrow in(\gamma_1(\boldsymbol{Z_1})), in(\gamma_2(\boldsymbol{Z_2})), \text{not } fail \tag{6}$$

to P_X, where $X \in \{\Omega, \Upsilon\}$, whenever there exist $1 \le j_1 \le n_1$, $1 \le j_2 \le n_2$ s.t. $\gamma_{1,j_1}(\boldsymbol{Z_{1,j_1}})$ and $\neg\gamma_{2,j_2}(\boldsymbol{Z_{2,j_2}})$ are unifiable.

Furthermore, also the relations between conclusions and justifications can be exploited for pruning purpose. If there exist $j \in \{1, \ldots, n_1\}$ and $j' \in \{1, \ldots, m_2\}$ such that $\gamma_{1,j}(\boldsymbol{Z_{1,j}})$ is unifiable with a disjunct in $\neg\beta_{2,j'}(\boldsymbol{Y_{2,j'}})$, then the conclusion $\gamma_1(\boldsymbol{Z_1})$ of δ_1 will block the application of δ_2 and the constraint (6) can also be added to P_X.

Forcing other defaults to be in. If $\gamma_1(\boldsymbol{Z_1})$ is part of $\gamma_2(\boldsymbol{Z_2})$, then adding an instance of $\gamma_2(\boldsymbol{Z_2})$ to an extension E requires also to add the respective instance of $\gamma_1(\boldsymbol{Z_1})$ to E. Thus, if for every $j_1 \in \{1, \ldots, n_1\}$, $\gamma_{1,j_1}(\boldsymbol{Z_{1,j_1}})$ is unifiable with $\gamma_{2,j_2}(\boldsymbol{Z_{i,j_2}})$ for some $j_2 \in \{1, \ldots, n_2\}$, then we add the following rule to P_X, where $X \in \{\Omega, \Upsilon\}$:

$$in(\gamma_1(\boldsymbol{Z_1})) \leftarrow in(\gamma_2(\boldsymbol{Z_2})) \ .$$

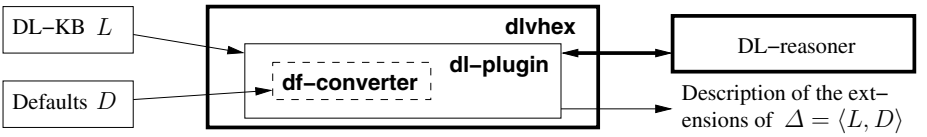


Fig. 1. Architecture of the front-end

Defaults whose conclusions are already in the background theory. Each extension contains all consequences of the background theory W of the default theory ($Cn(W) \subseteq E$). Hence, it is worth testing whether for a default $\frac{\alpha(\boldsymbol{X}):\beta_1(\boldsymbol{Y_1}),\ldots,\beta_m(\boldsymbol{Y_m})}{\gamma(\boldsymbol{Z})}$ a conjunct $\gamma_i(\boldsymbol{X_i}) \in \gamma(\boldsymbol{X})$ can be concluded from the DL-KB before the guessing phase or the application of the Γ_Δ operator. To this end, we can add to $P_X(X \in \{\Omega, \Upsilon\})$ the rule

$$in(\gamma_i(\boldsymbol{X_i})) \leftarrow \mathrm{DL}[\gamma_i](\boldsymbol{X_i}) \ .$$

5 Implementation

We have implemented the transformations from Section 3 in a framework that provides the user with a front-end to cq-programs for transparent default reasoning over ontologies. An architectural overview of this front-end is shown in Fig. 1.

The implementation makes use of the dlvhex environment,[2] a framework for solving HEX-programs. It has a plugin facility that allows to define external atoms, and dl-plugin is one of the plugins deployed in this environment. The dl-plugin provides a mechanism for converting cq-programs to HEX-programs. It receives a cq-program or a HEX-program together with an OWL ontology, communicates with a DL-reasoner to evaluate queries in the program, and dlvhex processes the query answers and generates models of the program. Based on this framework, we implemented a converter for default rules on top of description logics, df-converter, as a pre-processor in the dl-plugin which takes a set of defaults and an OWL ontology as input, converts this input into cq-rules according to a transformation, and transfers the result to the dl-plugin; dlvhex then does the rest. Hence, all the complications including cq-programs, HEX-programs, and the transformations are transparent to the users. They just need to specify defaults in a simple format and get (descriptions of) the extensions of the input default theory, which were modified from the models of the transformed HEX-programs.

The grammar for the syntax of input defaults is as follows:

`lit` ::= `atom` | −`atom`
`conjunction` ::= `lit` (& `lit`)∗
`default` ::= [$\underbrace{\texttt{conjunction}}_{\text{prerequisite}}$; $\underbrace{\texttt{conjunction}\,(\,,\texttt{conjunction})*}_{\text{justifications}}$] / [$\underbrace{\texttt{conjunction}}_{\text{conclusion}}$]

Here, '−' is classical negation, '&' is conjunction, and '`atom`' is an atomic formula with a concept or role.

As for the output, the interesting information about an extension E is the default conclusions that are in E. To this end, we filter all ground literals from the answer sets

[2] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

belonging to the default conclusions derived in the program for the user (reasoning tasks can be easily realized on top by customization).

The following example illustrates this elegant interface.

Example 7. For the Bird example, the input includes an OWL file for the ontology L in Ex. 1 and a text file for D in Ex. 2, whose content simply is

```
[ Bird(X); Flier(X) ] / [ Flier(X) ]
```

We can now invoke dlvhex to ask for the extensions. We get only one extension in this particular case, and the only fact returned to users is `Flier(tweety).`

However, users are not confined to simple defaults. Expert users in LP can provide more sophisticated pruning rules, or rules that select specific extensions (e.g., under preference) in terms of cq- or HEX-rules. They are directly sent to the dl-plugin and added to cq-rules supplied by the df-converter as an input for dlvhex. Our front-end therefore is flexible and can meet requirements of different classes of users.

Typing predicates. The front end supports explicit *typing predicates* as a means to control the instantiation of defaults and to limit the search space in advance. For example, in a predicate $hasScholarship(P, S)$ the first argument should be a *Student* while the second should be a *Scholarship*.

Users can attach to each default δ a *type guard* $\theta(\boldsymbol{W})$ whose variables $\boldsymbol{W}$ appear in δ, and list all facts for the predicate θ, or even write a program which computes them. Semantically, $\theta(\boldsymbol{W})$ is added to the precondition of δ and all facts $\theta(\boldsymbol{c})$ are added to the background knowledge of the default theory. If a rule r in a transformation of δ satisfies $\boldsymbol{W} \subseteq Vars(r)$, then each atom $dom(X)$ in it with $X \in \boldsymbol{W}$ is replaced by $\theta(\boldsymbol{W})$.

Example 8 (cont'd). Suppose we have many instances of the concept $Bird$ in L, but just want to know whether `tweety` and `joe` fly or not. We can modify the input to

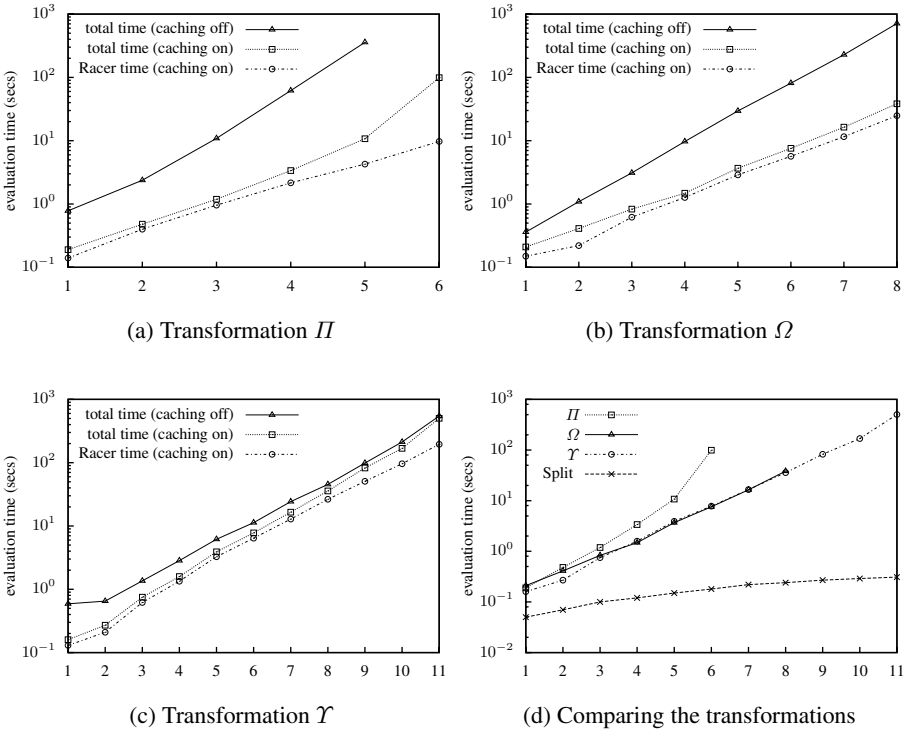
```
[ Bird(X); Flier(X) ] / [ Flier(X) ]<mb(X)>
```

and add facts `mb(tweety),mb(joe)` to specify these two birds.

We remark that in general, adding typing predicates makes the transformations incomplete w.r.t. to the original theory. However, for so called *semi-monotonic* default theories (where the extensions increase with an increasing set of defaults; e.g., the important *normal* default theories [1] have this property) credulous conclusions are sound, as well as skeptical conclusions if a unique extension is guaranteed.

6 Experimental Results

We have tested the transformations Π, Ω, and Υ using the prototype implementation of the front-end as a new component in the dl-plugin for dlvhex, which uses RacerPro 1.9.2Beta [16] as DL-reasoner, to explore different aspects which can influence the overall system performance, namely (i) the number of cq-atoms in each transformation, (ii) the number of individuals (size of the ABox), (iii) size of the TBox, and (iv) the query caching to the DL-reasoner. The tests were done on a P4 1.8GHz PC with 1GB RAM under Linux. We report here only the outcome of the Bird benchmark, which consists of a set of ontologies in spirit of Ex. 1 with increasing size of $Bird$ instances.



(a) Transformation Π (b) Transformation Ω

(c) Transformation Υ (d) Comparing the transformations

Fig. 2. Bird example – running time (x-axis: number of individuals)

Fig. 2 shows experimental results of this test, including running time of each transformation Π, Ω, Υ and the comparison between them when query caching to RacerPro is applied. Missing entries mean time exhaustion during evaluation. For each transformation, we are interested in the total running time and time that RacerPro consumes when caching is on. When caching is off, the difference between the total running time and time used by RacerPro is insignificant, hence only the total running time is displayed.

The experimental results reveal that Ω and Υ are much faster than Π, since Ω has fewer guessing rules and Υ has fewer cq-atoms, but Υ has a trade-off between consistency guessing rules and cq-atoms in rules that compute extensions. Hence, the performance of Ω and Υ depends very much on a particular set of defaults.

Regarding (i), the number of cq-atoms is important as they make up the search space which increases exponentially. Regarding (ii), also the number of individuals is clearly important (as it directly influences (i)), while for (iii), the size of the TBox was of minor concern. When increasing the TBoxes (by enlarging the taxonomies systematically), the performance was not much affected. This is not much surprising, as DL engines like RacerPro are geared towards efficient reasoning with (large) TBoxes (of course, one could have used "hard" TBoxes, but this seems less relevant from a practical perspective). Regarding (iv), it appeared that query caching plays an important role, as the system spends most of the time querying RacerPro for all ground cq-atoms.

To undercut the impact of (ii) on (i), a new version of dlvhex has been developed in which independence information about different cq-atoms, which is based on the individuals occurring in them, can be exploited to factorize a HEX-program into components that can be evaluated in parallel (see [17]). In particular, in the Bird benchmark, for each individual a separate component will be created; sequentially processed, this yields linear scalability with respect to (ii) (see Fig. 2d). Currently, the new dlvhex version is beneficial only to transformation Ω, but further improvements are expected for Π and Υ.

7 Related Work and Conclusions

As we already mentioned, this work is not the first considering default reasoning with description logics. Earlier ones [2,3,4] posed varying restrictions on the terminological representation to keep decidability, and provided no implementations or only for limited forms of defaults (on concepts). As our approach is theoretically based on a strict separation between rules and ontologies, it guarantees decidability as long as the DL-KB is convertible into a decidable first-order theory w.r.t. CQ-answering. Moreover, on the practical side, we provide a concrete implementation for Reiter-style default logic via a front-end hosted by a framework to combine rules and ontologies.

Hybrid Default Inheritance Theory (HDIT) [3] allows to specify defaults of form $\frac{A(X):C(X)}{C(X)}$ and $\frac{A(X)\wedge R(X,Y):C(Y)}{C(Y)}$. To retain decidability, concepts in HDIT must be conjunctions of (negated) atomic concepts; [2] allows only DLs from $\mathcal{ALC}$ to $\mathcal{ALCF}$. An implementation of the *DTL* language is reported in [4], but roles cannot be defined.

The DL $\mathcal{ALCK}$ [6] adds an epistemic operator $\mathbf{K}$ to $\mathcal{ALC}$ which allows to specify closed-world reasoning in queries. The later $\mathcal{DLK}_{\mathcal{NF}}$ [5] can be regarded as an extension of $\mathcal{ALCK}$ with epistemic operator $\mathbf{A}$ expressing "default assumption." Defaults can only be specified over concept expressions and are translated to TBox axioms using $\mathbf{K}$ for prerequisites and conclusions, and $\neg\mathbf{A}$ for negated justifications.

In [7], circumscriptive (minimal) models have been used to define the *Extended Closed World Assumption (ECWA) over hybrid systems* with a proof theory based on a translation to propositional theories. However, hybrid systems are actually a fragment of $\mathcal{ALE}$ and not very expressive. A recent paper [8] proposed extensions of expressive DLs $\mathcal{ALCIO}$ and $\mathcal{ALCQO}$ to form *circumscribed knowledge bases (cKBs)*. They can avoid the restriction of nonmonotonic reasoning to named individuals in the domain, but still allow only that concept names can be circumscribed.

Recently, [9] uses an argumentative approach for reasoning with inconsistent ontologies by translating DL ontologies into defeasible logic programs. However, only inconsistent concept definitions were considered. Concerning semantics, this approach is different from ours since it uses the notion of defeasible derivation which corresponds to SLD-derivation in logic programming.

Concerning further work, the experimental comparison of the transformations Π, Ω and Υ revealed several tasks that can help to improve performance. One is to investigate more refined pruning rules that depend on the structure of the default theory. Another issue is to look into particular kinds of default theories, such as normal or semi-normal default theories, for which more efficient transformations may be found. At the bottom level, we note that caching for cq-atoms (which is currently only available for plain dl-atoms) would give additional benefit. Furthermore, dlvhex is currently using RacerPro

as its (only) DL-reasoner; it would be interesting to have support for other DL-reasoners such as KAON2 or Pellet, and compare the results. Finally, improvement of dlvhex evaluation at the general level (e.g., by refined dependency handling) would be beneficial.

References

1. Reiter, R.: A logic for default reasoning. Artif. Intell. 13(1-2), 81–132 (1980)
2. Baader, F., Hollunder, B.: Embedding Defaults into Terminological Knowledge Representation Formalisms. J. Autom. Reasoning 14(1), 149–180 (1995)
3. Straccia, U.: Default inheritance reasoning in hybrid KL-ONE-style logics. In: IJCAI 1993, pp. 676–681. Morgan Kaufmann, San Francisco (1993)
4. Padgham, L., Zhang, T.: A terminological logic with defaults: A definition and an application. In: IJCAI 1993, pp. 662–668. Morgan Kaufmann, San Francisco (1993)
5. Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Trans. Comput. Logic 3(2), 177–225 (2002)
6. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: An epistemic operator for description logics. Artif. Intell. 100(1-2), 225–274 (1998)
7. Cadoli, M., Donini, F.M., Schaerf, M.: Closed world reasoning in hybrid systems. In: Methodologies for Intelligent Systems (ISMIS 1990), pp. 474–481. North-Holland, Amsterdam (1990)
8. Bonatti, P.A., Lutz, C., Wolter, F.: Expressive non-monotonic description logics based on circumscription. In: KR 2006, pp. 400–410. AAAI Press, Menlo Park (2006)
9. Gómez, S., Chesñevar, C., Simari, G.: An argumentative approach to reasoning with inconsistent ontologies. In: KROW 2008. CRPIT, vol. 90, pp. 11–20. ACS (2008)
10. Eiter, T., Ianni, G., Krennwallner, T., Polleres, A.: Rules and Ontologies for the Semantic Web. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web. LNCS, vol. 5224, pp. 1–53. Springer, Heidelberg (2008)
11. Eiter, T., Ianni, G., Krennwallner, T., Schindlauer, R.: Exploiting conjunctive queries in description logic programs. Ann. Math. Artif. Intell (2009); Published online January 27 (2009)
12. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. 172(12-13) (2008)
13. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge (2003)
14. Cholewinski, P., Truszczynski, M.: Minimal number of permutations sufficient to compute all extensions a finite default theory (unpublished note)
15. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective integration of declarative rules with external evaluations for semantic web reasoning. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 273–287. Springer, Heidelberg (2006)
16. Haarslev, V., Möller, R.: Racer system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 701–706. Springer, Heidelberg (2001)
17. Eiter, T., Fink, M., Krennwallner, T.: Decomposition of Declarative Knowledge Bases with External Functions. In: IJCAI 2009 (July 2009) (to appear)
18. Horrocks, I., Patel-Schneider, P.F.: Reducing OWL entailment to description logic satisfiability. J. Web Semant. 1(4), 345–357 (2004)
19. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a Web ontology language. J. Web Semant. 1(1), 7–26 (2003)
20. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In: Ganzinger, H., McAllester, D., Voronkov, A. (eds.) LPAR 1999. LNCS, vol. 1705, pp. 161–180. Springer, Heidelberg (1999)