CP4DL: Constraint-based Reasoning for Expressive Description Logics

Wudhichart Sawangphol, Yuan-Fang Li, and Guido Tack

Monash University, Melbourne, Australia
{wudhichart.sawangphol,yuanfang.li,guido.tack}@monash.edu

Abstract. Description logics (DLs) are a family of logic-based knowledge representation formalisms, which provide underlying semantics for modern ontology languages such as OWL 2. Reasoners for DLs are mostly specialised algorithms, which can answer questions such as whether an ontology as a whole is consistent, whether a certain concept in an ontology can be non-empty, or whether one concept subsumes another concept. Language extensions such as support for concrete domains (e.g. numbers) have been proposed in the past, but every extension typically requires the development of new reasoning algorithms. Some extensions have in fact never been implemented. This paper explores the use of CP technology for handling DL reasoning tasks. We show that CP modelling languages make it easy to extend DLs with new features, using a direct, high-level encoding into MiniZinc. Furthermore, our experiments show that modern CP solvers based on Lazy Clause Generation can be used as efficient DL reasoners. We present the first implementation of a reasoner for a DL that supports concrete domains and aggregate functions.

Keywords: Description Logic, Ontology Reasoning, Concrete domains, Aggregation, Constraint Modelling, MiniZinc

1 Introduction

Description logics (DLs) [6] are a family of logic-based knowledge representation formalisms, which provide underlying semantics for modern ontology languages such as OWL 2 [10]. A Description Logic defines *axioms* that state how certain *concepts* are related to each other, where concepts can be considered sets of abstract objects called *individuals*. Concepts are typically defined using constructs such as conjunction and disjunction, as well as quantification over binary relations, so-called *roles*.

As an example, Fig. 1 shows a fragment of Gene Ontology (GO) [1]. It introduces the concepts DomainCategory, GeneralisedStructure, AbstractStructure, DiabetogenicStructure, and Diabetes, and defines axioms that constrain their sub-concept relationships. E.g., a GeneralisedStructure subsumes an AbstractStructure, which simply means that all individuals that are classified as AbstractStructure are also classified as GeneralisedStructure. The class DiabetogenicStructure is defined as those individuals that are both classified as GeneralisedStructure, and for which there exists a successor individual in the IsCausallyLinkedTo binary relation that is classified as Diabetes.

Fig. 1. A fragment of Gene Ontology (GO).

Ontology Reasoning. Description Logics serve as formal languages for ontologies, describing complex concepts and relationships. In addition to mere notation, the formal nature of the languages allows for certain *reasoning tasks* to be performed automatically. Typical tasks check whether an ontology is *consistent* (i.e., whether the axioms do not contradict each other), whether a certain *concept is satisfiable* (i.e., whether the axioms allow the concept to be non-empty), or whether one concept *subsumes* another concept. Automatic DL reasoners are important tools for maintaining the quality of a large ontology and for deducing knowledge that is only encoded implicitly in the axioms.

Several specialised DL languages have been defined, with different levels of expressivity and, consequently, different computational complexity of the associated queries. This ranges from basic languages such as $\mathcal{EL}$ [2], for which many reasoning tasks can be performed in polynomial time, over $\mathcal{ALC}$ [25], for which concept satisfiability is PSpace- or even ExpTime-complete (depending on further limitations), to highly expressive languages that include support for so-called *concrete domains* [4, 22, 16] (such as numbers or strings), where care must be taken to restrict the language in order to even retain decidability.

Reasoning algorithms. Tableau calculi [6] are the mainstream reasoning algorithms for expressive DLs (we will discuss related approaches in detail in Sec. 4). Most modern ontology reasoners, such as HermiT [26] and Konclude [28], are based on specialised tableau procedures. However, the efficiency of tableau-based algorithms is still a bottleneck for large and complex ontologies. A major source of inefficiency is the large search space generated by disjunctive constructs (such as existential quantification). A further limitation of these algorithms is that any extension of the language typically requires significant re-engineering of the reasoning algorithm. Due to these limitations, only few implementations exist for highly expressive extensions such as support for concrete numerical domains.

Contributions. The main contribution of this paper is a high-level encoding scheme that allows us to translate ontologies modelled using expressive DLs into the constraint modelling language MiniZinc [21]. We can then exploit the power of modern Constraint Programming solvers to reason about ontology satisfiability, concept satisfiability, and concept subsumption. We demonstrate the flexibility of our approach by extending the encoding to support DLs with concrete domains and aggregate functions. To the best of our knowledge our CP-based approach is the first implementation of reasoning over such a language.

We discuss our implementation and present empirical results that show that CP solvers are competitive with and sometimes outperform dedicated reasoning algorithms on standard ontologies, and that they present a feasible implementation

strategy to support concrete domains and aggregation (for which there is currently no alternative implementation).

2 Description Logics

In this section we provide a brief introduction to the syntax and semantics of some description logics (DLs), starting from a simple DL $\mathcal{EL}$. Two extensions of $\mathcal{EL}$ are described: a well-known DL $\mathcal{ALC}$ as well as a new DL that encompasses concrete domains and aggregations over them, $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. Finally, some core reasoning tasks for DLs are introduced at the end of the section.

2.1 $\mathcal{EL}$

The description logic $\mathcal{EL}$ was designed to be tractable [2], while being expressive enough to represent knowledge in several large and widely-used biomedical ontologies such as GALEN [23], Gene ontology (GO, see Fig. 1) [1] and SNOMED CT [27].

Let A and B represent concept names, $\hat{C}$,$\hat{D}$,... represent (anonymous) concept descriptions, and R represent a role. An $\mathcal{EL}$ TBox (terminology box) $\mathcal{T}$ is a finite set of axioms as defined in the bottom of Table 1.

The semantics of $\mathcal{EL}$ (and other DLs) is normally defined in terms of *interpretations*. Due to readability, the semantics is described using Set Theory. $\mathbb{U}$ denotes the abstract domain or universe (a set of all individuals in the domain). An individual is an element in a set. A concept name A is a set S_A, where $S_A \subseteq \mathbb{U}$. A role name R is a binary relation that maps individuals to individuals, i.e., an individual x is R-related to an individual y, so y is called $R-$*successor*. In other words, it is a set of pairs of individuals S_R, where $S_R \subseteq \mathbb{U} \times \mathbb{U}$. A concept description $\hat{C}$ are defined through the concept constructs listed at the top of Table 1. For example, $\hat{C} = A \sqcap B \sqcap C$. Then the semantics of $\hat{C}$ is defined as $S_{\hat{C}} = S_A \cap S_B \cap S_C$.

An $\mathcal{EL}$ TBox $\mathcal{T}$ is in normal form if all axioms in $\mathcal{T}$ are of the following form: $\sqcap_i A_i \sqsubseteq B$, $A \sqsubseteq \exists R.B$, and $\exists R.B \sqsubseteq A$. Concept equivalence $A \equiv B$ can be normalised into two concept inclusion (subsumption) axioms $A \sqsubseteq B$ and $B \sqsubseteq A$.

Table 1. The syntax and semantics of $\mathcal{EL}$.

Concepts	**Syntax**	**Semantics**
top concept	$\top$	$\mathbb{U}$
atomic concept	A	S_A
conjunction	$\hat{C} \sqcap \hat{D}$	$S_{\hat{C}} \cap S_{\hat{D}}$
existential restriction	$\exists R.\hat{C}$	$\{x \in \mathbb{U} \mid \exists y : (x,y) \in S_R \wedge y \in S_{\hat{C}}\}$
Axioms	**Syntax**	**Semantics**
concept inclusion (subsumption)	$\hat{C} \sqsubseteq \hat{D}$	$S_{\hat{C}} \subseteq S_{\hat{D}}$
concept equivalence	$\hat{C} \equiv \hat{D}$	$S_{\hat{C}} = S_{\hat{D}}$

$\mathcal{EL}$ can be extended in several ways. On one hand, it has been extended by adding new concept constructs to obtain more expressivity in describing the abstract domain (concepts). We use the well-known DL $\mathcal{ALC}$ to describe such extensions in Sec. 2.2. On the other hand, it can be extended to obtain more expressivity by adding support for concrete domains (e.g., natural numbers). We illustrate such an extension with DL $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ in Sec. 2.3.

2.2 $\mathcal{ALC}$

One of the most well-known descriptions logics is $\mathcal{ALC}$ [25]. While it predates $\mathcal{EL}$, logically it can be seen as an extension of it. In addition to the constructs supported by $\mathcal{EL}$, $\mathcal{ALC}$ contains the bottom concept ($\bot$), concept disjunction ($\sqcup$), universal restriction ($\forall$), and concept negation ($\neg$). The concept descriptions in $\mathcal{ALC}$ are defined through the concept constructs listed in Tables 1 and 2.

A TBox $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is a finite set of axioms as defined in Tables 1 and 2. We restrict our attention $\mathcal{ALC}$ TBoxes in normal form, where all axioms are of the following form: $\sqcap_i A_i \sqsubseteq B$, $A \sqsubseteq \sqcup_i B_i$, $A \sqsubseteq \exists R.B$, $\exists R.B \sqsubseteq A$, $A \sqsubseteq \forall R.B$, and $\forall R.B \sqsubseteq A$. Note that both A and B can be negated concepts.

Table 2. The extension syntax and semantics for $\mathcal{ALC}$.

Concepts	Syntax	Semantics
bottom concept	$\bot$	$\emptyset$
concept negation	$\neg\hat{C}$	$\mathbb{U}\backslash S_{\hat{C}}$
disjunction	$\hat{C}\sqcup\hat{D}$	$S_{\hat{C}}\cup S_{\hat{D}}$
universal restriction	$\forall R.\hat{C}$	$\{x\in\Delta^{\mathcal{I}} \mid \forall y:(x,y)\in S_R \rightarrow y\in S_{\hat{C}}\}$

2.3 $\mathcal{ELU}^{(\neg)}(f,\Sigma)$

The description logic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ extends $\mathcal{EL}$ with concept disjunction ($\sqcup$) and concrete domain (non-negative integer) features with aggregations. Table 3 shows the main extension over $\mathcal{EL}$, which is the introduction of *features*, which map abstract individuals to (in our case) non-negative integers.

Concept descriptions in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ are defined through the concept constructs in Table 3, where f represents a (functional) feature and $\bowtie$ respresents a relational operator (a binary predicate). The semantics of aggregation functions is defined using multisets [8]. A $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox is a finite set of axioms as defined in Table 3.

Table 3. The syntax and semantics of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$.

Concepts	Syntax	Semantics
concrete domain	$\bowtie.(F_1,F_2)$	$\{x \in \mathbb{U} \mid \exists d_1, d_2 \in \mathbb{D} : (x,d_1) \in S_{F_1} \wedge (x,d_2) \in S_{F_2} \wedge (d_1,d_2) \in S_{\bowtie}\}$, where $\bowtie \in \{\geq,<,\leq,>,=,\neq\}$ and $S_{\bowtie} \subseteq \mathbb{D}\times\mathbb{D}$
Features	**Syntax**	**Semantics**
atomic feature	f	$\mathbb{U}\rightarrow\mathbb{D}$
natural number	d	d
aggregation	$\Sigma(R\circ f)$	$\begin{cases}\Sigma(M_x^{(R\circ f)}), & \text{if } M_x^{(R\circ f)} \text{is a multiset}\\ \text{undefined}, & \text{otherwise}\end{cases}$ where $M_x^{(R\circ f)} = [\![d \mid \exists y:(x,y)\in S_R \wedge f(y)=d]\!]$ and $\Sigma\in\{\mathsf{sum},\mathsf{count},\mathsf{min},\mathsf{max}\}$
Axioms	**Syntax**	**Semantics**
concept inclusion	$\hat{C}\sqsubseteq\hat{D}$	$S_{\hat{C}}\subseteq S_{\hat{D}}$
concept definition	$A\equiv\hat{D}$	$S_A = S_{\hat{D}}$

In order to retain decidability, some syntactic restrictions are placed on $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. The syntactic restrictions are:

1. $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ allows only acyclic TBoxes [3]. A TBox $\mathcal{T}$ is called *acyclic* if there are no cyclic dependencies between its concept names, i.e., concept names are neither defined directly nor indirectly in terms of themselves through axioms in $\mathcal{T}$.

2. $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ allows negation to be only in front of concept names.
3. Negation ($\neg A$) is not allowed on the left hand side of concept inclusion axioms.
4. Each feature needs to have a finite range of values, e.g., for a feature weight, $0 \leq \mathsf{weight}^{\mathcal{I}}(x) \leq d$ for any individual x, where d is natural number.
5. The use of concept equivalence axioms is restricted to definitions of concept names.

The semantics of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is defined using Set Theory. $\mathbb{D}$ is the concrete domain. For our work, we consider $\mathbb{D}$ to be a domain of natural numbers. In addition, F is constructed using the feature constructs in Table 3. F is a set of pairs of an abstract individual and a concrete individual S_F, where $S_F \subseteq \mathbb{U} \times \mathbb{D}$. A feature name f is a partial function $f : \mathbb{U} \to \mathbb{D}$.

The semantics of aggregations is defined in terms of multisets of concrete domain values. The mapping $M_x^{(R \circ f)}$ is, for each abstract individual x, the multiset of feature values f of its R-successors. For example, the concept description $=.(\mathsf{steps},\mathsf{sum}(\mathsf{exercise} \circ \mathsf{steps}))$ of axiom A6 in Fig 3 means that any individual in this concept needs to have steps-value equal to the sum of steps-values of exercise-successors. The multiset $\mathsf{exercise} \circ \mathsf{steps}$ is $[\![4000,3000,3000]\!]$, where there are three exercise-successors and each successor has steps-value 4000, 3000, and 3000 respectively. Thus, the sum value of the multiset $\mathsf{exercise} \circ \mathsf{steps}$ is 10000.

An $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ concept description is in *negative normal form* (NNF) when negation appears only in front of concept names. An $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox is in NNF when all of its axioms are in the following normal form: $A \sqsubseteq B$, $\sqcap_i A_i \sqsubseteq B$, $A \sqsubseteq \sqcup_i B_i$, $A \sqsubseteq \exists R.B$, $A \sqsubseteq \bowtie.(F_1,F_2)$, $A \equiv \sqcap_i B_i$, $A \equiv \sqcup_i B_i$, $A \equiv \exists R.B$, and $A \equiv \bowtie.(F_1,F_2)$. Note that only B can be concept negation, but not A.

Let us illustrate the usefulness of concrete domains in description logics with the help of an example. The ontology in Fig. 2 models daily fitness activities. For example, Treadmill is an exercise machine that a person may use for one hour (represented by feature hours) to get 4000 steps (represented by feature steps) and 800 calories burnt (represented by feature calburn). The ontology also defines a training goal (GoalState) of between 3 and 5 hours of exercise, more than 10,000 steps, at least 2,000 calories burnt, and reaching a maximum heart rate (represented by feature HR) of has-successors of at least 128 beats per minute.

$$\mathsf{Treadmill} \equiv =.(\mathsf{hours},1) \sqcap \geq.(\mathsf{steps},4000) \sqcap \geq.(\mathsf{calburn},800) \quad \text{(A1)}$$

$$\mathsf{FlexStrider} \equiv =.(\mathsf{hours},1) \sqcap \geq.(\mathsf{steps},3000) \sqcap \geq.(\mathsf{calburn},700) \quad \text{(A2)}$$

$$\mathsf{CrossTrainers} \equiv =.(\mathsf{hours},1) \sqcap \geq.(\mathsf{steps},3000) \sqcap \geq.(\mathsf{calburn},750) \quad \text{(A3)}$$

$$\begin{aligned} \mathsf{GoalState} \equiv\ & \geq.(\mathsf{hours},3) \sqcap \leq.(\mathsf{hours},5) \sqcap \\ & \geq.(\mathsf{steps},10000) \sqcap \geq.(\mathsf{calburn},2000) \sqcap \\ & =.(\mathsf{HR},\mathsf{max}(\mathsf{has} \circ \mathsf{HR})) \sqcap \geq.(\mathsf{max}(\mathsf{has} \circ \mathsf{HR}),128) \sqcap \\ & =.(\mathsf{count}(\mathsf{has} \circ \mathsf{HR}),3) \end{aligned} \quad \text{(A4)}$$

Fig. 2. An $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ ontology about daily fitness.

This static ontology is then combined with a *stream ontology*, a representation of the actual exercise performed by one person during a certain time window (e.g. obtained through a fitness tracking device). In this example, we have chosen a window size of three activities.

$$\begin{aligned}
\mathsf{CurrentStateA} \sqsubseteq\ & \exists\ \mathsf{exercise.Treadmill} \sqcap =.(\mathsf{id},1) \sqcap =.(\mathsf{HR},100) \sqcap \\
& =.(\mathsf{hours},\mathsf{sum}(\mathsf{exercise}\circ\mathsf{hours})) \sqcap =.(\mathsf{calburn},\mathsf{sum}(\mathsf{exercise}\circ\mathsf{calburn})) \sqcap \\
& =.(\mathsf{steps},\mathsf{sum}(\mathsf{exercise}\circ\mathsf{steps})) \sqcap =.(\mathsf{count}(\mathsf{exercise}\circ\mathsf{steps}),1) \sqcap \\
& =.(\mathsf{count}(\mathsf{exercise}\circ\mathsf{hours}),1) \sqcap =.(\mathsf{count}(\mathsf{exercise}\circ\mathsf{calburn}),1) && \text{(A5)} \\
\mathsf{CurrentStateB} \sqsubseteq\ & \exists\ \mathsf{exercise.FlexStrider} \sqcap =.(\mathsf{id},2) \sqcap =.(\mathsf{HR},110) \sqcap ... && \text{(A6)} \\
\mathsf{CurrentStateC} \sqsubseteq\ & \exists\ \mathsf{exercise.CrossTrainers} \sqcap =.(\mathsf{id},3) \sqcap =.(\mathsf{HR},128) \sqcap ... && \text{(A7)} \\
\mathsf{CurrentState} \equiv\ & \exists\ \mathsf{has.CurrentStateA} \sqcap \exists\ \mathsf{has.CurrentStateB} \sqcap \exists\ \mathsf{has.CurrentStateC} \sqcap \\
& =.(\mathsf{hours},\mathsf{sum}(\mathsf{has}\circ\mathsf{hours})) \sqcap =.(\mathsf{calburn},\mathsf{sum}(\mathsf{has}\circ\mathsf{calburn})) \sqcap \\
& =.(\mathsf{steps},\mathsf{sum}(\mathsf{has}\circ\mathsf{steps})) \sqcap =.(\mathsf{HR},\mathsf{max}(\mathsf{has}\circ\mathsf{HR})) \sqcap \\
& =.(\mathsf{count}(\mathsf{has}\circ\mathsf{hours}),3) \sqcap =.(\mathsf{count}(\mathsf{has}\circ\mathsf{calburn}),3) \sqcap \\
& =.(\mathsf{count}(\mathsf{has}\circ\mathsf{steps}),3) \sqcap =.(\mathsf{count}(\mathsf{has}\circ\mathsf{HR}),3) && \text{(A8)} \\
\mathsf{CurrentState} \sqsubseteq\ & \mathsf{GoalState} && \text{(A9)}
\end{aligned}$$

Fig. 3. A stream ontology recording daily fitness activities.

The stream ontology (Fig. 3) models three activities (CurrentStateA, CurrentStateB, and CurrentStateC) and combines them into CurrentState. The id feature is used to identify each activity. CurrentStateA, for instance, represents that a person has exercised on the Treadmill with a heart rate of 100 for one hour. Aggregate functions are used to link CurrentState with its component activities, e.g. by summing over their hours and calories, and taking the maximum of their heart rate features.

We can now use ontology reasoning to check whether a person achieves her goal. We combine the static ontology with a concrete stream ontology (for one time window) and then check whether CurrentState is a subclass of GoalState.

2.4 Reasoning Tasks

Consistency Checking is used to determine whether $\mathcal{T}$ is consistent [6]. $\mathcal{T}$ is consistent if there exists a model $\mathcal{I}$ of $\mathcal{T}$ that satisfies all axioms in $\mathcal{T}$. Otherwise, $\mathcal{T}$ is inconsistent.

Concept Satisfiability Checking is used to ensure that a particular concept is satisfiable w.r.t $\mathcal{T}$ when $\mathcal{T}$ is consistent [6]. A concept description $\hat{C}$ is satisfiable w.r.t $\mathcal{T}$, if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $S_{\hat{C}}$ is not empty. Otherwise, concept description $\hat{C}$ is not satisfiable.

Concept Subsumption Checking checks whether one concept is more general than another [6]. If a concept description $\hat{C}$ is subsumed by a concept description $\hat{D}$ w.r.t $\mathcal{T}$, i.e., $\mathcal{T} \vDash \hat{C} \sqsubseteq \hat{D}$, then $S_{\hat{C}} \subseteq S_{\hat{D}}$ is true for all models $\mathcal{I}$ of $\mathcal{T}$.

$\mathcal{ALC}$ is closed under negation, hence it supports full subsumption checking. On the other hand, $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is not closed under negation, we can only perform *limited* concept subsumption checking, where only concept names are allowed as subclasses. To check subsumption $A \sqsubseteq \hat{C}$, we can convert and negate it to $A \sqcap \neg\hat{C}$ and check whether this concept is unsatisfiable. Since we need to negate $\hat{C}$, $\hat{C}$ cannot contain existential quantifications ($\exists$).

3 Description Logics Reasoning via MiniZinc Encoding

We propose a DL reasoning algorithm by encoding an ontology into a MiniZinc model, which can then be solved by CP solvers. Our encoding scheme translates concepts

and roles in Tables 1, 2, and 3, in a direct and succinct way, to set and array variables in MiniZinc respectively. Importantly, our encoding into MiniZinc has linear time and space complexity.

In our encoding scheme, *individuals* are encoded as positive integers. Moreover, we encode the top concept $\top$ (the superclass of all concepts) and a concept A as set variables of abstract individuals `T` and `A` respectively. Hence, `A subset T` is true for all concepts A. A role R is a binary relation that maps abstract individuals of one concept (set) to abstract individuals (successors) of another concept (set). It is encoded as an array of sets `R`, where indices `i` are individuals in one set and the set `R[i]` contains the R-successors of individual i. A (functional) feature f maps an abstract individual to a natural number. It is encoded as an array of natural numbers `f`, whose indices are individuals in its domain, and the natural number `f[i]` is the f-value of individual i. Our encoding scheme closely follows the semantics of description logics.

3.1 MiniZinc Encoding for $\mathcal{EL}$

Without loss of generality, we assume an $\mathcal{EL}$ TBox $\mathcal{T}$ is in normal form (Sec. 2.1), where axioms in $\mathcal{T}$ can be of only three cases: (1) $\sqcap_i A_i \sqsubseteq B$, (2) $A \sqsubseteq \exists R.B$, and (3) $\exists R.B \sqsubseteq A$. $\mathcal{T}$ can be encoded by applying the following encoding rules. Let i be an individual, and n,m,k non-negative integers. Axioms in $\mathcal{T}$ are encoded as follows:

EL1 For every axiom $\hat{A} \sqsubseteq \hat{B}$ (i.e., $\sqcap_n A_n \sqsubseteq \sqcup_m B_m$),

```
constraint (A_1 intersect ... A_n subset B_1 union ... B_m)        (3.1)
```

This rule is straightforward since the conjunction (resp. union) of concept names can be easily translated into as `intersect` (resp. `union`) operations of sets $\mathtt{A}_i$, and the subclass relation $\sqsubseteq$ can be translated into a `subset` constraint in MiniZinc.

EL2 For every axiom $A \sqsubseteq \exists R.B \in \mathcal{T}$,

```
constraint forall (i in T)                                         (3.2)
    (i in A  ->  card(R[i] intersect B) >= 1)
```

The subclass relation $\sqsubseteq$ is encoded into an implication: If an individual `i` is in `A`, then the cardinality of the intersection of R-successors of `i` and `B` is at least 1.

EL3 Similar to the above rule, for every axiom $\exists R.B \sqsubseteq A \in \mathcal{T}$,

```
constraint forall (i in T)                                         (3.3)
    (card(R[i] intersect B) >= 1  ->  i in A)
```

3.2 MiniZinc Encoding for $\mathcal{ALC}$

Similarly, for an $\mathcal{ALC}$ TBox $\mathcal{T}$ in normal form, there are only six types of axiom: (1) $\sqcap_i A_i \sqsubseteq B$, (2) $A \sqsubseteq \exists R.B$, (3) $\exists R.B \sqsubseteq A$, (4) $A \sqsubseteq \sqcup_i B_i$, (5) $A \sqsubseteq \forall R.B$, and (6) $\forall R.B \sqsubseteq A$. For the first four types, the encoding rules are the same as those for $\mathcal{EL}$ (note (4) can be encoded using rule EL1). For (5) and (6), the following encoding rules are introduced:

ALC1 For every axiom $A \sqsubseteq \forall r.B \in \mathcal{T}$ (type (5)),

```
constraint forall(i in T) (i in A -> R[i] subset B)                    (3.4)
```

ALC2 For every axiom $\forall r.B \sqsubseteq A \in \mathcal{T}$ (type (6)),

```
constraint forall(i in T) (R[i] subset B -> i in A)                    (3.5)
```

The negation of a concept A can be encoded as (`T diff A`), where `diff` is set difference. As can be seen, it is easy to extend the encoding scheme for $\mathcal{EL}$ to support $\mathcal{ALC}$, and the encoded MiniZinc constraints closely follow the semantics of the DL.

3.3 MiniZinc Encoding for $\mathcal{ELU}^{(\neg)}(f,\Sigma)$

$\mathcal{ELU}^{(\neg)}(f,\Sigma)$ extends $\mathcal{EL}$ with concrete domains and aggregations over them. An $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox in negation normal form contains only axioms of the forms: (1) $\sqcap_i A_i \sqsubseteq B$, (2) $A \sqsubseteq \sqcup_j B_j$, (3) $A \sqsubseteq \exists r.B$, (4) $A \sqsubseteq \bowtie.(F_1,F_2)$, (5) $A \equiv \sqcap_i B_i$, (6) $A \equiv \sqcup_i B_i$, (7) $A \equiv \exists R.B$, and (8) $A \equiv \bowtie.(F_1,F_2)$. The encoding for (1)–(3) is the same as for $\mathcal{EL}$ and $\mathcal{ALC}$.

In the following we describe the encoding rules for (4) and (8), which involve concrete domains, features, and aggregation. Each feature f is encoded into an array `f` that maps individuals to integers. In order to handle the aggregations ($\Sigma(R \circ f)$), we introduce variables as follows:

- `x_`Σ`_R_f` be an array of natural numbers for each aggregation, where $\Sigma \in \{\mathsf{sum}, \mathsf{count}, \mathsf{max}, \mathsf{min}\}$. It contains the result of aggregating over the multiset for each individual (indices of the array).
- `sx_`Σ`_R_f` be a **slack variable**, which is an array of natural numbers for each aggregation count and sum, i.e., $\Sigma \in \{\mathsf{sum},\mathsf{count}\}$.
- `bx_`Σ`_R_f` be an array of Boolean variables for each aggregation max and min, i.e., $\Sigma \in \{\mathsf{max},\mathsf{min}\}$. The value of each array index is true if the value of `x_`Σ`_R_f` is in its multiset. Otherwise, the value is false.
- ub_array be a function that returns the maximum value of an array.

The following describes the encoding scheme for concrete domain and aggregations:

ELF1 Initialisation: for every R, f and aggregation $\Sigma \in \{\mathtt{min},\mathtt{max},\mathtt{sum},\mathtt{count}\}$, introduce `x_`$\Sigma$`_R_f`, `sx_`$\Sigma$`_R_f`, `bx_`$\Sigma$`_R_f`. Additionally, the following constraints will also be created for handling aggregations and their interactions.

```
constraint forall(i in T)(x_count_R_f[i]) =                            (3.6)
    card(R[i]) + bool2int(bx_max_R_f[i]) + bool2int(bx_min_R_f[i])) +
    sx_count_R_f[i]

constraint forall(i in T)(x_sum_R_f[i] =                               (3.7)
    sum(j in R[i])
        (if (j in R[i]) then f[j] else 0 endif) +
        sx_sum_R_f[i])

constraint forall(i in T) (                                            (3.8)
    (x_min_R_f[i] = min(j in R[i])
        (if (j in R[i]) then f[j] else ub_array(f)+1 endif)) ∨
    (x_min_R_f[i] < min(j in R[i])
        (if (j in R[i]) then f[j] else ub_array(f)+1 endif) ∧
     bx_min_R_f[i]));
```

```
constraint forall(i in T)(                                                    (3.9)
    (x_max_R_f[i] = max(j in R[i])
        (if (j in R[i]) then f[i] else 0 endif)) ∨
    (x_max_R_f[i] > max(j in R[i])
        (if (j in R[i]) then f[i] else 0 endif]) ∧
     bx_max_R_f[i]))

constraint forall(i in T)                                                     (3.10)
    (sx_sum_R_f[i] >= xmin_R_f[i] * sx_count_R_f[i])

constraint forall(i in T)                                                     (3.11)
    (sx_sum_R_f[i] <= xmax_R_f[i] * sx_count_R_f[i])

constraint forall(i in T)                                                     (3.12)
    (x_min_R_f[i] <= x_max_R_f[i])
```

ELF2 For every axiom $A \sqsubseteq \bowtie.(F_1,F_2) \in \mathcal{T}$ (type (4)),

```
constraint forall(i in A) (tran(F1) ⋈ tran(F2))                               (3.13)
```

where $\bowtie \in \{\geq,<,\leq,>,=,\neq\}$, and the function `tran`$(F_k)$ is defined as follows

$$\texttt{tran}(F_k) = \begin{cases} \texttt{f[i]} & \text{(i) if } F_k = f \text{ } (F_k \text{ is a feature}) \\ \texttt{d} & \text{(ii) if } F_k = d \text{ } (F_k \text{ is an integer value}) \\ \texttt{x\_}\Sigma\texttt{\_R\_f[i]} & \text{(iii) if } F_k = \Sigma(R \circ f) \text{ } (F_k \text{ is an aggregation}) \end{cases}$$

Let us consider each case as follows:
(i) A functional feature is translated into an array mapping abstract individuals to integers `f`. Therefore, `tran`(F) is encoded into `f[i]`.
(ii) A natural number is directly translated into a natural number. Therefore, `tran`(F) is encoded into `d` such that `d` is a natural number.
(iii) An aggregation is translated into the corresponding MiniZinc array. The array aggregates (with aggregation function $\Sigma \in \{$sum,count,max,min$\}$) the feature values of the abstract domain individuals (indices of the array).

However this may not be enough, and sometimes additional $R \circ f$-successors need to be introduced to satisfy the concrete domain constraints. To account for this additional contribution to an aggregation, *slack variables*, encoded as `sx_`Σ`_R_f`, are introduced to hold the aggregated value of all these additional successors, without actually introducing abstract individuals. A slack variable thereby corresponds to the slack value that is left in the concrete domain constraints introduced by [8].

For case (iii) where `tran`(F) is encoded into `x_`Σ`_R_f[i]`, we next consider each aggregation as follows.
count: encoded into a constraint 3.6. The first part of the constraint `card(R[i])` counts actual elements in the multiset. `sx_count_R_f[i]` is a number of additional $R \circ f$-successors (i.e., slacks) to satisfy the concrete domain constraints. `bx_min_R_f[i]` (resp. `bx_max_R_f[i]`) represents an additional $R \circ f$-successor that may be required to satisfy the min (resp. max) constraint.
sum: encoded into a constraint 3.7. The first part of constraint `sum(j in R[i])(if (j in R[i]) then f[i]else 0 endif)` sums the value of elements in a multiset. `sx_sum_R_f[i]` is a value of additional $R \circ f$-successors to satisfy the concrete domain constraints.

min: encoded into a constraint 3.8. The first disjunct is the case that the result of `min` function is in the multiset. The second disjunct is the case that the result of `min` function is not in the multiset. Then `x_min_R_f[i]` can be assigned by a natural number that is less than the result of `min` function over actual individuals in a multiset and `bx_min_R_f[i]` is a signal that is used in the `count` function since the `count` function has to increment when `bx_min_R_f[i]` is true.
max: is encoded similarly to min.
count and sum: Constraints 3.10 and 3.11 need to be added to ensure that the average value of elements in a multiset is between min and max.
min and max: Constraint 3.12 needs to be added to ensure that the min value is less than or equal to the max value.

Concept definitions ($\equiv$, types (5)–(8)) are be encoded into equivalences (`=` for sets, `<->` for individuals). For example, $A \equiv \sqcap_i B_i$ is encoded into a constraint `A = B1 intersect B2 intersect ....`

ELF3 To check the satisfiability of concept A, a constraint `card(A) > 0` is added. To check subsumption of $A \sqsubseteq B$, a constraint `card(A intersect (T diff B)) > 0` is added.

Fig. 4 below shows the corresponding MiniZinc model generated by our encoding scheme for the fitness ontology in Fig 2.

```
constraint forall(i in T)((i in Treadmill) <->
    ((hours[i] = 1) ∧ (steps[i] = 4) ∧ (calburn[i] = 800))
);
constraint forall(i in T)((i in FlexStrider) <->
    ((hours[i] = 1) ∧ (steps[i] = 3) ∧ (calburn[i] = 700))
);
constraint forall(i in T)((i in CrossTrainers) <->
    ((hours[i] = 1) ∧ (steps[i] = 3) ∧ (calburn[i] = 750))
);
constraint forall(i in T)((i in GoalState) <->
  ((hours[i] >= 3) ∧ (hours[i] <= 5) ∧
   (steps[i] >= 10) ∧ (calburn[i] >= 2000) ∧
   (HR[i] = max (j in has[i])
       (if (j in has[i]) then HR[j] else 0 endif)) ∧
   (max (j in has[i])
       (if (j in has[i]) then HR[j] else 0 endif) >= 128) ∧
   (card (has[i]) = 3))
);
```

Fig. 4. The MiniZinc model for the fitness ontology in Fig 2.

3.4 Finiteness of MiniZinc Models

Since our CP-based encoding requires a bounded universe, we need to calculate the number of individuals needed for the MiniZinc model. The number of necessary individuals to satisfy the abstract domain axioms is determined as the root individual plus one individual per existential quantification ($\exists$). Moreover, this number is usually small for typical ontologies (TBoxes). Hence, we do not need to perform *lifted inference* and can reason about individuals in an abstract way. However, for the case of concrete domains and aggregations, we perform reasoning on these in a lifted way by introducing *slack variables*.

3.5 Symmetry breaking and search

The encodings presented translate each syntactic construct of a DL into a corresponding MiniZinc construct. In conjunction with a suitable encoding of the reasoning task, we can therefore use the readily available MiniZinc solvers as ontology reasoners.

For a CP expert, it will however become clear very soon that the encoding introduces *symmetry* by identifying abstract individuals with natural numbers. In any solution to the MiniZinc model, we can swap the names of any pair of individuals to produce again a solution, as long as we take care and also swap all of their feature values.

The encoding can therefore be extended with *symmetry breaking constraints*, avoiding the unnecessary exploration of symmetric states by the constraint solver. In general, breaking value symmetries like this one can be achieved by imposing a lexicographic order `[i in A, i in B, i in C, ...] >= [i+1 in A, i+1 in B, i+1 in C, ...]`, where `i` ranges over the possible set elements and `A,B,C,...` are the set-valued variables.

We chose to implement simpler constraints to break only some of the symmetry, depending on the reasoning task. When testing **satisfiability** of a concept A: Instead of the constraint `card(A) > 0`, we can force individual `1` to be in `A` without loss of generality. In addition, we can enforce that `forall (i in 2..n) (i in A -> i-1 in A)`, where `n` is the total number of individuals (since all other individuals are symmetric). When testing **subsumption** $A \sqsubseteq B$: Since we prove subsumption by testing `card(A intersect (T diff B)) > 0`, we can use the same symmetry breaking constraints as above (since `A` cannot be empty). In addition, we know that individual `1` cannot be in `B`, so we can add that constraint. Furthermore, we can add the constraint `2 in B \/ card(B)=0`, since all other individuals are still symmetric.

The experiments in Sec. 5 show that these symmetry breaking constraints have a dramatic effect on the runtime of the solver.

Another point to note is that MiniZinc supports the declaration of search heuristics, suggesting an order in which the solver should try labelling variables during the search. The structure of ontologies suggests a search heuristic that attempts to add one individual at a time to each concept in turn (rather than trying to "fill up" a concept before adding individuals to another concept).

In particular in the case of testing subsumption $A \sqsubseteq B$, we know that B must either be empty or contain individual 2 (since it cannot contain 1, see above, and all other individuals are symmetric). Thus, we can use a heuristic that will first try to add 2 to `B`, and in the case of failure the symmetry breaking constraint will force `B` to be empty.

4 Related Work

The most widely used class of algorithms for ontology reasoning over expressive DLs such as $\mathcal{ALC}$ are based on tableau calculi [6, 7]. These algorithms have been implemented in many the-state-of-the-art reasoners such as FaCT++ [29]. Over the years, a number of optimisation techniques and specialised reasoning algorithms have been developed such as Absorption [6], hyper-tableau [19, 18, 20], which is implemented in an ontology reasoner called HermiT [26], and tableau calculus enhanced with sophisticated preprocessing methods and saturation, which is implemented in Konclude [28].

Nonetheless, tableau-based algorithms still exhibit inefficiencies when they are used to perform reasoning on large ontologies, which contain many disjunctions ($\sqcup$),

existential quantifications, qualified number restriction, or concrete domains, such as the Genomic CDS ontology [24]. These disjunctive constructs cause inefficiency in tableau-based algorithms, because their non-deterministic expansion rules lead to exponential search spaces [19, 18, 20]. In addition, qualified number restrictions and concrete domains can increase the search space exponentially [5].

Aggregation over concrete domains was proposed in the logic $\mathcal{EL}(\Sigma)$ [8], which can handle functions such as count and sum (but not both at the same time). A tableau-based algorithm was proposed for concept satisfiability of $\mathcal{EL}(\Sigma)$, for reasoning on the abstract domain and for creating conjunctions of aggregations, features, and predicates as constraints (concrete domain reasoning). This conjunction can be subsequently decided by linear or mixed integer programming solvers. However, reasoning support for this logic has not been implemented.

Recently several approaches have been proposed to exploit SAT- and SMT-based solving for DL reasoning. Our work is similar in that we also exploit mature solving techniques, in this case Constraint Programming (CP).

In order to use SAT solvers, a SAT encoding approach for an ontology based on the description logic $\mathcal{ALCN}$ has been proposed [17]. This approach transforms an $\mathcal{ALCN}$ ontology in negation normal form (NNF) into propositional logic formulas in conjunctive normal form (CNF). A SAT solver is then used to check satisfiability of the propositional logic formulas. An SMT-based encoding approach has been proposed for a more expressive description logic $\mathcal{ALCQ}$ [11], where an $\mathcal{ALCQ}$ ontology is encoded into a SMT formula with Theory of Costs (SMT($\mathcal{C}$)) [9]. The Theory of Cost is used to handle qualified number restriction ($\mathcal{Q}$).

A common drawback of the above approaches is that their encoding scheme can lead to an exponential number of clauses in CNF due to number restrictions. The SMT-based approach [11] may produce an SMT model exponential in size of an ontology that contains nested qualified number restrictions and/or nested subsumption relationships. For example, axioms $A_i \sqsubseteq \geq 2R.A_{i+1}$, where $i \in 1..n$, are a series of nested at-least qualified number restrictions, will lead to an exponential blow-up. In comparison, our encoding into set-based MiniZinc models (which can be extended to $\mathcal{ALCQ}$) is linear in the size of the original ontology.

Another approach, called *Intelligent Tableau* [30], uses QBF solvers to improve performance on highly disjunctive ontologies. The main idea is to perform reasoning in tableau algorithm style but use a *Quantified Boolean Formulas* (QBF) solver to lead the expansion of the search tree for existential and universal quantification. Implemented in the LIGHT reasoner, this approach also incorporates global learning, including *Unsat-learning*, *Sat-learning* and *Unknown-learning*. Their evaluation shows significant improvement over previous, tableau-based approaches. LIGHT supports $\mathcal{ALC}$ and perform ontology consistency checking. However, LIGHT needs to feed a whole set of new propositional logic formulas to the QBF solver each time. This means the learned clauses do not survive to be used in subsequent invocations of the QBF solver. Our approach translates the entire ontology into a single constraint model, so that learning solvers can take advantage of a more global perspective.

5 Empirical Evaluation

We conducted two main evaluations. The first evaluation is to show that our approach is competitive with tableau reasoners. For this evaluation, we performed consistency checking on $\mathcal{ALC}$ ontologies to evaluate our approach against two of the fastest reasoners, HermiT [26] and Konclude [28]. The second evaluation is to show the effectiveness of our approach for handling concrete domain and aggregation. We performed an empirical test on limited concept subsumption testing of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ ontologies.

We have implemented the encoder `MiniZincEncoder` in Java to encode an ontology into a MiniZinc model. In the following, runtimes do not include the times needed for pre-processing and compiling the MiniZinc models to solver-dependent FlatZinc models since these times are less than a second in all cases.

A runtime limit of 300 seconds was used for the first evaluation. We limited a runtime of 20 seconds for the second evaluation. All results presented in this section have been obtained on a 64bit quad-core `Intel Core i7 2.7GHz` machine, with `16GB` of RAM under MAC OS X 10.11.4. We used MiniZinc version 2.0.12[1].

5.1 Evaluation of $\mathcal{ALC}$

For the first evaluation, we perform *consistency checking* on $\mathcal{ALC}$ ontologies.

We generated MiniZinc models using our `MiniZincEncoder`, and then solved the models using `Opturion CPX (ocpx)`[2] (v. 1.0.2) and `chuffed`[3], which are lazy clause generation CP solvers, and `Gecode`[4] (v. 4.4.0). We compare the runtimes of our system with those of state-of-the-art tableau-based reasoners HermiT[5] (v. 1.3.8) and Konclude[6] (v. 0.6.2). The LIGHT reasoner [30] is not available on Mac OS X, hence is not included in this evaluation. The SMT-based reasoner [11] in not included due to excessive errors caused by URIs and special symbols.

We used two different sets of benchmarks to test our encoding. Ontologies in the first set ("jnh") contain many concept disjunctions [30], but no existential quantifications. In terms of metric values, SUPDCHN (number of chains of disjunctions in super-concepts) [15] for this set ranges between 71 and 77, and DISJ (number of disjunctions in the ontologies) ranges between 741 and 795. It has been generated by converting SAT benchmarks [12] into OWL syntax. The second set of benchmarks ("k_XX") come from Tableaux'98 [13]. Ontologies in this set contain many existential quantifications (EF (existential quantifications) [14] ranges between 363 and 1,423) but no disjunctions. These datasets demonstrate how constructs such as chains of disjunctions and existential quantification can lead to inefficiencies in tableau-based reasoners. The name of each ontology cannot express a satisfiability result. For example, some ontologies (e.g., $k\_d4\_sat\_09$ and 10) are actually unsatisfiable.

The results of this evaluation are presented in Table 4 (time measured in seconds, "to" stands for time out). The results show that the approach presented in this paper

[1] `http://www.minizinc.org`
[2] `http://www.opturion.com/#!cpx/ch52y`
[3] `https://github.com/geoffchu/chuffed`
[4] `http://www.gecode.org/`
[5] `http://www.hermit-reasoner.com/`
[6] `http://www.derivo.de/en/products/konclude.html`

(columns `ocpx`, `chuffed`, and `gecode`) clearly outperforms the dedicated ontology reasoners. This shows that for ontologies with highly disjunctive structure or a significant number of existential quantifications, tableau-based reasoners struggle to perform well, in particular in the case of inconsistent ontologies. On the other hand, CP-based techniques, especially those based on clause learning (such as `ocpx` and `chuffed`) can demonstrate their advantages.

While this set of benchmarks is by no means exhaustive, it demonstrates that the CP4DL approach is feasible and effective, and that clause learning solvers are crucial for achieving high reasoning performance on difficult ontologies.

Table 4. Comparison of ontology consistency checking time on $\mathcal{ALC}$ ontologies.

Ontologies	ocpx	chuffed	gecode	HermiT	Konclude	SAT?
sat-jnh204.owl	0.117	0.041	0.075	9.426	to	Y
sat-jnh212.owl	0.159	0.039	0.121	116.731	to	Y
sat-jnh220.owl	0.181	0.041	0.188	0.431	to	Y
unsat-jnh16.owl	0.568	0.055	0.499	66.063	to	N
k_d4_sat_09.owl	0.447	0.157	29.820	to	32.542	N
k_d4_sat_10.owl	0.43	0.176	41.618	to	41.113	N
k_d4_unsat_09.owl	0.469	0.102	0.198	to	3.506	N
k_d4_unsat_10.owl	0.430	0.122	0.400	to	5.315	N
k_d4_unsat_12.owl	0.215	0.169	0.75	to	11.957	N
k_d4_unsat_13.owl	0.238	0.184	0.726	to	17.919	N
k_poly_sat_09.owl	1.127	0.149	62.151	to	12.509	Y
k_poly_sat_10.owl	1.519	0.174	105.948	to	16.730	Y
k_poly_sat_12.owl	1.208	0.238	165.606	to	35.800	Y
k_poly_sat_13.owl	0.662	0.264	73.033	to	63.389	Y
k_poly_unsat_09.owl	0.425	0.149	to	8.346	10.244	N
k_poly_unsat_10.owl	4.106	0.173	103.845	10.918	18.013	N
k_poly_unsat_15.owl	0.842	0.331	to	84.591	107.831	N
k_poly_unsat_16.owl	1.93	0.387	to	99.268	167.877	N

5.2 Evaluation of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$

This set of experiments evaluates the performance of the CP4DL approach on $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ ontologies. Since there are no other reasoning algorithms for ontologies with concrete domain and aggregation, and no public data sets for these ontologies, we only aim to show the feasibility of the approach.

The reasoning task tested here is *Limited Concept Subsumption Checking*, i.e., for a given ontology and two concepts A and B, check whether A is a subclass of B. To check this, as discussed in Sec. 3.3, we add the negation of the subsumption as an axiom and prove unsatisfiability. The negation is encoded as the constraint `card(A intersect (T diff B)) > 0`.

The evaluation in Sec. 5.1 showed that `chuffed` outperformed all other solvers. We therefore only report results for `chuffed` here.

We generated 1,000 $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ ontologies to test our approach. These ontologies are similar to the ontology in Figs 2 and 3, with randomised HR-values of the concepts CurrentStateA, CurrentStateB and CurrentStateC, and randomised hours-value, steps-value and calburn-value of Treadmill, FlexStrider and CrossTrainers. The random values for HR are between 100 and 220 (a normal range for somebody who is exercising). The random values for hours range from 1 to 3. The steps are calculated from the hours, where the average steps is 4000 per hour. Finally, the calburn is calculated from steps,

where the average calories burned is 0.05 per step. The generated test set consists of 762 satisfiable cases and 238 unsatisfiable cases.

The base line for our experiments is (1) a **basic model** which is simply the result of the `MiniZincEncoder`, without any symmetry breaking constraints or search heuristics. We compare the performance of this basic model with (2) the basic model plus symmetry breaking, (3) the basic model plus search heuristic, and (4) the basic model plus both symmetry breaking and search heuristic (as discussed in Sec. 3.5).

The results of the experiments are shown in Figs 5 and 6. The version with both symmetry breaking and search heuristic is the clear winner, and shows almost constant, low runtime. If we analyse the results more carefully, we can see that for satisfiable instances (Fig. 5), the search heuristic is crucial for performance, while symmetry breaking has a lesser effect. For unsatisfiable instances (Fig. 6), we get the opposite picture: symmetry breaking speeds up the reasoning dramatically, while the search heuristic alone does not have any clear positive effect. The combination of the two techniques seems crucial to achieve robust performance across satisfiable and unsatisfiable instances. The empirical evaluation confirms that the symmetry breaking and search heuristic make our approach more effective.

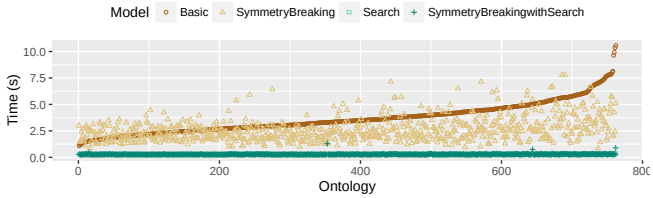

Fig. 5. Comparison of models with Chuffed: SAT ontologies.

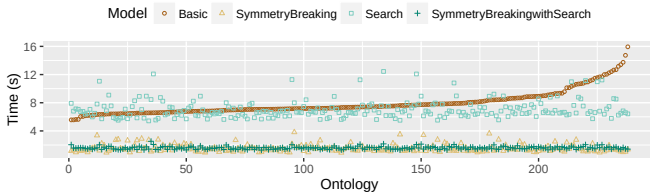

Fig. 6. Comparison of models with Chuffed: UNSAT ontologies.

6 Conclusion and Future Work

Description Logics are fundamental knowledge representation formalisms for Semantic Web ontologies. A number of description logics have been proposed that span a spectrum of expressivity. The development of a new, expressive description logic usually requires the development of new, specialised reasoning algorithms/systems.

In this paper, we present CP4DL, an alternative approach to ontology reasoning by exploiting modern CP techniques. Ontologies are encoded into MiniZinc models in a direct and succinct way. Existing CP solvers can then be readily applied to perform ontology reasoning without modification. We illustrate our CP-based reasoning

approach through the encoding of three description logics of varied expressivity. One of the logics, $\mathcal{ELU}^{(\neg)}(f,\Sigma)$, supports aggregation over concrete domains, and our CP-based approach is the first and only implementation for any logic with such features.

Our empirical evaluation on ontologies in the expressive DL $\mathcal{ALC}$ shows that our approach is competitive against existing ontology reasoners, outperforming some state-of-the-art tableau-based reasoners, sometimes by several orders of magnitude. We also demonstrate that, with some simple model-level optimisation, our approach is feasible for supporting concrete domain and aggregation reasoning in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ with sub-second reasoning time.

In the future, we plan to further investigate ways to improve reasoning efficiency. For example, we will identify more symmetries in encoded models to add more symmetry breaking constraints. We will also investigate deeper integration with existing CP solvers to support efficient reasoning for more expressive description logics.

References

1. Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G.: Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. Nature genetics 25(1), 25–29 (May 2000), http://dx.doi.org/10.1038/75556
2. Baader, F., Brand, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. of IJCAI 2005. pp. 364–369. Morgan-Kaufmann Publishers (2005)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA (2003)
4. Baader, F., Hanschke, P.: A scheme for integrating concrete domains into concept languages. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1. pp. 452–457. IJCAI'91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1991), http://dl.acm.org/citation.cfm?id=1631171.1631239
5. Baader, F., Horrocks, I., Sattler, U.: Description Logics. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, chap. 3, pp. 135–180. Elsevier (2008), download/2007/BaHS07a.pdf
6. Baader, F., Nutt, W.: Basic Description Logics. In: Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.) The Description Logic Handbook: Theory, Implementation and Applications, chap. 2, pp. 43–95. Cambridge University Press, New York, NY, USA (2003), http://dl.acm.org/citation.cfm?id=885746.885749
7. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69(1), 5–40 (2001), http://www.springerlink.com/index/j245221867285361.pdf
8. Baader, F., Sattler, U.: Description logics with aggregates and concrete domains. Information Systems 28(8), 979–1004 (2003)
9. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability Modulo the Theory of Costs: Foundations and Applications. In: Esparza, J., Majumdar, R. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 6015, pp. 99–113. Springer Berlin Heidelberg (2010)
10. Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: OWL 2: The next step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web 6(4), 309–322 (2008)
11. Haarslev, V., Sebastiani, R., Vescovi, M.: Automated Reasoning in $\mathcal{ALCQ}$ via SMT. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Automated Deduction – CADE-23, Lecture Notes in Computer Science, vol. 6803, pp. 283–298. Springer Berlin Heidelberg (2011)
12. Hoos, H.H.: Satlib - benchmark problems (2011), http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html
13. Horrocks, I., Patel-Schneider, P.F.: Dl systems comparison. In: Proc. of the 1998 Description Logic Workshop (DL'98). vol. 11, pp. 55–57 (1998)
14. Kang, Y.B., Li, Y.F., Krishnaswamy, S.: Predicting reasoning performance using ontology metrics. In: The Semantic Web–ISWC 2012, pp. 198–214. Springer (2012)
15. Kang, Y.B., Pan, J.Z., Krishnaswamy, S., Sawangphol, W., Li, Y.F.: How long will it take? accurate prediction of ontology reasoning performance. In: AAAI. pp. 80–86 (2014)
16. Lutz, C., Areces, C., Horrocks, I., Sattler, U.: Keys, nominals, and concrete domains. In: IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003. pp. 349–354 (2003)

17. Meissner, A.: The $\mathcal{ALCN}$ description logic concept satisfiability as a sat problem. In: Katarzyniak, R., Chiu, T.F., Hong, C.F., Nguyen, N. (eds.) Semantic Methods for Knowledge Management and Communication, Studies in Computational Intelligence, vol. 381, pp. 253–263. Springer Berlin Heidelberg (2011)
18. Motik, B., Shearer, R., Horrocks, I.: A hypertableau calculus for $\mathcal{SHIQ}$. In: Proc. of the 2007 Description Logic Workshop (DL 2007). CEUR (`http://ceur-ws.org/`), vol. 250 (2007), `download/2007/MoSH07b.pdf`
19. Motik, B., Shearer, R., Horrocks, I.: Optimized reasoning in description logics using hypertableaux. In: Proc. of the 21st Int. Conf. on Automated Deduction (CADE-21). Lecture Notes in Artificial Intelligence, vol. 4603, pp. 67–83. Springer (2007), `download/2007/MoSH07a.pdf`
20. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. of Artificial Intelligence Research 36, 165–228 (2009), `download/2009/MoSH09a.pdf`
21. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming – CP 2007, Lecture Notes in Computer Science, vol. 4741, pp. 529–543. Springer Berlin Heidelberg (2007)
22. Pan, J.Z.: A flexible ontology reasoning architecture for the semantic web. Knowledge and Data Engineering, IEEE Transactions on 19(2), 246–260 (2007)
23. Rector, A.L., Rogers, J.E., Pole, P.A.: The GALEN High Level Ontology. Proceedings MIE 96 pp. 174–178 (Jan 1996)
24. Samwald, M.: Genomic CDS: an example of a complex ontology for pharmacogenetics and clinical decision support. In: 2nd OWL Reasoner Evaluation Workshop (ORE 2013). p. 128 (2013)
25. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements,. Artificial Intelligence 48, 1–26 (1991)
26. Shearer, R., Motik, B., Horrocks, I.: HermiT: A Highly-Efficient OWL Reasoner. In: Ruttenberg, A., Sattler, U., Dolbear, C. (eds.) Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU). Karlsruhe, Germany (October 26–27 2008)
27. Stearns, M.Q., Price, C., Spackman, K.A., Wang, A.Y.: Snomed clinical terms: overview of the development process and project status. In: Proceedings of the AMIA Symposium. p. 662. American Medical Informatics Association (2001)
28. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: System description. Journal of Web Semantics (JWS) 27, 78–85 (2014)
29. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: Furbach, U., Shankar, N. (eds.) Automated Reasoning, Lecture Notes in Computer Science, vol. 4130, chap. 26, pp. 292–297. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
30. Zuo, M., Haarslev, V.: Intelligent tableau algorithm for dl reasoning. In: Automated Reasoning with Analytic Tableaux and Related Methods, pp. 273–287. Springer (2013)