

# The CEL Manual

(*version 1.0*)

Boontawee Suntisrivaraporn

Institute for Theoretical Computer Science  
TU Dresden, Germany  
*meng@tcs.inf.tu-dresden.de*

## Abstract

Description logics (DLs) are an important family of formalisms for reasoning about ontologies. CEL<sup>1</sup> is a reasoner for the description logic  $\mathcal{EL}^+$ , supporting as its main reasoning task the computation of the subsumption hierarchy induced by  $\mathcal{EL}^+$  ontologies. The most distinguishing feature of CEL is that, unlike other modern DL reasoners, it implements a tractable (i.e., polynomial-time) algorithm. The supported description logic  $\mathcal{EL}^+$  offers a selected set of expressive means that are tailored towards the formulation of medical and biological ontologies.

## Contents

<b>1</b>	<b>Introduction to CEL</b>	<b>3</b>
<b>2</b>	<b>An Overview of CEL Ontologies</b>	<b>4</b>
<b>3</b>	<b>Keywords, Macros and Functions</b>	<b>6</b>
3.1	Concept and role construction . . . . .	6
3.2	Knowledge base declarations and axioms . . . . .	7
3.2.1	Concepts . . . . .	8
3.2.2	Roles . . . . .	10
3.2.3	Individuals . . . . .	12
3.3	Ontology management operations . . . . .	12
3.4	Query answering operations . . . . .	15
3.4.1	Ontology, TBox, Abox . . . . .	15
3.4.2	Concepts . . . . .	17
3.4.3	Roles . . . . .	19
3.4.4	Individuals . . . . .	21
3.5	Other operations . . . . .	22
3.5.1	Ouputs . . . . .	22
3.5.2	DIG . . . . .	24
3.5.3	Information and utilities . . . . .	24

---

<sup>1</sup>which stands for a polynomial-time Classifier for the description logic  $\mathcal{EL}^+$

<b>4</b>	<b>Command Line Options</b>	<b>25</b>
<b>5</b>	<b>An Example: Learning by Doing</b>	<b>26</b>
5.1	Complex subsumption queries . . . . .	28
5.2	Incremental Classification . . . . .	28
	<b>References</b>	<b>28</b>
	<b>Index</b>	<b>29</b>

# 1 Introduction to CEL

The description logic  $\mathcal{EL}^{++}$  introduced in [1] is tailored towards the formulation of medical and biological ontologies. A distinguishing feature of  $\mathcal{EL}^{++}$  compared to other description logics such as  $\mathcal{SHIQ}$  and OWL [6, 7] is that, despite offering considerable expressivity, reasoning in  $\mathcal{EL}^{++}$  is tractable, i.e., can be performed in polynomial time. This is clearly an advantage over the EXPTIME worst-case complexity of reasoning in  $\mathcal{SHIQ}$  and OWL.<sup>2</sup>

The ultimate goal of CEL is to provide a highly efficient reasoner for the description logic  $\mathcal{EL}^{++}$ . A main emphasis of CEL is on reasoning with ontologies formulated in  $\mathcal{EL}^{++}$ , in particular on computing the subsumption hierarchy induced by such an ontology. As of now, CEL implements a practically useful subset of  $\mathcal{EL}^{++}$  that we call  $\mathcal{EL}^+$ . More precisely,  $\mathcal{EL}^+$  provides for the following concept constructors:

- top concept “ $\top$ ”;
- bottom concept “ $\perp$ ”, which boils down to the expressivity of concept disjointness axioms (see below);
- conjunction “ $C \sqcap D$ ”;
- existential restriction “ $\exists r.C$ ”.

For building up ontologies, the following means of expressivity are available:

- primitive concept definitions “ $A \sqsubseteq D$ ”;
- concept definitions “ $A \equiv D$ ”;
- general concept inclusion (GCI) axioms “ $C \sqsubseteq D$ ”;
- concept equivalence axioms “ $C \equiv D$ ”;
- concept disjointness axioms “ $C_1 \sqcap \dots \sqcap C_n \sqsubseteq \perp$ ”;
- role domain restrictions “ $\exists r.\top \sqsubseteq C$ ”;
- role range restrictions<sup>3</sup> “ $\top \sqsubseteq \forall r.C$ ”;
- reflexivity assertions for roles “ $\epsilon \sqsubseteq r$ ”;
- transitivity assertions for roles “ $r \circ r \sqsubseteq r$ ”;
- role hierarchy axioms “ $r \sqsubseteq s$ ”;
- right-identity rules “ $r \circ s \sqsubseteq r$ ” and left-identity rules “ $r \circ s \sqsubseteq s$ ”;

---

<sup>2</sup>The logic underlying CEL cannot be fully expressed in OWL due to the presence of role inclusions. This has however been addressed in the ongoing effort of OWL 1.1 [?].

<sup>3</sup>Two remarks are in order: (i) in general, value restrictions are not allowed; and (ii) undecidability may incur when range restrictions are used in conjunction with role inclusions.

- role inclusion (RI) axioms of the form “ $r \circ s \sqsubseteq t$ ”.
- concept assertions for individuals “ $C(ind)$ ”;
- role assertions for individuals “ $r(ind_1, ind_2)$ ”;

The ontology classification algorithm implemented in CEL has been first proposed in [3] which has been enhanced further (see [2, 4]). This algorithm is a refinement for implementation purposes of the well-known polynomial-time algorithm for computing classification in  $\mathcal{EL}^{++}$  given in [1]. Currently, CEL accepts inputs in a slight extension of the KRSS (Knowledge Representation System Specification) syntax [8]. The following system requirements are assumed:

- Linux operating system;
- Physical memory at least 128MB.<sup>4</sup>

CEL is available as a precompiled binary package which can be run on Linux platforms. The package can be obtained from:

<http://lat.inf.tu-dresden.de/systems/cel/>

The package consists of the CEL executable, this user manual, the related papers [1, 3], and some toy  $\mathcal{EL}^+$  ontologies.

This manual details how to get CEL up and running, how to create an ontology, and how to classify it. In Section 2, an overview of the concept and ontology language of CEL is given. In Section 3, all commands offered by CEL will be described in detail, including syntax and helpful examples. Finally, Section 5 gives a step-by-step introduction to CEL at work. This includes demonstrations of how to query complex subsumption relationships (before classification) and how to do incremental classification which are the most prominent new features of version 1.0.

## 2 An Overview of CEL Ontologies

Concepts and roles are built up from sets of concept and role names (which can be freely chosen by the user), using the concept and role constructors provided by  $\mathcal{EL}^+$ . The following syntax rules describe the formation of concepts and roles in CEL:

$$C \longrightarrow CN \mid \text{top} \mid \text{bottom} \mid (\text{and } C_1 \dots C_n) \mid (\text{some } RN \ C)$$

$$R \longrightarrow RN \mid (\text{compose } RN_1 \ RN_2)$$

where  $CN$ ,  $C$ ,  $RN$ , and  $R$  (all possibly with subscripts) range over concept names, concepts, role names, and roles, respectively.

A more detailed description of the syntax along with helpful examples is given in Section 3.1.

We now give an overview of the axioms that are available for building up ontologies. Such axioms can be divided into concept axioms and role axioms.

---

<sup>4</sup>Considerably more memory may be needed for larger ontologies.

**Primitive concept definitions** state the necessary condition of a (*primitively defined*) concept name.

DL notation:  $CN \sqsubseteq C$

CEL syntax: (define-primitive-concept  $CN C$ )

**Concept definitions** state the definition (both necessary and sufficient conditions) of a (*fully defined*) concept name.

DL notation:  $CN \equiv C$

CEL syntax: (define-concept  $CN C$ )

**General concept inclusions** state the subsumption (or containment) between two concept descriptions.

DL notation:  $C_1 \sqsubseteq C_2$

CEL syntax: (implies  $C_1 C_2$ )

**Concept equivalence axioms** state the equivalence between two concept descriptions.

DL notation:  $C_1 \equiv C_2$

CEL syntax: (equivalence  $C_1 C_2$ )

**Concept disjointness axioms** assert the disjointness of two or more concept descriptions.

DL notation:  $C_1 \sqcap \dots \sqcap C_n \sqsubseteq \perp$

CEL syntax: (disjoint  $C_1 \dots C_2$ )

**Role domain restrictions** assert the domain of a role name

DL notation:  $\exists RN.\top \sqsubseteq C$

CEL syntax: (domain  $RN C$ )

**Role range restrictions** assert the range of a role name

DL notation:  $\top \sqsubseteq \forall RN.C$

CEL syntax: (range  $RN C$ )

**Role reflexivity axioms** assert the reflexivity of a role name

DL notation:  $\epsilon \sqsubseteq RN$

CEL syntax: (reflexive  $RN$ )

**Role transitivity axioms** assert the transitivity of a role name

DL notation:  $RN \circ RN \sqsubseteq RN$

CEL syntax: (transitive  $RN$ )

**Role hierarchies** assert the relationship between a role name and its super role name.

DL notation:  $RN_1 \sqsubseteq RN_2$

CEL syntax: (define-primitive-role  $RN_1$  :parent  $RN_2$ )

**Right-identity (left-identity) rules** specifies a right (left) identity of another role.

DL notation:  $RN_1 \circ RN_2 \sqsubseteq RN_1$

CEL syntax: (define-primitive-role  $RN_1$  :right-identity  $RN_2$ ); and,

DL notation:  $RN_1 \circ RN_2 \sqsubseteq RN_2$

CEL syntax: (define-primitive-role  $RN_2$  :left-identity  $RN_1$ )

**Role inclusions** state the subsumption (or containment) between a role composition on the lhs and a role name on the right-hand side.

DL notation:  $RN_1 \circ RN_2 \sqsubseteq RN_3$

CEL syntax: (role-inclusion (compose  $RN_1$   $RN_2$ )  $RN_3$ )

**Concept assertions** state the instantiation relationship of an individual by a concept.

DL notation:  $C(IND)$

CEL syntax: (instance  $IND$   $C$ )

**Role assertions** state that two individuals are related by a specified role name.

DL notation:  $RN(IND_1, IND_2)$

CEL syntax: (related  $IND_1$   $IND_2$   $RN$ )

We note that an ontology axiom may be formulated in different ways. For instance, to say that `Human` is the domain of `child`, we could write (domain child Human), or integrate this facet in a more coarse-grained axiom (define-primitive-role child ... :domain Human ...), or even formulate as a concept inclusion (implies (some child top) Human). In fact, only `role-inclusion` and `implies` are sufficient to express most  $\mathcal{EL}^+$  ontologies, with exceptions of range restrictions, concept and role assertions. The supplemental commands are included for ease of use and for compatibility with other DL reasoners.

### 3 Keywords, Macros and Functions

In this section, we give a detailed description of all commands offered by CEL. The commands are arranged in five groups.

#### 3.1 Concept and role construction

In CEL, concept and role names can be used without prior declaration. To form concept and role names, we recommend the use of alphanumerical characters, probably including symbols such as “\_” and “-”. Per default, CEL does not distinguish capital letters from lower-case ones. All symbols will automatically be capitalized when processing. In case of the need for case distinction, symbols shall be embraced within a pair of vertical bars, that is, `MALE` is equivalent to `Male`, but `|MALE|` is not to `|Male|`.

Complex concepts and roles can be constructed using the constructors of  $\mathcal{EL}^+$  as summarized in the grammar on Page 3. In the following, the syntax is described in more detail.

---

**top** *keyword*

---

**Description:** The predefined, most general concept of any ontology, the top concept ( $\top$ , aka `owl:Thing` in the OWL lingo).  
**Syntax:** `top`  
**Type:** concept constructor

---

**bottom** *keyword*

---

**Description:** The predefined, most specific concept of any ontology, the bottom concept ( $\perp$ , aka `owl:Nothing` in the OWL lingo).  
**Syntax:** `top`  
**Type:** concept constructor

---

**and** *keyword*

---

**Description:**  $N$ -ary conjunction ( $C_1 \sqcap \dots \sqcap C_n$ )  
**Syntax:** `(and  $C_1 \dots C_n$ )`  
**Example:** `(and Human Male Drunk)`  
**Type:** concept constructor

---

**some** *keyword*

---

**Description:** Existential restriction ( $\exists RN.C$ )  
**Syntax:** `(some  $RN C$ )`  
**Example:** `(some has-child Male)`  
**Type:** concept constructor  
**Remarks:** Only atomic roles are allowed in an existential restriction

---

**compose** *keyword*

---

**Description:** Binary role composition ( $RN_1 \circ RN_2$ )  
**Syntax:** `(compose  $RN_1 RN_2$ )`  
**Example:** `(compose has-parent has-brother)`  
**Type:** role constructor  
**Remarks:** Only allowed on the left-hand side of role-inclusion

### 3.2 Knowledge base declarations and axioms

This section specifies the commands and their syntax that are used to build up CEL ontologies. There are five kinds of concept declarations and axioms; six kinds of role declarations and restrictions; and finally, two kinds of assertions for individuals.

### 3.2.1 Concepts

---

[define-primitive-concept](#) *macro*

---

**Description:** Asserts a subsumption relation between a concept name and a complex concept.

**Syntax:** `(define-primitive-concept CN &optional C)`

**Arguments:** *CN* - concept name  
*C* - (possibly complex) concept

**Examples:** `(define-primitive-concept Father Man)`  
`(define-primitive-concept Professor (and Lecturer (some supervises PhDStudent)))`

**Remarks:** This macro states the necessary conditions for *CN*.

---

[define-concept](#) *macro*

---

**Description:** Defines a concept name.

**Syntax:** `(define-concept CN C)`

**Arguments:** *CN* - concept name  
*C* - (possibly complex) concept

**Examples:** `(define-concept Father (and Man (some has-child Human)))`

**Remarks:** This macro states both the necessary and sufficient conditions for *CN*. In CEL, a concept name may have multiple definitions.

---

[concept-inclusion , implies](#) *macro*

---

**Description:** Asserts a general concept inclusion (GCI) between two concepts.

**Syntax:** `(implies C1 C2)`

**Arguments:** *C<sub>1</sub>, C<sub>2</sub>* - (possibly complex) concepts

**Examples:** `(implies (and Man (some has-sibling Parent)) Uncle)`

**Remarks:** The equivalent DL notation is  $C_1 \sqsubseteq C_2$ .

---

[equivalent](#) *macro*

---

**Description:** Asserts an equivalence between two concepts.

**Syntax:** `(equivalent C1 C2)`

**Arguments:** *C<sub>1</sub>, C<sub>2</sub>* - (possibly complex) concepts

**Examples:** `(equivalent (and Man (some has-sibling (some has-child Female)) (and Uncle (some has-niece top))))`

**Remarks:** The equivalent DL notation is  $C_1 \equiv C_2$ .

**Description:** Asserts that specified concepts are pairwise disjoint.

**Syntax:** (equivalent  $C_1 \dots C_n$ )

**Arguments:**  $C_i$  - (possibly complex) concepts

**Examples:** (disjoint UnderGraduateStudent GraduateStudent (some full-time-employed-by top))

**Remarks:** The equivalent DL notation is  $C_1 \sqcap \dots \sqcap C_n \sqsubseteq \perp$ .



**Description:** Asserts an inclusion between a (possibly composite) role and a role name.

**Syntax:** `(role-inclusion R RN)`

**Arguments:** *R* - role name or binary role composite  
*RN* - role name

**Example:** `(role-inclusion has-daughter has-child)`  
`(role-inclusion (compose has-father has-sister) has-aunt)`

**Remarks:** This is a generalization of `define-primitive-role`. Role hierarchy, transitivity, and right-identity can be expressed in a more elegant way by using `define-primitive-role`. This macro is not part of the KRSS standard.

---

[reflexive](#) *macro*

---

**Description:** Reflexivity assertion for a role name

**Syntax:** `(reflexive RN)`

**Arguments:** *RN* - role name

**Example:** `(reflexive loves)`

---

[transitive](#) *macro*

---

**Description:** Transitivity assertion for a role name

**Syntax:** `(transitive RN)`

**Arguments:** *RN* - role name

**Example:** `(transitive has-ancestor)`

---

[domain](#) *macro*

---

**Description:** Asserts the domain of a role name

**Syntax:** `(domain RN C)`

**Arguments:** *RN* - role name  
*C* - (possibly complex) concept

**Example:** `(domain supervises Professor)`

---

[range](#) *macro*

---

**Description:** Asserts the range of a role name

**Syntax:** `(range RN C)`

**Arguments:** *RN* - role name  
*C* - (possibly complex) concept

**Example:** `(range supervises PhDStudent)`

### 3.2.3 Individuals

---

[instance](#) *macro*

---

**Description:** Asserts instantiation between an individual and a concept.  
**Syntax:** `(instance IND C)`  
**Arguments:** *IND* - individual name  
*C* - (possibly complex) concept  
**Examples:** `(instance FRANZ (and Professor (some teaches DL)  
(some supervises GraduateStudent) (some supervises  
UnderGraduateStudent)))`  
**Remarks:** *IND* must not occur as a concept name in the ontology, i.e., the sets of concept names and individual names must be disjoint.

---

[related](#) *macro*

---

**Description:** Defines a concept name.  
**Syntax:** `(related IND1 IND2 RN)`  
**Arguments:** *IND*<sub>*i*</sub> - individual name  
*RN* - role name  
**Examples:** `(related FRANZ MENG supervises)`  
**Remarks:** *IND*<sub>*i*</sub> must not occur as concept names in the ontology

As pointed out in the last two axioms, the set of individuals must be disjoint with that of concept names. However, there is no such restriction on the use of role names, i.e., a role name may occur in the ontology as a concept name or as an individual. This means that punning for role names is allowed by the CEL system.

### 3.3 Ontology management operations

Starting from version 1.0, CEL supports multiple ontology management as compliant to the DIG interface specification [5]. Each ontology is equipped with and identified by a unique URI. The following macros can be used to create a new ontology, release or restore an existing one.

---

[create-ontology](#), [create-tbox](#) *macro*

---

**Description:** Creates a new ontology and assign a (user specified or system generated) URI to it.  
**Syntax:** `(create-ontology &optional URI)`  
**Arguments:** *URI* - string or symbol representing a unique uniform resource identifier for the new ontology. If not specified, the system will generate a unique one.  
**Examples:** `(create-ontology "urn:cel-ontology:med.tbox")`  
**Remarks:** This macro is internally invoked when calling `(start)` with an optional *URI* set to `t` or a string.

---

[clear-ontology](#), [clear-tbox](#) *macro*

---

**Description:** Removes all axioms and initializes everything.  
**Syntax:** (clear-tbox &optional *URI*)  
**Arguments:** *URI* - string or symbol representing a unique uniform resource identifier for the new ontology. If not specified, the currently active ontology will be cleared.  
**Remarks:** This macro is internally invoked for the default ontology when calling (start) with an optional *URI* set to default or "urn:cel-ontology:default"

---

[release-ontology](#), [release-tbox](#) *macro*

---

**Description:** Releases the URI-specified ontology or the current one if none given.  
**Syntax:** (release-ontology &optional *URI*)  
**Arguments:** *URI* - string or symbol representing a unique uniform resource identifier for the new ontology. If not specified, the currently active one is considered.  
**Remarks:** The default ontology cannot be released.

---

[release-all-ontologies](#), [release-all-tboxes](#) *macro*

---

**Description:** Releases all ontologies but the default  
**Syntax:** (release-all-ontologies)

---

[restore-ontology](#), [restore-tbox](#) *macro*

---

**Description:** Restores the URI-specified ontology and thereby makes it active.  
**Syntax:** (restore-ontology *URI*)  
**Arguments:** *URI* - string or symbol representing a unique uniform resource identifier for the new ontology.  
**Examples:** (restore-ontology default)  
**Remarks:** Nothing happens if the specified URI does not exist.

---

[repository](#) *macro*

---

**Description:** Displays all ontologies, their state and some of their properties. Symbol \* marks the active ontology.  
**Syntax:** (repository)  
**Remarks:** Use (detail-ontology) for a much more verbose description of the current ontology.

All the following macros work on the currently *active* ontology. When an operation is to perform on a specific ontology, make sure that the ontology is active by calling (restore-ontology) with its designated URI prior to calling the operation of interest.

---

**load-ontology , load-tbox***macro*

---

- Description:** Reads in and and normalizes the input ontology.  
**Syntax:** (load-ontology *tbxfile*)  
**Arguments:** *tbxfile* - path of the input ontology file. If a filename is given without path, the ontology is assumed to be either in the current directory, or in the subdirectory `./tboxes`.  
**Remarks:** The file specified by `tbox-file` is assumed to contain a list of commands as shown in Subsection 3.2. After having successfully loaded the ontology, it is ready to be classified, see the next command.

---

**classify-ontology , classify-tbox***macro*

---

- Description:** Classifies the preprocessed ontology.  
**Syntax:** (classify-ontology &key (*taxonomize* nil))  
**Arguments:** *taxonomize* - if set to `t`, the concept hierarchy (DAG) will be created after classification is done.  
**Remarks:** Constructing the concept hierarchy is relatively cheap given the fact that classification is completed, thus this procedure is carried out whenever required certain queries.

---

**detail-ontology , detail-tbox***macro*

---

- Description:** Displays detailed description of the current ontology.  
**Syntax:** (detail-ontology)

As a means to support incremental classification, each ontology is equipped with a secondary TBox which will be called *temporary* TBox or TTBox, in what follows. The TTBox can be activated, deactivated, classified, etc. The rest of this subsection devotes to the specifications of these macros.

---

**activate-ttbox***macro*

---

- Description:** Activates TTBox for incremental classification  
**Syntax:** (activate-ttbox)  
**Remarks:** The primary ontology has to be fully classified. After invocation of this macro, concept declarations and axioms from Subsection 3.2 can be asserted again. Retraction of all *incremental* axioms is also possible by calling (deactivate-ttbox).

---

**clear-ttbox***macro*

---

- Description:** Removes all *incremental* axioms and their consequences  
**Syntax:** (clear-ttbox)

---

[commit-ttbox](#) *macro*

---

**Description:** Commits to the primary ontology changes due to *incremental* axioms and their consequences. After it successfully terminates, TTBox is deactivated.

**Syntax:** (commit-ttbox)

**Remarks:** Compare with (deactivate--ttbox)

---

[deactivate-ttbox](#) *macro*

---

**Description:** Deactivates TTBox and thereby retract all *incremental* axioms from the knowledge base

**Syntax:** (deactivate-ttbox)

**Remarks:** Compare with (commit-ttbox)

---

[classify-ttbox](#) *macro*

---

**Description:** Classifies the temporary TBox against the ontology.

**Syntax:** (classify-ttbox &key (commit nil))

**Arguments:** *:commit* - if set to `t`, the *incremental* classification results will be committed, and hence not reversable. This is defaulted to `nil`.

**Remarks:** After an incremental classification, additional consequences can be committed through (commit-ttbox) or discarded through (deactivate-ttbox). This provides a weak mechanism for axiom retraction, meaning precisely that axioms in TTBox can be retracted.

### 3.4 Query answering operations

#### 3.4.1 Ontology, TBox, Abox

---

[ontology-prepared? , tbox-prepared?](#) *macro*

---

**Description:** Checks if the ontology is prepared for classification.

**Syntax:** (ontology-prepared?)

**Remarks:** This is a fundamental requirement for (classify-ontology).

---

[ontology-classified? , tbox-classified?](#) *macro*

---

**Description:** Checks if the ontology is successfully classified.

**Syntax:** (ontology-classified?)

**Remarks:** This test must be positive before any subsumption queries can be carried out.

---

[ontology-consistent?](#) *macro*

---

**Description:** Checks if the ontology is consistent.  
**Syntax:** (ontology-consistent?)  
**Remarks:** The ontology must be classified before. For more fine-grained inconsistency inspection, see (tbox-consistent?), (abox-consistent?) and (satisfiable?).

---

[tbox-consistent?](#) *macro*

---

**Description:** Checks if the intentional part of the ontology (TBox) is consistent.  
**Syntax:** (tbox-consistent?)  
**Remarks:** The ontology must be classified before.

---

[abox-consistent?](#) *macro*

---

**Description:** Checks if the extentional part of the ontology (ABox) is consistent.  
**Syntax:** (abox-consistent?)  
**Remarks:** The ontology must be classified before.

---

[ttbox-active?](#) *macro*

---

**Description:** Checks if the *temporary* TBox is active.  
**Syntax:** (ttbox-active?)

---

[ttbox-cleared?](#) *macro*

---

**Description:** Checks if the *temporary* TBox is empty or cleared.  
**Syntax:** (ttbox-cleared?)

---

[ttbox-prepared?](#) *macro*

---

**Description:** Checks if the *temporary* TBox is prepared for incremental classification, i.e., all additional axioms are preprocessed.  
**Syntax:** (ttbox-prepared?)

---

[ttbox-classified?](#) *macro*

---

**Description:** Checks if the *temporary* TBox is incrementally classified against the main ontology.  
**Syntax:** (ttbox-classified?)

### 3.4.2 Concepts

---

<a href="#">concept?</a>	<i>macro</i>
<b>Description:</b> Checks if the given name is known as a concept name.	
<b>Syntax:</b> (concept? <i>CN</i> )	
<b>Arguments:</b> <i>CN</i> - name to be tested	

---

<a href="#">all-concepts</a>	<i>macro</i>
<b>Description:</b> Retrieves the set of all concept names occurring in the ontology.	
<b>Syntax:</b> (all-concepts)	

---

<a href="#">satisfiable? , concept-satisfiable?</a>	<i>macro</i>
<b>Description:</b> Checks if the given concept name is satisfiable w.r.t. the ontology.	
<b>Syntax:</b> (satisfiable? <i>CN</i> )	
<b>Arguments:</b> <i>CN</i> - concept name	
<b>Remarks:</b> Equivalent to asking (subsumes? bottom <i>CN</i> )	

---

<a href="#">all-unsatisfiable-concepts</a>	<i>macro</i>
<b>Description:</b> Retrieves the set of all unsatisfiable concept names occurring in the ontology.	
<b>Syntax:</b> (all-unsatisfiable-concepts)	
<b>Remarks:</b> This macro allows an application to make a single query invocation to check satisfiability of all concept names in the ontology.	

---

<a href="#">concept-subsumes? , subsumes?</a>	<i>macro</i>
<b>Description:</b> Queries if one concept name subsumes the other.	
<b>Syntax:</b> (subsumes? <i>CN</i> <sub>1</sub> <i>CN</i> <sub>2</sub> )	
<b>Arguments:</b> <i>CN</i> <sub>1</sub> , <i>CN</i> <sub>2</sub> - concept names	
<b>Remarks:</b> This macro only performs a lookup in the preclassified concept hierarchy; it cannot be called before the test (tbody-classified?) is positive. DL equivalent notation is $CN_2 \sqsubseteq_{\mathcal{T}}^? CN_1$ . See (?subsumes) for a more powerful variant of this query.	

---

**?subsumes***macro*

---

- Description:** Queries if one concept subsumes the other.  
**Syntax:** (?subsumes  $C_1$   $C_2$ )  
**Arguments:**  $C_1, C_2$  - well-formed (possibly complex)  $\mathcal{EL}^+$  concepts  
**Examples:** (?subsumes (and (some child Rich)  
(some child Gorgeous))  
(some child (and Rich Gorgeous)))  
**Remarks:** DL equivalent notation is  $C_2 \sqsubseteq_{\mathcal{T}}^? C_1$ . This macro could be used either *before* or after classification (thus question-mark-prefixed name). Note that if the ontology is classified and the two arguments are atomic, this operation boils down to the suffixed variant, i.e., (subsumes?).

---

**concept-implies? , implies?***macro*

---

- Description:** Queries if one concept name implies (is subsumed by) the other.  
**Syntax:** (implies?  $CN_1$   $CN_2$ )  
**Arguments:**  $CN_1, CN_2$  - concept names  
**Remarks:** Identical to (subsumes?), but with swapped arguments  
See (?implies) for a more powerful variant of this query.

---

**?implies***macro*

---

- Description:** Queries if one concept implies (is subsumed by) the other.  
**Syntax:** (concept-implies?  $C_1$   $C_2$ )  
**Arguments:**  $C_1, C_2$  - well-formed (possibly complex)  $\mathcal{EL}^+$  concepts  
**Remarks:** Identical to (?subsumes), but with swapped arguments.

---

**concept-equivalent? , equivalent?***macro*

---

- Description:** Queries if two concept names are equivalent (i.e., subsume each other)  
**Syntax:** (concept-equivalent?  $CN_1$   $CN_2$ )  
**Arguments:**  $CN_1, CN_2$  - concept names

---

**parents , parent-concepts***macro*

---

- Description:** Returns the set of *direct* subsumers of a concept name.  
**Syntax:** (parents  $CN$ )  
**Arguments:**  $CN$  - concept name  
**Remarks:** This feature is available only after taxonomization. If the concept hierarchy has not been constructed before, it will be after a successful call to this operation.

---

**children , child-concepts**

---

*macro*

- Description:** Returns the set of *direct* subsumees of a concept name.  
**Syntax:** (children *CN*)  
**Arguments:** *CN* - concept name  
**Remarks:** This feature is available only after taxonomization. If the concept hierarchy has not been constructed before, it will be after a successful call to this operation.

---

**ancestors , super-concepts**

---

*macro*

- Description:** Returns the set of all (direct and indirect) subsumers of a concept name.  
**Syntax:** (ancestors *CN*)  
**Arguments:** *CN* - concept name

---

**descendants , sub-concepts**

---

*macro*

- Description:** Returns the set of all (direct and indirect) subsumees of a concept name.  
**Syntax:** (descendants *CN*)  
**Arguments:** *CN* - concept name  
**Remarks:** This feature is available only after taxonomization. If the concept hierarchy has not been constructed before, it will be after a successful call to this operation.

---

**equivalents**

---

*macro*

- Description:** Returns the set of all equivalent concept names including explicitly told synonyms.  
**Syntax:** (equivalents *CN*)  
**Arguments:** *CN* - concept name  
**Remarks:** This feature is available only after taxonomization. If the concept hierarchy has not been constructed before, it will be after a successful call to this operation.

### 3.4.3 Roles

---

**role?**

---

*macro*

- Description:** Checks if the given name is known as a role name.  
**Syntax:** (concept? *RN*)  
**Arguments:** *RN* - name to be tested

<a href="#">all-roles</a>	<i>macro</i>
<p><b>Description:</b> Retrieves the set of all role names occurring in the ontology.</p> <p><b>Syntax:</b> (all-roles)</p>	
<a href="#">role-subsumes?</a>	<i>macro</i>
<p><b>Description:</b> Queries if one role name subsumes the other.</p> <p><b>Syntax:</b> (role-subsumes? <math>RN_1</math> <math>RN_2</math>)</p> <p><b>Arguments:</b> <math>RN_1, RN_2</math> - role names</p> <p><b>Remarks:</b> DL equivalent notation is <math>RN_2 \sqsubseteq_T^? RN_1</math>.</p>	
<a href="#">role-implies?</a>	<i>macro</i>
<p><b>Description:</b> Queries if one role name implies (is subsumed by) the other.</p> <p><b>Syntax:</b> (role-implies? <math>RN_1</math> <math>RN_2</math>)</p> <p><b>Arguments:</b> <math>RN_1, RN_2</math> - role names</p> <p><b>Remarks:</b> Identical to <i>role-subsumes?</i>, but with swapped arguments</p>	
<a href="#">reflexive?</a>	<i>macro</i>
<p><b>Description:</b> Checks if a role name is reflexive.</p> <p><b>Syntax:</b> (reflexive? <math>RN</math>)</p> <p><b>Arguments:</b> <math>RN</math> - role name</p>	
<a href="#">transitive?</a>	<i>macro</i>
<p><b>Description:</b> Checks if a role name is specified as transitive.</p> <p><b>Syntax:</b> (transitive? <math>RN</math>)</p> <p><b>Arguments:</b> <math>RN</math> - role name</p>	
<a href="#">parent-roles</a>	<i>macro</i>
<p><b>Description:</b> Returns the set of <i>direct</i> super-roles of a role name.</p> <p><b>Syntax:</b> (parent-roles <math>RN</math>)</p> <p><b>Arguments:</b> <math>RN</math> - role name</p> <p><b>Remarks:</b> Not yet supported.</p>	
<a href="#">child-roles</a>	<i>macro</i>
<p><b>Description:</b> Returns the set of <i>direct</i> sub-roles of a role name.</p> <p><b>Syntax:</b> (child-roles <math>RN</math>)</p> <p><b>Arguments:</b> <math>RN</math> - role name</p> <p><b>Remarks:</b> Not yet supported.</p>	

---

[super-roles](#) *macro*

---

**Description:** Returns the set of all super-roles of a role name.  
**Syntax:** (super-roles *RN*)  
**Arguments:** *CN* - role name

---

[sub-roles](#) *macro*

---

**Description:** Returns the set of all sub-roles of a role name.  
**Syntax:** (sub-roles *RN*)  
**Arguments:** *CN* - role name

### 3.4.4 Individuals

---

[individual?](#) *macro*

---

**Description:** Checks if the given name is known as an individual name.  
**Syntax:** (individual? *IND*)  
**Arguments:** *IND* - name to be tested

---

[all-individuals](#) *macro*

---

**Description:** Retrieves the set of all individuals occurring in the ontology.  
**Syntax:** (all-individuals)

---

[instance?](#) *macro*

---

**Description:** Queries if the individual is an instance of the concept name.  
**Syntax:** (instance? *IND* *CN*)  
**Arguments:** *IND* - individual  
*CN* - concept name  
**Example:** (instance? MENG GraduateStudent)

---

[individual-direct-types](#) *macro*

---

**Description:** Returns the set of *direct* types of the given individual, i.e., most specific concept names of which the individual is an instance.  
**Syntax:** (individual-direct-types *IND*)  
**Arguments:** *IND* - individual  
**Remarks:** This feature is available only after taxonomization. If the concept hierarchy has not been constructed before, it will be after a successful call to this operation.

---

[individual-types](#) *macro*

---

**Description:** Returns the set of all types of the given individual, i.e., all concept names of which the individual is an instance.

**Syntax:** (`individual-types` *IND*)

**Arguments:** *IND* - individual

---

[concept-instances](#) *macro*

---

**Description:** Returns the set of all individuals belonging to the specified concept name.

**Syntax:** (`concept-instances` *CN*)

**Arguments:** *CN* - concept name

**Remarks:** This feature is available only after taxonomization. If the concept hierarchy has not been constructed before, it will be after a successful call to this operation.

---

## 3.5 Other operations

### 3.5.1 Outputs

---

[output-subsumption](#) *macro*

---

**Description:** Outputs all pairs of (told and inferred) subsumption relationships between concept names in the ontology.

**Syntax:** (`output-subsumption` *&key* (*file-name* *nil*))

**Arguments:** *:file-name* - file name, possibly with path, to which the subsumption relationships are written. If not given, the output is written on the console screen.

---

[output-imp-sets](#) *macro*

---

**Description:** Outputs all computed subsumption relationships in the form of implication sets.

**Syntax:** (`output-imp-sets`)

**Remarks:** This macro is DEPRECATED! Please use macro (`output-supers`) defined below instead.

---



---

[output-taxonomy](#) *macro*

---

- Description:** Outputs the computed Hasse diagram by identifying the parents, children and possibly also equivalents of each concept name.
- Syntax:** (output-taxonomy &key (*file-name* nil))
- Arguments:** *:file-name* - file name, possibly with path, to which the subsumption relationships are written. If not given, the output is written on the console screen.
- Remarks:** This feature is available after classification. The concept hierarchy is constructed automatically, if it is not present.

### 3.5.2 DIG

---

[startup-dig-server](#) *function*

---

- Description:** Starts up the DIG server for CEL
- Syntax:** (startup-dig-server &key (*port* 8080))
- Arguments:** *:port* - port number to which a DIG client can connect. If not given, CEL tries to startup the server at port 8080.
- Remarks:** See command line options (4) for an alternative way to startup the server directly.

---

[shutdown-dig-server](#) *function*

---

- Description:** Shuts down the DIG server for CEL
- Syntax:** (shutdown-dig-server)
- Remarks:** To safely exit the CEL system, especially when the server is up and active, use (quit) function!

### 3.5.3 Information and utilities

---

[help](#) *function*

---

- Description:** A simple interactive online help system that displays brief help messages
- Syntax:** (help &optional *name*)
- Arguments:** *name* - keyword, macro or function name for which the help message is to be displayed. If not given or unrecognized, it lists all available *names*.

---

[version](#) *function*

---

- Description:** Displays the version of CEL
- Syntax:** (version)

---

**build** *function*

---

**Description:** Displays the version and build of the current CEL system  
**Syntax:** (build)

---

**el+** *function*

---

**Description:** Describes the underpinning Description Logic  $\mathcal{EL}^+$   
**Syntax:** (el+ &key (*range-restriction* nil)  
                  (*role-inclusion* nil))  
**Arguments:** *:range-restriction* - if set to **t**, describes a tractable variant of  $\mathcal{EL}^+$   
                  with range restrictions  
*:role-inclusion* - if set to **t**, describes a tractable variant of  $\mathcal{EL}^+$   
                  with complex role inclusions  
**Remarks:** This function only displays information about the two variants of  
the underpinning logic. It does not change the internal state nor  
modify the input ontology. If the input ontology contains both com-  
plex role inclusions and range restrictions, CEL will automatically  
detect that and will not classify the ontology.

---

**start** *function*

---

**Description:** Loads, preprocesses, and classifies the ontology, all in one com-  
mand.  
**Syntax:** (start *tboxfile* &key (*uri* default))  
**Arguments:** *tboxfile* - path string pointing to the input ontology file.  
*:uri* - unique URI for the ontology. If not given, the default URI and  
thus default ontology is used. If set to **t**, the system will generate  
a unique URI for the new ontology.  
**Remarks:** For more information, see (create-ontology), (load-ontology)  
and (classify-ontology).

---

## 4 Command Line Options

This section describes the CEL command line and its options which are catered for batch classification operations and the DIG server. To begin with, consider the following command line:

```
$cel -l "med.tbox" -c -outputHierarchy "med.hierarchy" -q
```

It tells the CEL reasoner to load the ontology from file “med.tbox” and classify it. After successfully classifying the ontology, it outputs the concept hierarchy to file “med.hierarchy” and finally quits the system. If the last bit of the line (-q) is absent, then CEL prompts for interactive commands, so that further operations (see Section 3) could be carried out. Another example shows how to get the DIG server running:

`$cel -digServer`

There are a few other command line options which are specified in the table below.

<code>-loadOntology file &lt;-l&gt;</code>	Load and preprocess ontology from file
<code>-classifyOntology &lt;-c&gt;</code>	Classify the ontology
<code>-outputSupers [file]</code>	Output the sets of all super-classes
<code>-outputTaxonomy [file]</code>	Output the direct sub- and super-classes
<code>-outputHierarchy [file]</code>	Output the hierarchy as an indented tree
<code>-digServer [port]</code>	Start CEL as a backend DIG reasoner
<code>-quit &lt;-q&gt;</code>	Exit; Don't enter the interactive interface
<code>-help &lt;-h&gt;</code>	Display this help message

## 5 An Example: Learning by Doing

In this section, we demonstrate the use of CEL in the interactive mode step by step. This includes:

- creating an ontology,
- starting up the reasoner,
- loading the ontology,
- classifying the ontology,
- querying subsumptions, and
- output the classification.

Additionally, we also illustrate two new features – complex subsumption queries and incremental classification – in Subsection 5.1 and 5.2, respectively.

**Creating an ontology** CEL is a reasoning backend, and as such does not provide an ontology editor to create and maintain ontologies. Any ontology editor that produces KRSS syntax and supports (a subset of) the expressive means provided by  $\mathcal{EL}^+$  can be used together with CEL. To get started, it is perhaps advisable to generate some simple ontologies using a standard text editor such as emacs.

The following simple ontology is provided as the file `./tboxes/med.tbox` of the CEL distribution. The concept name HDSNT is an abbreviation for “heart disease needing treatment”.

```
(define-primitive-role cont-in :parent comp-of)

(implies Pericardium (and Tissue
                        (some cont-in Heart)))
(implies Pericarditis (and Inflammation
                            (some has-loc Pericardium)))
```

```

(implies Inflammation (and Disease
                        (some acts-on Tissue)))

(implies (and Disease
            (some has-loc
              (some comp-of Heart)))
         (and Heartdisease
            (some has-state NeedsTreatment)))

(define-concept HDSNT
  (and Heartdisease
    (some has-state NeedsTreatment)))

```

**Starting up the reasoner** CEL can be started by simply calling the executable in the `./bin` subdirectory of the distribution. After the reasoner has been successfully loaded, the greeting screen will appear and CEL will prompt for commands from the user.

**Loading the ontology** At the prompt, write down the following command:

```
(load-ontology "med.tbox")
```

CEL searches for the specified ontology not only in the current directory but also in the directory `./tboxes`. A full path to the ontology file can also be given.

**Classifying the ontology** After having read in and preprocessed the ontology, CEL is ready to classify it. The user may want to check this status by asking

```
(ontology-prepared?)
```

The answer will then be `t`, i.e., “yes”. At this point, the classification can be started by calling the command

```
(classify-ontology)
```

This may take some time depending on the size and complexity of the ontology. To load and classify an ontology in one step, the `start` function can be used.

**Querying subsumption** When the ontology has successfully been classified (to check this, call `(ontology-classified?)`), subsumption between two concept names w.r.t. the classified ontology can be queried. In the example ontology above, the user might want to know if `Pericarditis` is in fact a heart disease needing treatment. This can be checked by querying

```
(concept-subsumes? HDSNT Pericarditis).
```

The set of all subsumers and subsumees of a concept name can also be obtained by using `ancestors` and `descendants`, respectively.

**Output the classification results** After the classification, the concept hierarchy can be archived in a file or displayed on screen by using the functions `output-subsumption` and `output-imp-sets`, depending on the format and brevity needed. If the classification has been carried out in mode 1, it is additionally possible with `output-hasse` to output the most compact representation of a taxonomy, i.e., Hasse diagram. By using `output-hierarchy`, a visualization of the terminological hierarchy can be displayed. Moreover, (maximal) sets of pairwise subsuming concept names (called *synonyms*) can be displayed by the function `output-synonyms`. Please refer to the more detailed explanation in Section 3.5.

## 5.1 Complex subsumption queries

## 5.2 Incremental Classification

## References

- [1] F. Baader, S. Brandt, and C. Lutz. Pushing the  $\mathcal{EL}$  envelope. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.
- [2] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.
- [3] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. Is tractable reasoning in extensions of the description logic  $\mathcal{EL}$  useful in practice? In *Proceedings of the 2005 International Workshop on Methods for Modalities (M4M-05)*, 2005.
- [4] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. Is tractable reasoning in extensions of the description logic  $\mathcal{EL}$  useful in practice? In *Journal of Logic, Language and Information, Special Issue on Method for Modality (M4M)*, 2007. To appear.
- [5] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG description logic interface. In *Proceedings of the International Workshop on Description Logics (DL-2003), Rome, Italy, September 5-7, 2003*.
- [6] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
- [7] D. L. McGuinness and F. van Harmelen. OWL web ontology language overview. See <http://www.w3.org/TR/owl-features/>, 2004.
- [8] P. Patel-Schneider and B. Swartout. Description-logic knowledge representation system specification from the krss group of the arpa knowledge sharing effort. Technical

report, DARPA Knowledge Representation System Specification (KRSS) Group of the Knowledge Sharing Initiative, 1993.

## Index

- CEL axioms
  - concept-inclusion, 8
  - define-concept, 8
  - define-primitive-concept, 8
  - define-primitive-role, 10
  - disjoint, 8
  - domain, 11
  - equivalent, 8
  - implies, 8
  - instance, 12
  - range, 11
  - reflexive, 11
  - related, 12
  - role-inclusion, 10
  - transitive, 11
- CEL dig support
  - shutdown-dig-server, 24
  - startup-dig-server, 24
- CEL information & utilities
  - build, 24
  - el+, 25
  - help, 24
  - start, 25
  - version, 24
- CEL keywords
  - and, 7
  - bottom, 7
  - compose, 7
  - some, 7
  - top, 6
- CEL ontology operations
  - activate-ttbox, 14
  - classify-ontology, classify-tbox, 14
  - classify-ttbox, 15
  - clear-ontology, clear-tbox, 12
  - clear-ttbox, 14
  - commit-ttbox, 14
  - create-ontology, create-tbox, 12
  - deactivate-ttbox, 15
  - detail-ontology, detail-tbox, 14
  - load-ontology, load-tbox, 13
  - release-all-ontologies, release-all-tboxes, 13
  - release-ontology, release-tbox, 13
  - repository, 13
  - restore-ontology, restore-tbox, 13
  - start, 25
- CEL outputs
  - output-hierarchy, 23
  - output-subsumption, 22
  - output-supers, 22
  - output-synonyms, 23
  - output-taxonomy, 23
- CEL queries
  - ?implies, 18
  - ?subsumes, 17
  - abox-consistent?, 16
  - all-concepts, 17
  - all-individuals, 21
  - all-roles, 19
  - all-unsatisfiable-concepts, 17
  - ancestors, super-concepts, 19
  - child-roles, 20
  - children, child-concepts, 18
  - concept-equivalent?, equivalent?, 18
  - concept-implies?, implies?, 18
  - concept-instances, 22
  - concept-subsumes?, subsumes?, 17
  - concept?, 17
  - descendants, sub-concepts, 19
  - equivalents, 19
  - individual-direct-types, 21
  - individual-types, 21
  - individual?, 21
  - instance?, 21
  - ontology-classified?, tbox-classified?, 15
  - ontology-consistent?, 15
  - ontology-prepared?, tbox-prepared?, 15
  - parent-roles, 20
  - parents, parent-concepts, 18
  - reflexive?, 20

role-implies?, 20  
role-subsumes?, 20  
role?, 19  
satisfiable?, concept-satisfiable?, 17  
sub-roles, 21  
super-roles, 20  
tbox-consistent?, 16  
transitive?, 20  
tbox-active?, 16  
tbox-classified?, 16  
tbox-cleared?, 16  
tbox-prepared?, 16

Concept language, 4

Language

for concepts, 4

for ontology, 4

Ontology language, 4

System requirements, 4

Underlying logic, 3