An Algorithm for Resolution of Common Logic (Edition 2) Importation Implemented in OntoMaven

Tara ATHAN [a,1], Ralph SCHAEFERMEIER [b,2] and Adrian PASCHKE [b,3]
[a] *Athan Services, USA*
and [b] *Freie Universitaet Berlin, Germany*
.

Abstract. This paper describes the OntoMaven implementation of a resolution algorithm for importation in the ISO Common Logic (Edition 2) language. OntoMaven is a plug-in based extension of the Apache Maven software build tool for the development and reuse of knowledge base (KB) artifacts, such as ontologies and rule bases, managed in distributed (Web) repositories, such as Github and Subversion. The OntoMaven CL2 import plug-in contributes a proof-of-concept implementation of CL2 importation semantics that handles circular imports.

Keywords. Common Logic, importation, OntoMaven, circular importation, domain restriction

Introduction

The ISO Common Logic (CL) Standard [1] was published in 2007, with a syntax that includes syntactic categories called *named text*, *imports* and *module* which were intended to allow distributed publishing of CL texts. Some technical issues were discovered [2] in the CL Standard semantics, particularly in regard to modules, prompting commencement of a revision of the CL Standard [3]. A proposal has been developed [4] to revise the abstract syntax and semantics in order to resolve these issues, and at the time of publication of this paper, an Initial Submission [5] (CL2) has been presented that adopts this proposal. Relative to the CL Standard, CL2 takes a significantly different approach to distributed publishing of texts. The goal of this paper is to demonstrate an implementation of importation resolution of distributed CL2 texts.

In this paper, we will make use of an abbreviated functional notation for the structural aspects of the CL2 syntax, defined in terms of the normative CL2 functional syntax as follows:

- $\mathrm{ttl}(N, \Gamma) := \mathrm{title}(N, \Gamma)$ is a N titling with content Γ.
- $\mathrm{X}(E_1, \ldots, E_n) := \mathrm{txt}(E_1, \ldots, E_n)$ is a text construction with arguments E_i, which may be texts, statements or sentences.

[1]Corresponding Author: Tara Athan, Athan Services, athant.com, 334 Hollowood Dr, W Lafayette, IN USA; E-mail:taraathan@gmail.com.
[2]Corresponding Author: Ralph Schaefermeier; Freie Universitaet Berlin; E-mail:schaef@inf.fu-berlin.de
[3]Corresponding Author: Adrian Paschke; AG Corporate Semantic Web, Freie Universitaet Berlin.

- $\mathrm{Q}_N(\Gamma) := \mathrm{domain}(N, \Gamma)$ is a domain restriction with domain name N applied to a text Γ.
- $\mathrm{M}_N := \mathrm{import}(N)$ is an importation of the title N.

We also make use of composition notation and some shortcuts:
- $\mathrm{L}(N, \Gamma) := \mathrm{X}(\mathrm{ttl}(N, \Gamma))$ is a titling text.
- $\perp$ is a canonical unsatisfiable text, e.g. a text construction whose argument is the empty disjunction; Γ is unsatisfiable iff $\Gamma \models \perp$.
- $A \circ B(\Gamma) := A(B(\Gamma))$
- $\mathrm{Q}_{\langle N_1,\ldots,N_m\rangle} := \mathrm{Q}_{N_1} \circ \ldots \circ \mathrm{Q}_{N_m}$
- $\mathrm{Q}_S := \mathrm{Q}_{\langle N_1,\ldots,N_m\rangle}$, where $S = \{N_1,\ldots, N_m\}$ with $N_j < N_{j+1}$.

In the last item, WLOG we make use of some total ordering, e.g. lexicographic, on the countable vocabulary V.

As defined in the CL2 Standard, the entailment relation ($\models$) holds fundamentally between corpora - sets of texts; however, it is left unspecified how a corpus is to be delineated in practice. In this proof-of-concept implementation, we use the file system as our mechanism for corpus delineation - a corpus is the set of XCL2 documents in a directory.

Instead of named texts, CL2 has **titlings**, $\mathrm{ttl}(N, \Gamma)$. The content Γ is not considered to be asserted unless it is imported, and the semantics of the assertions *is* affected by any domain restrictions that are applied to that importation, so that in general $\mathrm{Q}_S(\mathrm{M}_N) \not\models \mathrm{Q}_{S'}(\mathrm{M}_N)$ for $S \neq S'$. Expressions that occur within some titling are called **inaccessible**; otherwise, they are **accessible**. Unlike OWL and RDF, Common Logic is not closely tied to the physical structure of the Web or its standards and conventions. In particular, in titling $\mathrm{ttl}(N, \Gamma)$ it is *not* assumed that N is an IRI. If N is an IRI, it is *not* assumed that Γ may be dereferenced from that IRI. Here, we assume that titles are IRIs; however, our methods could be easily extended to the general case.

CL2 importations are allowed to be circular; e.g., the content of a titling may import itself, as in $\mathrm{L}(N, \mathrm{M}_N)$, and the content of a pair of titlings may import each other, as in $\mathrm{X}(\mathrm{L}(N, \mathrm{M}_M), \mathrm{L}(M, \mathrm{M}_N))$. Because of this, the CL2 semantics employs a fixed point approach to define the semantics of importations (see the Appendix).

A common strategy for handling circular importations in other languages, e.g. in OWL, is to ignore repetitions. However, since CL2 importation within a domain restriction may change the semantics of the imported text, more complex criteria must be satisfied in order to avoid circularity.

Our resolution of CL2 importation iteratively applies a semantics-preserving transformation $\mathscr{R}$ to a corpus $\mathscr{C}$ to obtain a sequence $\mathscr{C}_n = \mathscr{R}^n(\mathscr{C})$, $n =0, 1, \ldots$, terminating when a fixed point ($\mathscr{C}_m = \mathscr{C}_{m+1}$) is reached or inconsistent titlings (different texts assigned to the same title, e.g. $X(\mathrm{L}(N, \Gamma), \mathrm{L}(N, \Gamma')))$ are found (in which case, $\mathscr{C} \models \perp$).

In the CL Standard, as well as its revision CL2, multiple dialects (serializations) are presented. In our implementation, we handle only the XML-based syntax XCL2, although this approach could be applied to CLIF2 or other CL2 dialects through translations based on their shared abstract syntax. The XML-based syntax of XCL2 allows us to take advantage of the many standards, and implementations thereof, available for manipulating XML. In particular, canonical XML [6] is employed to compare XCL2 texts for syntactic equivalence. Before determining the canonical XML form, syntactic elements that are external to the CL2 abstract syntax, such as key attributes for element labels, and XML comments, are removed. CURIEs are resolved according to their prefix declarations, and the optional `symbol` element is made explicit in all `Name` and `Data` elements.

1. Approach

The contribution of this paper is the proof-of-concept implementation of the OntoMaven [7] CL2 import plug-in with a resolution algorithm that handles circular importations.

OntoMaven [8–10] extends the Apache Maven software project management tool [11] and adapts it for knowledge base (KB) project management. OntoMaven dynamically downloads KB artifacts (ontologies, rule bases, semantic data resources, etc.) and plug-ins for ontology/rule engineering from distributed OntoMaven repositories and manages them in a local development cache for further reuse and development. OntoMaven repositories are extensions of Maven repositories (supporting e.g. Github, Subversion and folder-based document management systems) with ontology versioning and maintenance metadata for the build life cycle management of KB artifacts.

Like Maven, OntoMaven is also based around the central concept of a build lifecycle which is made up of build phases representing certain stages in the development life cycle. It therefore extends an OntoMaven declarative XML-based POM (Project Object Model) to describe a KB project being build, its dependencies on required external KB artifacts and components which are published in distributed repositories, the build order, and the required plug-ins which implement and execute the goals defined for each project build phase. A goal represents a specific task which contributes to the build and management of an OntoMaven KB project. OntoMaven is using Maven's extensible plug-in architecture for implementing predefined goals which are performing certain well-defined tasks in different build phases, such as import and dependency management of KB artifacts from distributed repositories, versioning with semantic difference computation, automated documentation and testing, and deployment etc. The focus of this section is on the CL2 import plug-in which implements an importation resolution algorithm.

1.1. Algorithm

The importation resolution algorithm iteratively transforms a given corpus of CL2 texts into a logically-equivalent corpus by resolving each accessible *N* importation provided there is a unique accessible *N* titling.

The import resolution algorithm is initialized with a set of XCL2 documents *Docs* which constitute the text corpus. The documents are contained in a data structure *CL2-Corpus*, which reflects the state of the corpus during the transformation process (see Listing 1).

The documents of the corpus may contain XInclude directives, either created by an earlier application of the importation resolution algorithm, inserted manually, or created by some other application. To assist in following these include directives, the corpus data structure also contains an XML Catalog *Catalog* and an *Includes* map. The *Catalog* is a map from the IRIs referenced by the XInclude directives to the local paths of any cached copies of the included texts. When the corpus is loaded, the locally-cached includes are loaded into memory in the *Includes* map, using their local path as the key, and uncached includes are downloaded from their remote locations, if possible, and added to the *Catalog* and *Includes* maps as well. Failure to obtain an included text causes an Exception to be thrown.

```
CL2-Corpus(Docs, Catalog, Includes)
Init(Docs)
$T \leftarrow$ Empty Map
for each document $d$ in Docs
    do Identify-Available-Titlings($d.root$, $T$)

Identify-Available-Titlings($e$, $T$)
if $e$ is Titling
  then if $T$ contains key $e$.name
         then if $e$.content = $T$.get($e$.name)
                then return
                else Conflicting Titling Error
         else $T$.add($e$.name, $e$.content)
       return
if $e$ is Include
  then
       $i \leftarrow$ Includes.get(Catalog.get($e$.href))
       Identify-Available-Titlings($i$, $T$)
       return
for each node $c$ in $e$.children
    do if $c$ is Restrict or Construct
         then Identify-Available-Titlings($c$, $T$)
```

Listing 1. The initialization of the CL text corpus structure

```
Process-Imports($T$)
while resolvable accessible importations exist
    do for each document $d$ in Docs
         do $R \leftarrow$ Empty Stack
            Process-Import($d.root$, $R$, $T$)
```

Listing 2. The Process-Imports function constitutes the entry point to the recursion.

Titlings initially accessible are collected into a set T by traversing the node tree of each document d, starting from the root element $d.root$, until the first `Titling` element in the subtree is encountered. Nodes other than the text elements Restrict and Construct also terminate their subtree recursion. Duplicate titlings are ignored, while inconsistent titlings lead to an error message and the premature termination of the importation process.

Listing 2 shows the entry into the recursion. Resolution of available imports is accomplished by traversing the structural elements (root and text elements) of each document in the corpus (XInclude directives encountered in this traversal are followed) and substituting each accessible importation occurrence with the content of a titling for the imported title, if such a titling is accessible anywhere in the corpus (see Listing 3). Substitution in this context means that the titling content is copied into an external document, and a reference to the external files is established by inserting an XML include (XInclude) directive in place of the CL2 importation.

In order to avoid duplicate importations, which could produce non-terminating loops, whenever an importation is resolved with an XInclude directive, its generated URI (described below) is added to $Catalog$ prior to entry into a recursion over the subtree of the imported text. If the generated URI for the current `Import` element is found in $Catalog$,

```
PROCESS-IMPORT$(e, R, T)$ $\triangleright$ $e$: an element, $R$: the restriction context stack, $T$: the titling context stack.
if $e$ is Import
  then
      $includeURI \leftarrow$ GENERATE-INCLUDE-URI($e$.name, $R$)
      if $Catalog$ contains $includeURI$
        then Duplicate import detected, replace $e$ with empty Construct
        else
            if $T$ contains key $e$.name
              then
                  $i \leftarrow$ new xml include element
                  $i$.href $\leftarrow includeURI$
                  replace $e$ with $i$
                  $newtext \leftarrow$ clone of $T$.get($e$.name)
                  $localURI \leftarrow$ a local path for caching $newtext$
                  $Catalog$.add($includeURI$, $localURI$)
                  $Includes$.add($localURI$, $newtext$)
                  IDENTIFY-AVAILABLE-TITLINGS$(newtext, T)$
                  PROCESS-IMPORT$(newtext, R, T)$
      return
if $e$ is Include
  then
      $i \leftarrow Includes$.get($Catalog$.get($e$.href))
      PROCESS-IMPORT$(i, R, T)$
      return
if $e$ is Restrict
  then $R$.push($e$.name)
if $e$ is Restrict or Construct or Root
  then
      for each node $c$ in $e$.children
          do PROCESS-IMPORT$(c, R, T)$
if $e$ is Restrict
  then $R$.pop()
```

Listing 3. The function PROCESS-IMPORT Handles a single CL element.

a duplicate importation has been detected, and an empty text construction is inserted in place of the `Import` element.

If a `Titling` element is encountered, there is no need to continue the recursion in the current branch, since all importations encountered in the content would be titled.

Each `Import` element, after checking for duplicates, is replaced with an `include` element, while the content of the imported titling is added to an *Includes* structure, which is in turn part of the *CL2-Corpus* structure and stores all imported titled texts.

`Include` elements may refer to texts that contain importations or nested titlings which have become accessible due to their importation. Therefore, the text referenced by the include is retrieved from the *Includes* structure and the recursion continues in the content of the included text, after newly available titlings are added to T.

The XML Catalogs standard [12] is used to later allow to resolve the identifier of the included resource to their physical location on disk.

When an `Include` element is encountered directly in subtree recursion, the included text is retrieved from the *Includes* structure and the recursion continues in the included text.

The history of domain restrictions is traced using a stack R. For each entry into a restricted context (within a Restricts element) the domain of the restriction is pushed onto

```
GENERATE-INCLUDE-URI(n, R)

1  $uri \leftarrow$ "http://ontomaven.org?uri=" + URLENCODE(n)
2  $R_{normalized} \leftarrow R$ sorted alphabetically, without duplicates
3  for $i \leftarrow 1$ to $R$.length
4      do $uri \leftarrow uri$ + ";dom" + $i$ + "=" + URLENCODE($R_{normalized}$[i])
5  return $uri$
```

Listing 4. The generation of an include URI.

the stack and deleted upon leaving the restricted context.

The scheme for constructing the system identifiers for the included resources is implemented in the procedure *generate-include-uri*, which is called from line 4 in the *Process-Import* procedure (see Listing 4).

As pointed out in the Appendix, the order of nestings of domain restrictions has no effect on the semantics of the resulting text, and it is sufficient to ensure that every arbitrary sequence of domain restrictions under which an importation is processed is recognized as the same domain restriction, as long as the members of the sequence are the same, regardless of their order or duplications. To achieve this, it is sufficient to sort the domains of the nested domain restrictions in alphabetical order and purge duplicates (line 2).

The system identifier for a resource that contains a text resulting from resolution of an importation of titled text with the title `http://example.org/myText-A1#a` (without domain restrictions) would be `http://ontomaven.org?uri=http\%3A\%2F\%2Fexample.org\%2FmyText-A1\%23a`.

The system identifier for the same text imported both under the nested domain restrictions `http://example.org/myDomain-N1` and `http://example.org/myDomain-N2`, as well as under the nested domain restrictions `http://example.org/myDomain-N2`, `http://example.org/myDomain-N1` would be `http://ontomaven.org?uri=http\%3A\%2F\%2Fexample.org\%2FmyText-N1\%23a;dom1=http\%3A\%2F\%2Fexample.org\%2FmyDomain-N1;dom2=http\%3A\%2F\%2Fexample.org\%2FmyDomain-N2`.

Determination of equivalent titlings is accomplished on the syntactical level by following these steps:

- resolution of all CURIES and removal of prefix declarations
- stripping out of XML comments and processing instructions
- removal of @key attributes
- making the symbol edge explicit in non-IRI Names and `Data` elements.
- transformation to canonical XML form

1.2. Integration into OntoMaven

The implementation of the CL2 importation algorithm described in this paper has been incorporated in a plug-in to the Maven build management system and complements the OntoMaven suite, which provides support for lifecycle management of OWL ontolgies and related artifacts in the form of importation, semantic versioning [13], automated entailment unit testing, consistency checking, aspect-oriented modularization [14], and documentation generation, in conjunction with automated dependency resolution and build support provided by Maven.

Such tasks are referred to as *goals* in Maven terminology, and the CL2 importation task has been implemented in the form of such a Maven goal. The invocation and parametrization of Maven goals, as well as the rest of the build cycle, is controlled in a declarative way, using XML-based POM (Project Object Model) descriptors.

2. Related Work

In software engineering circular dependencies between imports of software artifacts are widely considered as problematic. Several solutions on how to find and avoid circular dependencies have been proposed, including [15–18]. Tools with built-in support dependencies are e.g., Lattix [19] and JDepend [20].

The halting problem of cyclic imports is also closely related to infinite loops in recursion, as it occurs e.g. in several backward-reasoning resolution algorithms such as SLDNF. Much work has been done in this field to define restriction properties (on the dependency graph whose vertices are the predicate symbols from a logic program) for which SLDNF then is complete, such as e.g. stratification. Other approaches use alternating fixpoint semantics to capture the negation of positive existential closure (e.g. for transitive closure), such as the constructive characterization of the well-founded semantics for normal logic programs in terms of alternating fixpoint partial models [21], as well as negation of positive universal closure with alternating fixpoint semantics for general logic programs (negation in the conclusion) [22].

The general problem of cycle detection also occurs in graphs and several algorithms have been proposed such as Floyd's cycle-finding algorithm or Brents's algorithm.

From the perspective of modular ontologies or knowledge bases, it has been recognised since at least the 90's [23] that circular dependencies may be considred a desirable feature rather than a problem to be avoided. Circular importations are acceptable in OWL [24]. CL2 titlings have similarities to named graphs in RDF 1.1 [25], in that RDF names (IRIs or blank nodes) can perform a dual role of denoting entities in the universe of reference and independently naming graphs. However, the CL2 approach to titling differs from that of RDF 1.1 in that RDF Datasets, expressions which create the associations between names and graphs, are not given a model-theoretic semantics, while the CL2 titlings are integrated into the model-theoretic semantics.

The Common Logic community has chosen to embrace circular importations and semantics-altering importations into domain restrictions, together with the difficulties in implementation that arise from their interaction. The implementation presented here differs from previous approaches to circular dependencies due to the special requirements imposed on the resolution by importation within domain restriction, as well as the properties provide by the CL2 semantics that permit termination of the importation cycles.

3. Conclusion

The presented research follows a software engineering-oriented research methodology grounded in Design Science research. It contributes with a new practical design artifact, the OntoMaven CL2 plug-in, being a proof-of-concept implementation for the rigorous ISO Common Logic (Edition 2) importation resolution that handles circular importation. Future work will include

- integration of importation resolution with a mechanism to assemble a corpus from the IRIs of texts published on the internet (e.g. an XML configuration file, or a DOL [26] expression);

- extension to handle non-IRI titles;
- extension to other CL2 dialects;
- complexity analysis and optimization of performance (e.g. by tracking of the paths and restriction history of unresolved importations for use during the recursive resolution);
- profiling for applications that use CL2 importations;
- implementation of the construction of the closure core and canonical corpus, as defined in the Appendix.

The first point is already partly addressed by Maven's dependency resolution mechanism but is only applicable under the precondition that the external texts are published as Maven artefacts in Maven repositories. Future work will be focused on a generalization of this approach in order to cover cases where the external texts are published as general web resources, as well as generating transformed corpora in preparation for inference.

A. Appendix

The correctness of the importation resolution algorithm presented above depends on the corpus $\mathscr{C}_n$ at the n-th step of the importation resolution process being logically equivalent to the original corpus $\mathscr{C}_0$ for all n. This may be proved from two theorems: Th. 9 shows that substituting the content of an accessible N titling in place of an occurrence of an accessible N importation is a semantics-preserving transformation. Th. 11 shows that each step of the importation resolution algorithm is semantics-preserving. The termination of the importation resolution algorithm, proved in Th. 10, depends on the achievement of a fixed point after a finite number of steps.[4]

From [5], we obtain the following definitions. An **importation fixed-point** under a title mapping *ttl* (from names to texts) is a corpus $\mathscr{C}$ such that if Γ is a text in $\mathscr{C}$ then $\mathscr{C}$ contains every text Γ' derived from Γ by substituting $ttl(N)$ in place of one occurrence of an accessible N importation. The **importation closure** of a corpus $\mathscr{C}$ under a title mapping *ttl* is the intersection of all importation fixed-points under *ttl* that contain $\mathscr{C}$.

Let $\mathscr{T}$ be the set of texts in some CL2 language $\mathscr{L}$ with vocabulary V. Following [27], we adopt a simpler semantics than [5], which we call the structural semantics of $\mathscr{L}$. We define a **structural pre-interpretation** I of $\mathscr{L}$ to be a mathematical structure that has a title mapping ttl_I from names in V to texts in $\mathscr{T}$, and assigns a Boolean truth value $TV_I(\Gamma)$ to every text $\Gamma \in \mathscr{T}$. A corpus $\mathscr{C} \subseteq \mathscr{T}$ is **satisfied** by a structural pre-interpretation I (denoted $I \overset{st}{\models} \mathscr{C}$) if and only if $TV_I(\Gamma) = \text{true}$ for every text Γ in $Clo^I(\mathscr{C})$, the importation closure of $\mathscr{C}$ under the title mapping ttl_I.

Definition 1. *Let a* ***structural interpretation*** *I be a structural pre-interpretation that satisfies the following constraints: for any names N, $N_i \in V$, S any finite set of names in V, texts Γ, $\Gamma_i \in \mathscr{T}$, and sentences, statements or texts $E_i \in \mathscr{L}$,*
(a) $TV_I(\mathrm{M}_N) = \text{true}$.
(b) $TV_I(\mathrm{L}(N, \Gamma)) = \text{true}$ *iff* $ttl_I(N) = \Gamma$.
(c) If Γ *is an importation or titling, then* $TV_I(\Gamma) = TV_I(\mathrm{Q}_S(\Gamma))$.
(d) $TV_I(\mathrm{X}()) = TV_I(\mathrm{Q}_S(\mathrm{X}())) = \text{true}$.
(e) $TV_I(\mathrm{Q}_S(\mathrm{X}(\Gamma))) = TV_I(\mathrm{Q}_S(\Gamma))$.
(f) $TV_I(\mathrm{X}(E_1, \ldots, E_n)) = \wedge_{i=1}^{n} TV_I(\mathrm{X}(E_i))$, $n \geq 1$.

[4]Due to space limitations, we provide proofs for only some of the Theorems stated here. A journal paper with complete proofs is in preparation.

(g) $TV_I(\mathrm{Q}_S(\mathrm{X}(E_1, \ldots, E_n))) = TV_I(\mathrm{X}(\mathrm{Q}_S(\mathrm{X}(E_1)), \ldots, \mathrm{Q}_S(\mathrm{X}(E_n))))$, $n \geq 1$.
(h) $TV_I(\mathrm{Q}_S \circ \mathrm{Q}_{N_1} \circ \mathrm{Q}_{N_2}(\Gamma)) = TV_I(\mathrm{Q}_S \circ \mathrm{Q}_{N_2} \circ \mathrm{Q}_{N_1}(\Gamma))$.

We say that $\mathscr{C}_0 \subseteq \mathscr{T}$ **structurally entails** $\mathscr{C}_1 \subseteq \mathscr{T}$ ($\mathscr{C}_0 \overset{st}{\models} \mathscr{C}_1$) if every structural interpretation that satisfies $\mathscr{C}_0$ also satisfies $\mathscr{C}_1$. Structural entailment may be used to define a **structural-equivalence** relation: $\mathscr{C}_0 \dashv\overset{st}{\models} \mathscr{C}_1$ iff $\mathscr{C}_0 \overset{st}{\models} \mathscr{C}_1$ and $\mathscr{C}_1 \overset{st}{\models} \mathscr{C}_0$. A corpus transformation $T(\mathscr{C})$ **preserves structural semantics** iff $T(\mathscr{C}) \dashv\overset{st}{\models} \mathscr{C}$.

Theorem 1. *For every pair of corpora $\mathscr{C}_0$, $\mathscr{C}_1 \subseteq \mathscr{T}$, $\mathscr{C}_0 \overset{st}{\models} \mathscr{C}_1$ iff $\mathscr{C}_0 \overset{st}{\models} \{\Gamma\}$ for every $\Gamma \in \mathscr{C}_1$.*

The relevance of this structural semantics to the CL2 semantics is embodied in the following results.

Theorem 2. *For every CL2 interpretation I there is a structural interpretation $Struc(I)$ that has an equivalent satisfaction relation. That is, for any $\mathscr{C} \subseteq \mathscr{T}$, $I \mid\!\models \mathscr{C}$ iff $Struc(I) \mid\!\overset{st}{\models} \mathscr{C}$.*

Proof. Given I, we may define a structural interpretation $J = Struc(I)$ having the same title mapping, where for any text $\Gamma \in \mathscr{T}$, $TV_J(\Gamma) =$ true iff $I(\Gamma) =$ true and $ArgC(\Gamma) \in UD \cup UD^*$, as defined in [5]. It follows from the CL2 semantics that J satisfies the structural interpretation constraints. $\square$

Corollary 1. *If $\mathscr{C}_1 \overset{st}{\models} \mathscr{C}_2$, then $\mathscr{C}_1 \models \mathscr{C}_2$.*

Corollary 2. *If $\mathscr{C}_1 \dashv\overset{st}{\models} \mathscr{C}_2$, then $\mathscr{C}_1 \dashv\models \mathscr{C}_2$.*

In order to demonstrate that the transformations of the importation resolution algorithm preserve (structural) semantics, we will make use of some auxiliary definitions. We define the following subsets of $\mathscr{T}$.

- $\mathscr{T}_{ttl} := \{\Gamma \in \mathscr{T} : \Gamma = \mathrm{L}(N, \Delta)$ for some $N \in V$, $\Delta \in \mathscr{T}\}$.
- $\mathscr{T}_{prop0} := \{\Gamma \in \mathscr{T} : \Gamma = \mathrm{X}(E)\}$ for some sentence or discourse statement $E\}$.
- $\mathscr{T}_{prop} := \{\ \Gamma \in \mathscr{T} : \Gamma = \mathrm{Q}_S(\Gamma')$ for some $\Gamma' \in \mathscr{T}_{prop0}$ and some finite $S \subseteq V\}$.
- $\mathscr{T}_{imp0} := \{\Gamma \in \mathscr{T} : \Gamma = \mathrm{M}_N$ for some $N \in V\ \}$.
- $\mathscr{T}_{imp} := \{\ \Gamma \in \mathscr{T} : \Gamma = \mathrm{Q}_S(\Gamma')$ for some $\Gamma' \in \mathscr{T}_{imp0}$ and some finite $S \subseteq V\}$.

We define the set of **elementary texts** $\mathscr{T}_{elem}$ of $\mathscr{L}$ to be the union of $\mathscr{T}_{ttl}$, $\mathscr{T}_{prop}$ and $\mathscr{T}_{imp}$. A **formal** corpus is a corpus of elementary texts.

Theorem 3. *For every text $\Gamma \in \mathscr{T}$, there is a unique finite formal corpus $\mathscr{F}(\Gamma) \subset \mathscr{T}_{elem}$ that may be derived from Γ by a finite sequence of the following transformations, which preserve structural semantics:*

- *replacement of a corpus member that is a text construction with multiple arguments by the set of unary text constructions having one of these arguments;*
- *replacement of a corpus member that is a text construction with only text arguments by the set (possibly empty) of arguments;*
- *replacement of a corpus member that is a composition of domain restrictions of a text by a permuted composition of the same set of domain restrictions applied to the same text.*

- *replacement of a corpus member that is a composition of domain restrictions applied to a text construction by the set (possibly empty) of texts consisting of this composition of domain restrictions applied to a text construction of one of the arguments;*
- *replacement of a corpus member that is a composition of domain restrictions applied to a unary text construction of a text by the same composition of domain restrictions applied to the text.*
- *replacement of a corpus member that is a composition of domain restriction of a titling text by the titling text.*

Proof. Intuitively, Γ corresponds to a finite abstract structural syntax tree, where each non-leaf subtree is a text, while leaves may be sentences, statements, importations or empty text constructions. Each path from the root to a leaf that is not an empty text construction corresponds to a unique elementary text, which together constitute $\mathscr{F}(\Gamma)$. Each transformation above preserves structural semantics, due to the structural interpretation constraints. A finite number of such transformations can be performed for a given finite text, terminating in $\mathscr{F}(\Gamma)$. Thus $\mathscr{F}(\Gamma) \overset{st}{=\!|\!\models} \{\Gamma\}$. $\square$

Theorem 4. *For each $\mathscr{C} \subseteq \mathscr{T}$ we define $\mathscr{F}(\mathscr{C}) := \cup_{\Gamma \in \mathscr{C}} \mathscr{F}(\Gamma)$. Then $\mathscr{F}(\mathscr{C}) \subseteq \mathscr{T}_{elem}$, $\mathscr{F}(\mathscr{F}(\mathscr{C})) = \mathscr{F}(\mathscr{C})$, $\mathscr{F}(\mathscr{C}_0 \cup \mathscr{C}_1) = \mathscr{F}(\mathscr{C}_0) \cup \mathscr{F}(\mathscr{C}_1)$, and $\mathscr{C} \overset{st}{=\!|\!\models} \mathscr{F}(\mathscr{C})$.*

Let the **closure core** $\mathscr{K}(\mathscr{C})$ of a structurally-satisfiable corpus $\mathscr{C} \overset{st}{\not\models} \bot$ be the intersection over all structurally-satisfying interpretations of the formal corpora of its importation closures: $\mathscr{K}(\mathscr{C}) := \cap_{I \overset{st}{\models} \mathscr{C}} \mathscr{F}(Clo^I(\mathscr{C}))$. If $\mathscr{C} \overset{st}{\models} \bot$, we define $\mathscr{K}(\mathscr{C}) := \mathscr{T}_{elem}$.

Theorem 5. *For every corpus $\mathscr{C}$, $\mathscr{F}(\mathscr{C}) \subseteq \mathscr{K}(\mathscr{C})$, $\mathscr{K}(\mathscr{F}(\mathscr{C})) = \mathscr{K}(\mathscr{C})$, and $\mathscr{K}(\mathscr{C}_0) \cup \mathscr{K}(\mathscr{C}_1) \subseteq \mathscr{K}(\mathscr{C}_0 \cup \mathscr{C}_1)$. Further, $\mathscr{K}(\mathscr{C}) \overset{st}{=\!|\!\models} \mathscr{C}$, and hence $\mathscr{K}(\mathscr{C}) =\!|\!\models \mathscr{C}$.*

A **canonical** corpus is a corpus $\mathscr{C} \subseteq \mathscr{T}_{elem}$ such that there is no name N such that $\mathscr{C}$ contains both an accessible importation of N and an accessible titling of N. For every corpus $\mathscr{C} \subset \mathscr{T}$, define $Can(\mathscr{C})$ to be the canonical corpus obtained by deleting from its closure core $\mathscr{K}(\mathscr{C})$ any accessible N importation if there is an accessible N titling in $\mathscr{K}(\mathscr{C})$.

Theorem 6. *The closure core and canonical corpus of a finite, structurally-satisfiable corpus $\mathscr{C} \subseteq \mathscr{T}$, $\mathscr{C} \overset{st}{\not\models} \bot$ may be derived by the following algorithm, which preserves structural semantics at each step. Starting with $\mathscr{C}_0 = \mathscr{F}(\mathscr{C})$, $\mathscr{C}_0^{res} = \emptyset$,*

- *Partition $\mathscr{C}_n$ into disjoint sets $\mathscr{C}_n^{ttl}$, $\mathscr{C}_n^{prop}$, $\mathscr{C}_n^{imp}$, $\mathscr{C}_n^{res}$, where $\mathscr{C}_n^{ttl} = \mathscr{C}_n \cap \mathscr{T}_{ttl}$, $\mathscr{C}_n^{prop} = \mathscr{C}_n \cap \mathscr{T}_{prop}$, $\mathscr{C}_n^{imp} = \mathscr{C}_n \cap \mathscr{T}_{imp} - \mathscr{C}_n^{res}$.*
- *Select a text Γ_n from $\mathscr{C}_n^{imp}$, containing an N importation, such that $\mathscr{C}_n^{ttl}$ contains an N titling. If there is no such text, stop. Substitute the content of the N titling in place of the N importation occurrence to obtain Γ'_n.*
- *Construct $\mathscr{C}_{n+1} = \mathscr{C}_n \cup \mathscr{F}(\Gamma'_n)$, $\mathscr{C}_{n+1}^{res} = \mathscr{C}_n^{res} \cup \{\Gamma_n\}$.*

Repeat until termination at some step $m < \infty$. Then $\mathscr{K}(\mathscr{C}) = \mathscr{C}_m$, and $Can(\mathscr{C}) = \mathscr{C}_m - \mathscr{C}_m^{res}$.

Corollary 3. *If $\mathscr{C} \subseteq \mathscr{T}$, $\mathscr{C} \overset{st}{\not\models} \bot$ is finite, then so is $\mathscr{K}(\mathscr{C})$.*

Corollary 4. *If $\mathscr{C} \subseteq \mathscr{T}$, $\mathscr{C} \overset{st}{\not\models} \bot$ is finite, then so is $Can(\mathscr{C})$.*

Corollary 5. *If $\mathscr{C} \subseteq \mathscr{T}$, $\mathscr{C} \overset{st}{\models} \bot$, then $Can(\mathscr{C}) = \mathscr{T}_{ttl} \cup \mathscr{T}_{prop}$.*

Theorem 7. *For every corpus $\mathscr{C} \subseteq \mathscr{T}$, $Can(\mathscr{C}) \dashv\overset{st}{\models} \mathscr{C}$, and hence $Can(\mathscr{C}) \dashv\models \mathscr{C}$.*

Theorem 8. *For any pair of corpora $\mathscr{C}_0$, $\mathscr{C}_1 \subseteq \mathscr{T}$, $\mathscr{C}_0 \overset{st}{\models} \mathscr{C}_1$ iff $Can(\mathscr{C}_0) \supseteq Can(\mathscr{C}_1 \cup (Can(\mathscr{C}_0) \cap \mathscr{T}_{ttl}))$. Further, $\mathscr{C}_0 \dashv\overset{st}{\models} \mathscr{C}_1$ iff $Can(\mathscr{C}_0) = Can(\mathscr{C}_1)$.*

Theorem 9. *Let $\mathscr{C}_0$ be a corpus containing an accessible N titling with content Γ. Let $\mathscr{C}_1$ be the corpus resulting from $\mathscr{C}_0$ by substituting Γ in place of an occurrence of an accessible N importation. Then $\mathscr{C}_0 \dashv\overset{st}{\models} \mathscr{C}_1$, and hence $\mathscr{C}_0 \dashv\models \mathscr{C}_1$.*

Proof. By construction, we have $Can(\mathscr{C}_0) = Can(\mathscr{C}_1)$ and so from Th. 8 we have $\mathscr{C}_0 \dashv\overset{st}{\models} \mathscr{C}_1$. By Th. 2, it follows that $\mathscr{C}_0 \dashv\models \mathscr{C}_1$. □

Let $\mathscr{C} \subseteq \mathscr{T}$ be finite, $\mathscr{C} \overset{st}{\not\models} \bot$, and let $\mathscr{C}_n = \mathscr{R}^n(\mathscr{C})$, n =0, 1, ... be the result of the importation resolution procedure (ignoring the termination criterion).

Theorem 10. *There exists a non-negative integer m such that $\mathscr{C}_m = \mathscr{C}_{m+1}$.*

Proof. There are a finite number of titlings, importations and domain restrictions (accessible or not) in the original corpus. Hence there will be a finite number of importations that are resolved by substituting titling content. Once these substitutions have been exhausted, the only further changes in the corpus will be substituting empty text constructions, which reduce the number of importations in the corpus, and so a fixed point is reached after some finite number of steps. □

Theorem 11. *$\mathscr{C}_n$ is structurally, and hence logically, equivalent to $\mathscr{C}_0$.*

Proof. Consider the sequence $\hat{\mathscr{C}_n}$ where $\hat{\mathscr{C}_n} = \mathscr{C}_n \cup \mathscr{I}_n$, $\mathscr{I}_0$ is the empty set and $\mathscr{I}_n = \mathscr{I}_{n-1} \cup \{\Gamma_n\}$ where Γ_n is the elementary text that is an N importation within the same set of domain restrictions as the importation occurrence that is substituted for at step n. Then it may be shown by induction that there is a finite subsequence n_j, j =0, ..., k such that $\mathscr{F}(\hat{\mathscr{C}}_{n_j})$ is the same as some sequence obtained by the procedure defined in Th. 6. Further, for any integer i such that $n_j \leq i < n_{j+1}$ $\mathscr{F}(\hat{\mathscr{C}_i}) = \mathscr{F}(\hat{\mathscr{C}}_{n_j})$ and for any integer $i \geq n_k$, $\mathscr{F}(\hat{\mathscr{C}_i}) = \mathscr{F}(\hat{\mathscr{C}}_{n_k}) = \mathscr{K}(\mathscr{C})$. □

Corollary 6. *If $\mathscr{C}_m = \mathscr{C}_{m+1}$, then $\mathscr{F}(\mathscr{C}_m) = Can(\mathscr{C})$.*

References

[1] ISO/IEC 24707-2007: Information technology – Common Logic (CL): a framework for a family of logic-based languages. ISO, Geneva, Switzerland (2007)

[2] Neuhaus, F., Hayes, P.: Common Logic and the Horatio problem. Applied Ontology **7**(2) (2012) 211–231

[3] Ontolog Wiki: CommonLogic V2. `http://ontolog.cim3.net/cgi-bin/wiki.pl?CommonLogic_V2` Accessed: 2014-07-20.

[4] Athan, T., Neuhaus, F.: Towards a Revised Semantics for Common Logic. (in preparation)

[5] ISO CD 24707: Common Logic (CL): a framework for a family of logic-based languages. 2. edn. ISO, Geneva, Switzerland (under revision)

[6] W3C: Canonical XML Version 2.0. `http://www.w3.org/TR/xml-c14n2/` Accessed: 2014-07-20.

[7] OntoMaven: GitHub repository. `https://github.com/ag-csw/OntoMaven/tree/master/OntoMaven-CL` Accessed: 2014-07-20.

[8] Paschke, A.: OntoMaven API4KB - A Maven-based API for Knowledge Bases. In: 6th International Semantic Web Applications and Tools for the Life Science (SWAT4LS 2013), Edinburgh, UK, December 10-12, 2013

[9] Paschke, A.: OntoMaven: Maven-based Ontology Development and Management of Distributed Ontology Repositories. CoRR **abs/1309.7341** (2013)

[10] Paschke, A.: OntoMaven. In: 9th International Workshop on Semantic Web Enabled Software Engineering (SWESE 2013), Berlin, Germany, December 2-5, 2013. (2013)

[11] Apache Maven: Project Web Site. `http://maven.apache.org/` Accessed: 2014-07-20.

[12] OASIS: Canonical XML Version 2.0. `https://www.oasis-open.org/committees/download.php/14809/xml-catalogs.html` Accessed: 2014-07-20.

[13] Luczak-Rösch, M., Coskun, G., Paschke, A., Rothe, M., Tolksdorf, R.: SVoNt - Version Control of OWL Ontologies on the Concept Level. In Fähnrich, K.P., Franczyk, B., eds.: Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik (GI). Volume 176 of LNI., GI (2010) 79–84

[14] Schäfermeier, R., Paschke, A.: Aspect-Oriented Ontologies: Dynamic Modularization Using Ontological Metamodeling. In: Proceedings of the Seventh International Conference (FOIS 2014), IOS Press (2014)

[15] Laval, Jannik and Denier, Simon and Ducasse, Stephane and Bergel, Alexandre: Identifying Cycle Causes with Enriched Dependency Structural Matrix. In: Proceedings of the 2009 16th Working Conference on Reverse Engineering. WCRE '09, Washington, DC, USA, IEEE Computer Society (2009) 113–122

[16] Sangal, N., Jordan, E., Sinha, V., Jackson, D.: Using Dependency Models to Manage Complex Software Architecture. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. OOPSLA '05, New York, NY, USA, ACM (2005) 167–176

[17] Melton, H., Tempero, E.: JooJ: Real-time Support for Avoiding Cyclic Dependencies. In: Proceedings of the Thirtieth Australasian Conference on Computer Science - Volume 62. ACSC '07, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2007) 87–95

[18] Vainsencher, D.: MudPie: Layers in the Ball of Mud. Comput. Lang. Syst. Struct. **30**(1-2) (April 2004) 5–19

[19] Lattix: Product Home Page. `http://lattix.com/` Accessed: 2014-07-20.

[20] JDepend: Project Home Page. `http://clarkware.com/software/JDepend.html` Accessed: 2014-07-20.

[21] Van Gelder, A.: The Alternating Fixpoint of Logic Programs with Negation. In: Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. PODS '89, New York, NY, USA, ACM (1989) 1–10

[22] Van Gelder, A., Ross, K.A., Schlipf, J.S.: The Well-founded Semantics for General Logic Programs. J. ACM **38**(3) (July 1991) 619–649

[23] Farquhar, A., Fikes, R., Rice, J.: Tools for Assembling Modular Ontologies in Ontolingua, Citeseer

[24] OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, 2nd edition. `http://www.w3.org/TR/owl2-syntax/#Imports` Accessed: 2014-07-20.

[25] RDF 1.1: Concepts and Abstract Syntax. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#section-dataset` Accessed: 2014-07-20.

[26] Lange, C., Kutz, O., Mossakowski, T., Grüninger, M.: The Distributed Ontology Language (DOL): Ontology Integration and Interoperability Applied to Mathematical Formalization. In Jeuring, J., Campbell, J., Carette, J., Reis, G., Sojka, P., Wenzel, M., Sorge, V., eds.: Intelligent Computer Mathematics. Volume 7362 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 463–467

[27] Athan, T.: Importation Closure that is Robust to Circular Dependencies. In Fodor, P., Roman, D., Anicic, D., Wyner, A., Palmirani, M., Sottara, D., Lévy, F., eds.: RuleML (2). Volume 1004 of CEUR Workshop Proceedings., CEUR-WS.org (2013)