Distributed Reasoning with Modular Ontologies

November 3, 2007 17:41

Jie Bao [a], Giora Slutzki [a], George Voutsadakis [b] and Vasant Honavar [a],
[a] *Department of Computer Science, Iowa State University, Ames, IA, USA 50010.*
E-mail: {baojie,slutzki,honavar}@cs.iastate.edu
[b] *Lake Superior State University, Sault Ste. Marie, MI 49783, USA.*
E-mail: gvoutsad@lssu.edu

Many real world applications of ontologies call for reasoning with modular ontologies. We describe a tableau-based reasoning algorithm based on Package-based Description Logics (P-DL), an ontology language that extends description logics with language features to support modularity. Unlike classical approaches that assume a single centralized, consistent ontology, the proposed algorithm adopts a federated approach to reasoning with modular ontologies wherein each ontology module has associated with it, a local reasoner. The local reasoners communicate with each other, as needed, in an asynchronous fashion. Hence, the proposed approach offers an attractive alternative to reasoning with multiple, autonomously developed ontology modules, in settings where it is neither possible nor desirable to integrate all involved modules into a single centralized ontology.

Keywords: Modular Ontology, Distributed Reasoning, Tableau Algorithm

1. Introduction

The success of the world wide web is, in large part, due to the *network effect* which leverages the participation of independent contributors who publish the web pages that constitute the web. Unlike the current web, which consists largely of web pages intended for human consumption, the semantic web aims at making the information sources machine interpretable and resources and services interoperable by annotating them using terms and relationships defined in controlled vocabularies or ontologies. Such ontologies typically represent conceptualizations of entities and properties developed by individuals or communities for use in a specific context. Consequently, such ontologies are autonomous, decentralized, and offer necessarily incomplete, partially overlapping coverage of specific domains (e.g., biology, medicine, pharmacology).

Effective use of web ontologies in practice requires support for inference across a loosely coupled federation of multiple, distributed, autonomous ontology modules, without having to combine the ontologies in one location. Current web ontology languages such as OWL [18] and the associated reasoners (e.g., FaCT++ [26] and Pellet [22]) provide, at best, very limited capabilities in such a setting. For example, an OWL ontology can "reuse" knowledge from another OWL ontology via the `owl:imports` construct. When one ontology imports another, the result is a union of the two ontologies with a single domain of interpretation. Inference in such a setting requires an *integration* of the relevant ontologies.

Because an OWL ontology can indirectly import knowledge from other OWL ontologies through arbitrarily deep importing chains, which collectively constitute its *importing transitive closure*, querying a small ontology might involve inference over a significant portion of the semantic web. This presents *scalability* challenges in terms of memory, time, and bandwidth requirements. Ontologies with more than a few tens of thousands of concepts are often beyond the capabilities of current reasoners [10].

The situation is further complicated in applications where *no global knowledge* of all ontology modules is available. For example, in a peer-to-peer setting, that is not at all atypical of semantic web applications, each peer has access to only a subset of peers, namely, its local acquaintances [6]. In addition, many web applications require the protection of private information in their ontologies; hence, those applications only provide limited query interfaces instead of exposing their ontologies explicitly. In both scenarios, integration of all ontologies is not possible.

In response to these needs, Package-based Description Logics (P-DL) [5] allows context-preserving knowledge reuse between description logic ontologies connected by *importing* relations. This paper presents a federated reasoning algorithm for P-DL $\mathcal{ALCP_C}$, that allows importing of concepts between ontologies, which overcomes many of these limitations and offers several advantages over existing approaches. By using distributed reasoning with localized P-DL semantics, the algorithm avoids combining the local ontology modules in a centralized memory space, thereby allowing local reasoning modules to operate in a peer-to-peer fashion. The P-DL semantics also guarantees that the results of reasoning in the distributed setting are identical to those obtainable by applying a reasoner to an ontology constructed by integrating the different modules [5].

One reason for which description logics enjoy good computational properties, e.g., being robustly decidable, is that they have the tree model property [27,12], i.e., if the ontology in question is consistent, it has at least one model which has a tree-shaped relational structure. Hence, a tableau algorithm for DL may decide the consistency of an ontology by searching for the existence of such a tree-shaped model, or a *completion graph*[1]. P-DL, as an extension of DL, still enjoys the tree-model property, but in a distributed fashion. If a P-DL ontology is consistent, it has a *distributed model* such that each local model (tableau) of it (for a component module of the ontology) is a forest, and all those local models can be seen as fragments of a conceptual, tree-shaped "global model". The P-DL tableau algorithm is motivated by the desire to discover such a model using a federation of local reasoners, each maintaining a local tableau, by message exchanging between those reasoners.

In this paper, we focus on algorithmic design rather than on implementation details. The latter may include the communication protocols between the local reasoners and aspects of the process of synchronization and backtracking, such as, e.g., handshaking and acknowledgement protocols, remembering of previous choices, dependency between choices and the token passing protocol. We leave those details to the implementation of the algorithm, that is expected to be influenced by experimental studies for best performance. Some of those techniques have already been applied in popular DL reasoners, e.g., Pellet [22].

The rest of the paper is organized as follows. Section 2 briefly reviews the syntax and semantics of P-DL and the basic tableau algorithm for the description logic $\mathcal{ALC}$. Section 3 presents the tableau data structure for $\mathcal{ALCP_C}$. In Section 4, the reasoning algorithm for modular ontologies with acyclic importing is presented. Section 5 extends this algorithm to cover modular ontologies with cyclic importing. In Section 6, related work is discussed and, finally, Section 7 concludes with a summary.

2. Preliminaries

We start by briefly reviewing the syntax and semantics of $\mathcal{ALCP_C}$ [4] and the tableau algorithm for $\mathcal{ALC}$. We assume that the reader is familiar with the basic theory of description logics.

2.1. $\mathcal{ALCP_C}$

Informally, a package in $\mathcal{ALCP_C}$ can be viewed as an extended $\mathcal{ALC}$ TBox. We define the *signature* $\mathsf{Sig}(P_i)$ of a package P_i as the set of names used in P_i. $\mathsf{Sig}(P_i)$ is the disjoint union of the set of concept names NC_i and the set of role names NR_i, used in package P_i.

The set of $\mathcal{ALCP_C}$ concepts in P_i is defined inductively by the following grammar:

$$C := A|\neg_k C|C \sqcap C|C \sqcup C|\forall R.C|\exists R.C$$

[1]In some expressive DLs, such as the ones with transitive roles, the completion graph is a tree-shaped skeleton of a model from which the model can be reconstructed. In DLs with nominals, the completion graph may not be a tree but a forest.

where $A \in \mathsf{NC}_i$, $R \in \mathsf{NR}_i$; $\neg_k C$ denotes the *contextualized negation* of concept C *w.r.t.* P_k. For any k and k-concept name C, $\top_k = \neg_k C \sqcup C$, and $\bot = \neg_k C \sqcap C$. Thus, there is no universal top concept ($\top$) or global negation ($\neg$). Instead, we have, for each package P_k, a contextualized top $\top_k$ and a contextualized negation $\neg_k$. This allows a logical formula in P-DL, including $\mathcal{ALCP_C}$, to be interpreted within the context of a specific package.

A *general concept inclusion* (GCI) *axiom* in P_i is an expression of the form $C \sqsubseteq D$, where C, D are concepts in P_i. Thus, formally, a *package* P_i in $\mathcal{ALCP_C}$ is the set of all GCIs in P_i, i.e., its TBox $\mathcal{T}_i$. An $\mathcal{ALCP_C}$ ontology Σ is a set of packages $\{P_i\}$. We assume that every name used in an $\mathcal{ALCP_C}$ ontology Σ has a home package in Σ.

The signature $\mathsf{Sig}(P_i)$ of package P_i is divided into two disjoint parts: its local signature $\mathsf{Loc}(P_i)$ and its external signature $\mathsf{Ext}(P_i)$. For all $t \in \mathsf{Loc}(P_i)$, P_i is the *home package* of t, denoted by $P_i = \mathsf{Home}(t)$, and t is called an *i-name*; more specifically, an *i-concept name* or an *i-role name*. If a concept name $t \in \mathsf{Loc}(P_j) \cap \mathsf{Ext}(P_i)$, $i \neq j$, we say that P_i *imports* t and denote it as $P_j \xrightarrow{t} P_i$. If any local name of P_j is imported into P_i or $\neg_j$ is used in P_i, we say that P_i imports P_j and denote it by $P_j \mapsto P_i$.

The *importing transitive closure* of a package P_i, denoted by P_i^+, is the set of all packages that are directly or indirectly imported by P_i. Let $P_i^* = \{P_i\} \cup P_i^+$. An $\mathcal{ALCP_C}$ ontology $\Sigma = \{P_i\}$ has an *acyclic importing relation* if, for all $i \neq j$, $P_j \in P_i^+ \rightarrow P_i \notin P_j^+$; otherwise, it has a *cyclic importing relation.* We denote by $\mathcal{ALCP}_\mathcal{C}^-$ a restricted type of $\mathcal{ALCP_C}$, namely, that with acyclic importing.

A concept C is *understandable* by a package P_i if each name occurring in C has a home package in P_i^* and for each k-negation occurring in C, $P_k \mapsto P_i$.

An $\mathcal{ALCP_C}$ ontology has *localized semantics* in the sense that each package has its own local interpretation domain. Formally, for an $\mathcal{ALCP_C}$ ontology $\Sigma = \{P_i\}$, a *distributed interpretation* is a tuple $\mathcal{I} = \langle \{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^+} \rangle$, where $\mathcal{I}_i$ is the *local interpretation* of package P_i, with domain $\Delta^{\mathcal{I}_i}$, and $r_{ij} \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$ is the *(image) domain relation* for the interpretation of the direct or indirect importing relation from P_i to P_j. For convenience, we use $r_{ii} = \{(x,x) | x \in \Delta^{\mathcal{I}_i}\}$ to denote the identity mapping on the local domain $\Delta^{\mathcal{I}_i}$.

Given i, j, such that $P_i \in P_j^*$, define:

$$r_{ij}(A) = \{y \in \Delta^{\mathcal{I}_j} | \exists x \in A, (x,y) \in r_{ij}\}, \text{ for every } A \subseteq \Delta^{\mathcal{I}_i}.$$

Moreover, let ρ be the equivalence relation on $\bigcup_i \Delta^{\mathcal{I}_i}$ generated by the collection of all domain relations, i.e., the symmetric and transitive closure of the set $\bigcup_{P_i \in P_j^*} r_{ij}$. For every i, j such that $P_i \in P_j^*$, $\rho_{ij} = \rho \cap (\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j})$.

Each of the local interpretations $\mathcal{I}_i = \langle \Delta^{\mathcal{I}_i}, \cdot^{\mathcal{I}_i} \rangle$ consists of a domain $\Delta^{\mathcal{I}_i}$ and an interpretation function $\cdot^{\mathcal{I}_i}$, which maps every concept name to a subset of $\Delta^{\mathcal{I}_i}$ and every role name to a subset of $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i}$, such that the following equations are satisfied, where R is an i-role name and C, D are concepts:

$$(C \sqcap D)^{\mathcal{I}_i} = C^{\mathcal{I}_i} \cap D^{\mathcal{I}_i}$$
$$(C \sqcup D)^{\mathcal{I}_i} = C^{\mathcal{I}_i} \cup D^{\mathcal{I}_i}$$
$$(\neg_j C)^{\mathcal{I}_i} = r_{ji}(\Delta^{\mathcal{I}_j}) \backslash C^{\mathcal{I}_i}$$
$$(\exists R.C)^{\mathcal{I}_i} = \{x \in \Delta^{\mathcal{I}_i} | (\exists y \in \Delta^{\mathcal{I}_i})((x,y) \in R^{\mathcal{I}_i} \wedge y \in C^{\mathcal{I}_i})\}$$
$$(\forall R.C)^{\mathcal{I}_i} = \{x \in \Delta^{\mathcal{I}_i} | (\forall y \in \Delta^{\mathcal{I}_i})((x,y) \in R^{\mathcal{I}_i} \rightarrow y \in C^{\mathcal{I}_i})\}$$

Note that, when $i = j$, $(\neg_j C)^{\mathcal{I}_i}$ reduces to the usual negation $(\neg_i C)^{\mathcal{I}_i} = \Delta^{\mathcal{I}_i} \backslash C^{\mathcal{I}_i}$.

A local interpretation $\mathcal{I}_i$ is said to *satisfy* a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$. $\mathcal{I}_i$ is called a *model* of P_i, denoted by $\mathcal{I}_i \vDash P_i$, if it satisfies all axioms in P_i.

Definition 1 *An interpretation $\mathcal{I} = \langle \{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^*} \rangle$ is a* model *of an $\mathcal{ALCP_C}$ KB $\Sigma = \{P_i\}$, denoted by $\mathcal{I} \vDash \Sigma$, if the following conditions are satisfied.*

1. *For all i, j, r_{ij} is one-to-one, i.e., it is an injective partial function;*

2. Compositional Consistency*: For all i, j, k, $i \neq j$, s.t. $P_i \in P_k^*$ and $P_k \in P_j^*$, we have $\rho_{ij} = r_{ij} = r_{kj} \circ r_{ik}$;*
3. *For every i-concept name C that appears in P_j, we have $r_{ij}(C^{\mathcal{I}_i}) = C^{\mathcal{I}_j}$;*
4. *$\mathcal{I}_i \vDash P_i$, for every i.*

Note that if $P_j \notin P_i^*$, r_{ji} does not exist even if r_{ij} exists. Moreover, we have that $r_{ij} = r_{ji}^-$ if P_i and P_j mutually import one another. Also note that r_{ij} may not be a total function.

Definition 2 *An ontology Σ is* consistent as witnessed by a package P_i *of Σ if P_i^* has a model $\mathcal{I} = \langle \{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^+} \rangle$, such that $\Delta^{\mathcal{I}_i} \neq \emptyset$. A concept C is* satisfiable as witnessed by P_i *if there is a model of P_i^*, such that $C^{\mathcal{I}_i} \neq \emptyset$. A concept subsumption $C \sqsubseteq D$* is valid as witnessed by P_i, *denoted by $C \sqsubseteq_i D$, if for every model of P_i^*, $C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$. We use $C \equiv_j D$ as the abbreviation of $C \sqsubseteq_j D$ and $D \sqsubseteq_j D$.*

Hence, in $\mathcal{ALCP_C}$, consistency, satisfiability and subsumption problems are always answered from the local point of view of a *witness package*, and it is possible for different packages to draw different conclusions from their own points of view.

2.2. Tableau Algorithm for $\mathcal{ALC}$

Modern description logics exploit tableau algorithms [3] for deciding concept satisfiability with respect to (w.r.t.) a knowledge base. For an $\mathcal{ALC}$ ontology O and an $\mathcal{ALC}$-concept C, a tableau algorithm will construct a common model for both O and C. If one such model, represented as a *completion graph*, is found, C is satisfiable w.r.t. O, otherwise C is unsatisfiable w.r.t. O.

Before the reasoning process starts, the concepts in O and C should be transformed into *Negation Normal Form* (NNF), i.e., with negation only occurring in front of concept names, using the following rewriting rules:

$$\begin{array}{cc} \neg\neg C \equiv C & \neg(C \sqcap D) \equiv \neg C \sqcup \neg D \\ \neg(C \sqcup D) \equiv \neg C \sqcap \neg D & \neg\exists R.C \equiv \forall R.\neg C \\ \neg\forall R.C \equiv \exists R.\neg C & \end{array}$$

Reasoning w.r.t. a TBox $\mathcal{T}$ can be reduced to reasoning w.r.t. an empty TBox with the *internalization* technique. Given $\mathcal{T}$, a concept $C_{\mathcal{T}}$ is defined as $C_{\mathcal{T}} = \underset{(C_i \sqsubseteq D_i) \in \mathcal{T}}{\sqcap} (\neg C_i \sqcup D_i)$. Any individual x in any model of $\mathcal{T}$ will be an instance of $C_{\mathcal{T}}$.

A *completion graph* or a *tableau* $T = \langle V, E, \mathcal{L} \rangle$ for $\mathcal{ALC}$ is a tree, where V is the node set, E is the edge set and $\mathcal{L}$ is a function that assigns a label to each node and each edge. Each node x in the tree represents an individual in the domain of the model and its label $\mathcal{L}(x)$ contains all concepts of which x is an instance. Each edge $\langle x, y \rangle$, on the other hand, represents a set of role instances in the model and its label $\mathcal{L}(\langle x, y \rangle)$ contains the names of the roles of which $\langle x, y \rangle$ is an instance. If $R \in \mathcal{L}(\langle x, y \rangle)$, y is an *R-successor* of x. An $\mathcal{ALC}$-tableau satisfies the following conditions:

(A0) for every $x \in V$, $C_{\mathcal{T}} \in \mathcal{L}(x)$;
(A1) if $C \in \mathcal{L}(x)$, then $\neg C \notin \mathcal{L}(x)$;
(A2) if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, then $C_1 \in \mathcal{L}(x)$ and $C_2 \in \mathcal{L}(x)$;
(A3) if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, then $C_1 \in \mathcal{L}(x)$ or $C_2 \in \mathcal{L}(x)$;
(A4) if $\forall R.C \in \mathcal{L}(x)$ and $R \in \mathcal{L}(\langle x, y \rangle)$, then $C \in \mathcal{L}(y)$;
(A5) if $\exists R.C \in \mathcal{L}(x)$, then, there exists $y \in V$, such that $R \in \mathcal{L}(\langle x, y \rangle)$ and $C \in \mathcal{L}(y)$.

Given a concept C and a TBox $\mathcal{T}$, the tableau is a tree expanded from an initial root node x_0, with $\mathcal{L}(x_0) = C \sqcap C_{\mathcal{T}}$, using the following *expansion rules*:

- CE-rule: if $C_{\mathcal{T}} \notin \mathcal{L}(x)$, then $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_{\mathcal{T}}\}$;
- $\sqcap$-rule: if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$, then $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$;
- $\sqcup$-rule: if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$, then $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1\}$ or $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2\}$;

- $\exists$-rule: if $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and x has no R-successor y with $C \in \mathcal{L}(y)$, then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{R\}$ and $\mathcal{L}(y) = \{C\}$;
- $\forall$-rule: if $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and there is an R-successor y of x with $C \notin \mathcal{L}(y)$, then $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$.

To ensure termination, a node can be blocked with the *subset blocking* strategy: for any node x, if there is an ancestor node y of x in the tree, and $\mathcal{L}(x) \subseteq \mathcal{L}(y)$, x is blocked. No expansion rule will be applied to a blocked node.

An $\mathcal{ALC}$ tableau contains a *clash* if $\{C, \neg C\} \subseteq \mathcal{L}(x)$ for some node x and concept C. A tableau is *consistent* if it contains no clash, and is *complete* if no expansion rule can be applied. The given concept is satisfiable if and only if the algorithm finds a consistent and complete tableau.

3. A Tableau for $\mathcal{ALCP_C}$

We first introduce the notion of tableau for $\mathcal{ALCP_C}$.

Before the reasoning process starts, all concepts are converted into negation normal form (NNF), i.e., a form in which negation only occurs before concept names, including local "tops", and there are only j-negations in a package P_j. We use $\dot{\neg}_i C$ to denote the NNF of $\neg_i C$. We can transform formulae in P_j into NNF by applying the following rules:

$$\begin{array}{ll}
\neg_i(\neg_k D) \Rightarrow \top_i \sqcap (D \sqcup \neg_i \top_k) & \neg_i C \Rightarrow \top_i \sqcap \neg_j C, \text{ where } C \text{ is a concept name or a local top,} \\
\neg_i(C_1 \sqcap C_2) \Rightarrow \neg_i C_1 \sqcup \neg_i C_2 & \neg_i(C_1 \sqcup C_2) \Rightarrow \neg_i C_1 \sqcap \neg_i C_2 \\
\neg_i \exists R.D \Rightarrow \top_i \sqcap \forall R.\neg_j D, & \neg_i \forall R.D \Rightarrow \top_i \sqcap \exists R.\neg_j D \\
\neg_i \bot \Rightarrow \top_i & \neg_i \top_i \Rightarrow \bot
\end{array}$$

Lemma 1 *For any concept C in a package P_j and for any i such that $P_i \mapsto P_j$, $\dot{\neg}_i C \equiv_j \neg_i C$.*

Proof: The statement is obvious if $C = \top_i$ or $C = \bot$. For any model $\mathcal{I}$ of P_j^*, we have:

- if C is a concept name or a local top concept, $(\dot{\neg}_i C)^{\mathcal{I}_j} = (\top_i \sqcap (\neg_j C))^{\mathcal{I}_j} = r_{ij}(\Delta^{\mathcal{I}_i}) \cap \Delta^{\mathcal{I}_j} \backslash C^{\mathcal{I}_j} = r_{ij}(\Delta^{\mathcal{I}_i}) \backslash C^{\mathcal{I}_j} = (\neg_i C)^{\mathcal{I}_j}$;
- if $C = \neg_k D$, then

$$\begin{aligned}
(\dot{\neg}_i C)^{\mathcal{I}_j} &= (\top_i \sqcap (D \sqcup \neg_i \top_k))^{\mathcal{I}_j} && \text{(by NNF transformation rules)} \\
&= \top_i^{\mathcal{I}_j} \cap (D^{\mathcal{I}_j} \cup (\neg_i \top_k)^{\mathcal{I}_j}) && \text{(by the definition of } \cdot^{\mathcal{I}_j}) \\
&= r_{ij}(\Delta^{\mathcal{I}_i}) \cap (D^{\mathcal{I}_j} \cup (r_{ij}(\Delta^{\mathcal{I}_i}) \backslash r_{kj}(\Delta^{\mathcal{I}_k}))) && \text{(by the definition of } \mathcal{I}) \\
&= (r_{ij}(\Delta^{\mathcal{I}_i}) \backslash (r_{kj}(\Delta^{\mathcal{I}_k}) \backslash D^{\mathcal{I}_j})) && \text{(set-theorectically)} \\
&= (\neg_i(\neg_k D))^{\mathcal{I}_j} && \text{(by the definition of } \cdot^{\mathcal{I}_j}) \\
&= (\neg_i C)^{\mathcal{I}_j}; && \text{(since } C = \neg_k D)
\end{aligned}$$

- if $C = C_1 \sqcap C_2$, then

$$\begin{aligned}
(\dot{\neg}_i C)^{\mathcal{I}_j} &= (\neg_i C_1 \sqcup \neg_i C_2)^{\mathcal{I}_j} && \text{(by NNF transformation rules)} \\
&= (r_{ij}(\Delta^{\mathcal{I}_i}) \backslash C_1^{\mathcal{I}_j}) \cup (r_{ij}(\Delta^{\mathcal{I}_i}) \backslash C_2^{\mathcal{I}_j}) && \text{(by the definition of } \cdot^{\mathcal{I}_j}) \\
&= r_{ij}(\Delta^{\mathcal{I}_i}) \backslash (C_1^{\mathcal{I}_j} \cap C_2^{\mathcal{I}_j}) && \text{(set-theorectically)} \\
&= (\neg_i(C_1 \sqcap C_2))^{\mathcal{I}_j} && \text{(by the definition of } \cdot^{\mathcal{I}_j}) \\
&= (\neg_i C)^{\mathcal{I}_j}; && \text{(since } C = C_1 \sqcap C_2)
\end{aligned}$$

- if $C = C_1 \sqcup C_2$, the proof is similar;
- if $C = \exists R.D$, then

$$
\begin{aligned}
(\dot{\neg}_i C)^{\mathcal{I}_j} &= (\top_i \sqcap \forall R.\neg_j D)^{\mathcal{I}_j} && \text{(by NNF transformation rules)}\\
&= \{x \in r_{ij}(\Delta^{\mathcal{I}_i}) \cap \Delta^{\mathcal{I}_j} | (\forall y \in \Delta^{\mathcal{I}_j})((x,y) \in R^{\mathcal{I}_j} \to y \in (\neg_j D)^{\mathcal{I}_j})\} && \text{(by the definition of } \cdot^{\mathcal{I}_j}\text{)}\\
&= \{x \in r_{ij}(\Delta^{\mathcal{I}_i}) | (\forall y \in \Delta^{\mathcal{I}_j})((x,y) \in R^{\mathcal{I}_j} \to y \notin D^{\mathcal{I}_j})\} && \text{(by the definition of } \cdot^{\mathcal{I}_j}\text{)}\\
&= r_{ij}(\Delta^{\mathcal{I}_i}) \backslash \{x \in \Delta^{\mathcal{I}_j} | (\exists y \in \Delta^{\mathcal{I}_j})((x,y) \in R^{\mathcal{I}_j} \wedge y \in D^{\mathcal{I}_j})\} && \text{(set-theorectically)}\\
&= (\neg_i(\exists R.D))^{\mathcal{I}_j} && \text{(by the definition of } \cdot^{\mathcal{I}_j}\text{)}\\
&= (\neg_i C)^{\mathcal{I}_j}; && \text{(since } C = \exists R.C\text{)}
\end{aligned}
$$

– if $C = \forall R.D$, the proof is similar to the previous case.

Hence, $(\dot{\neg}_i C)^{\mathcal{I}_j} = (\neg_i C)^{\mathcal{I}_j}$ holds, for every model $\mathcal{I}$ of P_j^*, whence $\dot{\neg}_i C \equiv_j \neg_i C$. Q.E.D.

The main idea behind the $\mathcal{ALCP_C}$ tableau algorithm is to construct multiple, federated local tableaux using only knowledge locally available to each module, instead of creating a single tableau using the integrated ontology resulting by combining all those modules. A set of messages will be exchanged between the local modules to connect the local tableaux by creating partial correspondences between them. Formally, we have:

Definition 3 *The set of subconcepts $sub(C)$ of an $\mathcal{ALCP_C}$ concept C in NNF is inductively defined by:*

$$
\begin{aligned}
sub(A) &= \{A\},\ \textit{for a concept name , including a local top concept, or its negation } A\\
sub(C \sqcap D) &= \{C \sqcap D\} \cup sub(C) \cup sub(D)\\
sub(C \sqcup D) &= \{C \sqcup D\} \cup sub(C) \cup sub(D)\\
sub(\exists R.C) &= \{\exists R.C\} \cup sub(C)\\
sub(\forall R.C) &= \{\forall R.C\} \cup sub(C)
\end{aligned}
$$

For every package P_i, we define $C_{\mathcal{T}_i} = \bigsqcap_{(C \sqsubseteq D) \in \mathcal{T}_i} (\dot{\neg}_i C \sqcup D)$.

Definition 4 *Let P_w be a witness package and D be an $\mathcal{ALCP_C}$-concept in NNF w.r.t. P_w, such that D is understandable by P_w. A* distributed tableau *for D* w.r.t. *P_w is a tuple $T = \langle \{T_i\}, \{t_{ij}\}_{P_i \in P_j^+} \rangle$, where each T_i is a local tableau, for $P_i \in P_w^*$, and t_{ij} is the tableau relation from a local tableau T_i to a local tableau T_j. Each local tableau is a tuple $T_i = (\mathbf{S}_i, \mathcal{L}_i, \mathcal{E}_i)$, where*

- *$\mathbf{S}_i$ is a set of individuals,*
- *$\mathcal{L}_w : \mathbf{S}_w \to 2^{sub(D) \cup sub(C_{\mathcal{T}_w})}$ and $\mathcal{L}_i : \mathbf{S}_i \to 2^{sub(C_{\mathcal{T}_i})}$, $i \neq w$, map individuals to corresponding sets of concepts,*
- *$\mathcal{E}_i : \mathsf{NR}_i \to 2^{\mathbf{S}_i \times \mathbf{S}_i}$ maps roles to the corresponding sets of pairs of individuals.*

Each tableau relation t_{ij} is a subset of $\mathbf{S}_i \times \mathbf{S}_j$. Let ρ^t be the symmetric and transitive closure of the set $\bigcup_{P_i \in P_j^*} t_{ij}$. And, for all i, j, such that $P_i \in P_j^*$, set $\rho_{ij}^t = \rho^t \cap (\mathbf{S}_i \times \mathbf{S}_j)$.

The distributed tableau T should satisfy the following conditions:

(E) *there exists $x \in \mathbf{S}_w$, such that $D \in \mathcal{L}_w(x)$;*
(A0) *for every $x \in \mathbf{S}_i$, $C_{\mathcal{T}_i} \in \mathcal{L}_i(x)$;*
(A1) *if $C \in \mathcal{L}_i(x)$, then $\neg_i C \notin \mathcal{L}_i(x)$;*
(A2) *if $C_1 \sqcap C_2 \in \mathcal{L}_i(x)$, then $C_1 \in \mathcal{L}_i(x)$ and $C_2 \in \mathcal{L}_i(x)$;*
(A3) *if $C_1 \sqcup C_2 \in \mathcal{L}_i(x)$, then $C_1 \in \mathcal{L}_i(x)$ or $C_2 \in \mathcal{L}_i(x)$;*
(A4) *if $\forall R.C \in \mathcal{L}_i(x)$ and $\langle x, y \rangle \in \mathcal{E}_i(R)$, then $C \in \mathcal{L}_i(y)$;*
(A5) *if $\exists R.C \in \mathcal{L}_i(x)$, then, there exists $y \in \mathbf{S}_i$, such that $\langle x, y \rangle \in \mathcal{E}_i(R)$ and $C \in \mathcal{L}_i(y)$;*
(B1) *t_{ij} is a one-to-one partial function, for all i, j;*
(B2) *$\rho_{ij}^t = t_{ij} = t_{kj} \circ t_{ik}$ for all i, j, k, $i \neq j$, such that $P_i \in P_k^*$ and $P_k \in P_j^*$;*

(B3) *if C is an i-concept name, $P_i \xrightarrow{C} P_j, i \neq j$, then*

$$(\forall x' \in \mathbf{S}_j)((\exists x \in \mathbf{S}_i)(\langle x, x' \rangle \in t_{ij} \textit{ and } C \in \mathcal{L}_i(x)) \textit{ iff } C \in \mathcal{L}_j(x'));$$

Explanation: Conditions (A0)-(A5) are similar to the ones used in the tableau definition of $\mathcal{ALC}$ [16]. Intuitively, Conditions (B1) and (B2) ensure that domain relations are one-to-one and compositionally consistent. On the other hand, Condition (B3) ensures that $r_{ij}(C^{\mathcal{I}_i}) = C^{\mathcal{I}_j}$, for any concept name C.

It should be noted that the correspondence of individuals across multiple local tableaux is only *partial*. Some individuals in a local tableau may not be connected to any individuals in another local tableau. This conforms with the localized semantics of P-DL stipulating that each ontology module has its own interpretation domain.

The following lemma establishes the correspondence between concept satisfiability, hence, also between TBox consistency and concept subsumption, and the existence of a tableau for that concept in $\mathcal{ALCP_C}$:

Lemma 2 *Let D be an $\mathcal{ALCP_C}$ concept that is understandable by an $\mathcal{ALCP_C}$ package P_w. Then D is satisfiable as witnessed by P_w iff D has a distributed tableau w.r.t. P_w.*

Proof: For the "if" direction, suppose that $\langle \{T_i\}, \{t_{ij}\}_{P_i \in P_j^+} \rangle$, with $T_i = (\mathbf{S}_i, \mathcal{L}_i, \mathcal{E}_i)$, is a tableau for D w.r.t. P_w^*. Then, a model $\mathcal{I} = \langle \{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^+} \rangle$ of P_w^* may be defined as follows:

$$\Delta^{\mathcal{I}_i} = \mathbf{S}_i;$$

$$A^{\mathcal{I}_i} = \{x | A \in \mathcal{L}_i(x)\}, \text{ for every concept name } A;$$

$$R^{\mathcal{I}_i} = \mathcal{E}_i(R), \text{ for every } i\text{-role name } R;$$

$$r_{ij} = t_{ij}.$$

By using induction on the structure of concepts, we show that

$$C \in \mathcal{L}_i(x) \quad \text{implies} \quad x \in C^{\mathcal{I}_i}. \tag{1}$$

- If C is a concept name, then the statement follows by the definition of $C^{\mathcal{I}_i}$.
- If $C = \neg_i E$, where E is a concept name, then, by Property (A1) of the tableau, $E \notin \mathcal{L}_i(x)$, whence, by the definition of $E^{\mathcal{I}_i}$, $x \notin E^{\mathcal{I}_i}$ and, hence, $x \in \Delta^{\mathcal{I}_i} \backslash E^{\mathcal{I}_i} = C^{\mathcal{I}_i}$.
- If $C = C_1 \sqcap C_2$, then, by Property (A2), $C_1 \in \mathcal{L}_i(x)$ and $C_2 \in \mathcal{L}_i(x)$, whence, by the induction hypothesis, $x \in C_1^{\mathcal{I}_i}$ and $x \in C_2^{\mathcal{I}_i}$ and, therefore, $x \in (C_1 \sqcap C_2)^{\mathcal{I}_i}$.
- The case $C = C_1 \sqcup C_2$ may be handled similarly.
- If $C = \forall R.E$ and $\langle x, y \rangle \in R^{\mathcal{I}_i}$, then $\langle x, y \rangle \in \mathcal{E}_i(R)$ and, by Property (A4), $E \in \mathcal{L}_i(y)$, whence, by the induction hypothesis, $y \in E^{\mathcal{I}_i}$ and, hence, $x \in (\forall R.E)^{\mathcal{I}_i}$.
- If $C = \exists R.E$, then, by Property (A5), there exists $y \in \mathbf{S}_i$, such that $\langle x, y \rangle \in \mathcal{E}_i(R)$ and $E \in \mathcal{L}_i(y)$, whence, by definition, $\langle x, y \rangle \in R^{\mathcal{I}_i}$ and, by the induction hypothesis, $y \in E^{\mathcal{I}_i}$, and, therefore, $x \in (\exists R.E)^{\mathcal{I}_i}$.

Next, using Implication (1), it is shown that all $\mathcal{ALCP_C}$ restrictions on domain relations are satisfied.

- First, $D^{\mathcal{I}_w}$ is not empty, since there exists, by hypothesis, $x \in \mathbf{S}_w$, such that $D \in \mathcal{L}_w(x)$.
- The image domain relations r_{ij} are one-to-one and compositionally consistent by tableau Properties (B1) and (B2).
- For every concept importing $P_i \xrightarrow{C} P_j$, where C is an i-concept name, we have $r_{ij}(C^{\mathcal{I}_i}) = C^{\mathcal{I}_j}$, by Property (B3).
- For every $P_i \in P_w^*$, every axiom $C \sqsubseteq D \in P_i$ and every individual $x \in \mathbf{S}_i$, we have, using tableau Properties (A0) and (A2), that $\neg_i C \sqcup D \in \mathcal{L}_i(x)$. Thus, by Property (A3), either $\neg_i C \in \mathcal{L}_i(x)$ or $D \in \mathcal{L}_i(x)$. Hence, by Implication (1), $x \notin C^{\mathcal{I}_i}$ or $x \in D^{\mathcal{I}_i}$, whence $C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$, and, therefore, $\mathcal{I}_i \models P_i$.

For the "only if" direction, if $\mathcal{I} = \langle\{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^+}\rangle$ is a model of P_w^*, with $C^{\mathcal{I}_w} \neq \emptyset$, then a tableau $T = \langle\{T_i\}, \{t_{ij}\}_{P_i \in P_j^+}\rangle$ for P_w^* may be defined as follows:

$$\begin{aligned}
\mathbf{S}_i &= \Delta^{\mathcal{I}_i};\\
\mathcal{L}_i(x) &= \{C \in sub(C_{\mathcal{T}_i}) | x \in C^{\mathcal{I}_i}\}, x \in \Delta^{\mathcal{I}_i}, i \neq w;\\
\mathcal{L}_w(x) &= \{C \in sub(D) \cup sub(C_{\mathcal{T}_w}) | x \in C^{\mathcal{I}_w}\}, x \in \Delta^{\mathcal{I}_w};\\
\mathcal{E}_i(R) &= R^{\mathcal{I}_i};\\
t_{ij} &= r_{ij}.
\end{aligned}$$

We now verify that T is indeed a tableau for D w.r.t. P_w, i.e., that it satisfies all conditions in Definition 2:

- **(E)**: Since $C^{\mathcal{I}_w} \neq \emptyset$, there exists $x \in \mathbf{S}_w$, such that $C \in \mathcal{L}_w(x)$.
- **(A0)**: Since $\mathcal{I}_i$ is a model of P_i, we have, for every $x \in \mathbf{S}_i$, $x \in C_{\mathcal{T}_i}^{\mathcal{I}_i}$, whence $C_{\mathcal{T}_i} \in \mathcal{L}_i(x)$.
- **(A1)**: If $C \in \mathcal{L}_i(x)$, then $x \in C^{\mathcal{I}_i}$, whence $x \notin (\neg_i C)^{\mathcal{I}_i} = \Delta^{\mathcal{I}_i} \backslash C^{\mathcal{I}_i}$, and, hence, $\neg_i C \notin \mathcal{L}_i(x)$.
- **(A2)**: If $C_1 \sqcap C_2 \in \mathcal{L}_i(x)$, then $x \in (C_1 \sqcap C_2)^{\mathcal{I}_i} = C_1^{\mathcal{I}_i} \cap C_2^{\mathcal{I}_i}$, hence $C_1 \in \mathcal{L}_i(x)$ and $C_2 \in \mathcal{L}_i(x)$.
- **(A3)**: The proof is similar to the previous one.
- **(A4)**: If $\forall R.C \in \mathcal{L}_i(x)$ and $\langle x, y\rangle \in \mathcal{E}_i(R)$, we have $x \in (\forall R.C)^{\mathcal{I}_i}$ and $\langle x, y\rangle \in R^{\mathcal{I}_i}$, whence, according to the semantics of $\forall R.C$, $y \in C^{\mathcal{I}_i}$ and, hence, $C \in \mathcal{L}_i(y)$.
- **(A5)**: If $\exists R.C \in \mathcal{L}_i(x)$, then there exists $y \in \Delta^{\mathcal{I}_i} = \mathbf{S}_i$, such that $\langle x, y\rangle \in R^{\mathcal{I}_i} = \mathcal{E}_i(R)$ and $y \in C^{\mathcal{I}_i}$, whence $C \in \mathcal{L}_i(y)$.
- **(B1)**: $t_{ij} = r_{ij}$ must be a one-to-one partial function, for all i, j.
- **(B2)**: By the compositional consistency of the r_{ij}, we have, for all i, j, k, $i \neq j$, such that $P_i \in P_k^*$ and $P_k \in P_j^*$, that $\rho_{ij} = r_{ij} = r_{kj} \circ r_{ik}$, whence, by the definition of $\{t_{ij}\}$, we have $\rho_{ij}^t = t_{ij} = t_{kj} \circ t_{ik}$.
- **(B3)**: If C is an i-concept name, $P_i \xrightarrow{C} P_j$, $j \neq i$, then $r_{ij}(C^{\mathcal{I}_i}) = C^{\mathcal{I}_j}$, whence, since $t_{ij} = r_{ij}$, $(\forall x' \in \mathbf{S}_j)((\exists x \in \mathbf{S}_i)(\langle x, x'\rangle \in t_{ij}$ and $C \in \mathcal{L}_i(x))$ iff $C \in \mathcal{L}_j(x'))$.

Q.E.D.

4. A Tableau Algorithm for $\mathcal{ALCP}_{\mathcal{C}}^{-}$

We now proceed to describe a sound and complete algorithm to determine the existence of a tableau for an $\mathcal{ALCP}_{\mathcal{C}}$ concept w.r.t. a witness package. We start with the special case in which there is only acyclic importing between packages, i.e., $\mathcal{ALCP}_{\mathcal{C}}^{-}$. The algorithm allows each local tableau to be created and maintained by a local reasoner. Thus, reasoning is carried out by a federation of reasoners that communicate with each other via messages instead of a single reasoner over an integrated ontology.

4.1. Distributed Completion Graph

The algorithm works on a *distributed completion graph*, which is a partial finite description of a tableau. A distributed completion graph is $G = \{G_i\}$, where $\{G_i\}$ is a set of *local completion graphs*. Each *local completion graph* $G_i = \langle V_i, E_i, \mathcal{L}_i\rangle$ consists of a finite set of finite trees, i.e., a forest, where V_i and E_i are the corresponding sets of nodes and edges respectively, and of a function $\mathcal{L}_i$, that assigns labels to nodes and edges in G_i. Each node x in V_i represents an individual in the corresponding tableau, denoted as $i : x$, and is labeled with $\mathcal{L}_i(x)$, a set of concepts of which x is a member. Each edge $\langle x, y\rangle \in E_i$ represents a set of role memberships in the tableau, and is labeled with $\mathcal{L}_i(\langle x, y\rangle)$, the corresponding set of role names.

If $R \in \mathcal{L}_i(\langle x, y\rangle)$, y is said to be a *local R-successor* of x and x is said to be a *local R-predecessor* of y. *Local ancestors* and *local descendants* of a node are defined in the usual manner.

Every node x has associated with it a node $origin(x)$, which, informally speaking, is the "original" node from which x is "copied". If $origin(i : x) = origin(j : y)$ and $P_i \in P_j^+$, we say that node y in G_j is an *image* of node x in G_i, denoted by $y = x^{i \rightarrow j}$, that node x is a *pre-image* of y, denoted by $x = y^{i \leftarrow j}$, and that there is a *graph relation* $\langle x, y \rangle$.[2]

A typical distributed completion graph is shown in Figure 1. Dotted edges in the graph represent graph relations. If we merge nodes of the same origin, all local graphs may, in fact, be merged into a tree-shaped global graph. Tree(s) in a local graph are fragments of the corresponding (virtual) global tree. In fact, the virtual global tree represents a conceptual model for the ontology resulting by integrating all modules.

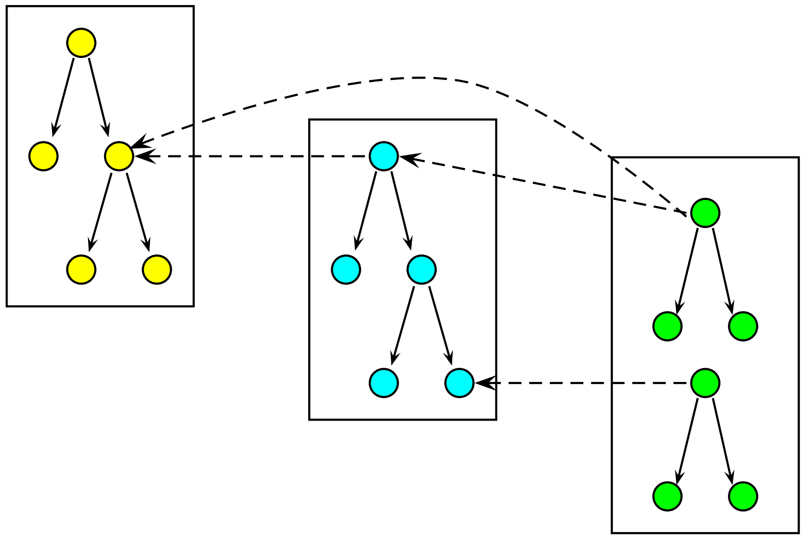


Fig. 1. $\mathcal{ALCP}_\mathcal{C}^-$ Distributed Tableaux Example

4.2. Distributed Tableau Expansion

A distributed $\mathcal{ALCP}_\mathcal{C}^-$ completion graph is constructed by applying a set of tableau expansion rules and by exchanging messages between local reasoners. The $\mathcal{ALCP}_\mathcal{C}^-$ expansion rules are adapted from the $\mathcal{ALC}$ expansion rules (see Section 2.2) as follows: Each module is only locally internalized, instead of being globally internalized with respect to a combined TBox. A local completion graph can create "copies" of its local nodes in another local completion graph, as needed, during an expansion.

A *concept reporting message* propagates concept labels of a node to the corresponding image node or pre-image node. We use $S += X$ to denote the operation of adding the elements of the set X to a set S, i.e., the operation $S = S \cup X$. Using this notation, we have:

- A *forward concept reporting message* $r^{i \rightarrow j}(x, C)$ executes the following action: **if** there is a node $x' \in V_j$, such that $origin(x) = origin(x')$ and $C \notin \mathcal{L}_j(x')$, **then** $\mathcal{L}_j(x') += \{C\}$.
- A *backward concept reporting message* $r^{j \leftarrow i}(x, C)$ executes the following actions: **if** there is a node $x' \in V_j$, such that $origin(x) = origin(x')$, then do $\mathcal{L}_j(x') += \{C\}$ if $C \notin \mathcal{L}_j(x')$ and $C \neq \top_j$; **else** create a node x' in V_j with $origin(x') = origin(x)$, and do $\mathcal{L}_j(x') += \{C\}$ if $C \neq \top_j$.

Some nodes in the graph may be *blocked*, as will be explained later. The expansion rules are:

- CE-rule: **if** $C_{\mathcal{T}_i} \notin \mathcal{L}_i(x)$, **then** $\mathcal{L}_i(x) += C_{\mathcal{T}_i}$.

[2]Sometimes we use the same name with different prefixes for two nodes to indicate that they have the same origin, e.g., $i : x$ and $j : x$ means $origin(i : x) = origin(j : x)$. We may omit the prefix when it is clear from the context.

– $\sqcap$-rule: **if** $C_1 \sqcap C_2 \in \mathcal{L}_i(x)$, x is not blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}_i(x)$, **then** $\mathcal{L}_i(x)$+= $\{C_1, C_2\}$.
– $\sqcup$-rule: **if** $C_1 \sqcup C_2 \in \mathcal{L}_i(x)$, x is not blocked, and $\{C_1, C_2\} \cap \mathcal{L}_i(x) = \emptyset$, **then** $\mathcal{L}_i(x)$+= $\{C\}$ for some $C \in \{C_1, C_2\}$.
– $\exists$-rule: **if** $\exists R.C \in \mathcal{L}_i(x)$, x is not blocked and x has no local R-successor y of x in G_i with $C \in \mathcal{L}_i(y)$, **then** create a new node y with $orgin(y) = y$, $\mathcal{L}_i(\langle x, y\rangle) = \{R\}$ and $\mathcal{L}_i(y) = \{C\}$.
– $\forall$-rule: **if** $\forall R.C \in \mathcal{L}_i(x)$, x is not blocked and there is a local R-successor y of x in G_i with $C \notin \mathcal{L}_i(y)$, **then** $\mathcal{L}_i(y)$+= $\{C\}$.
– CPush-rule: **if** $C \in \mathcal{L}_i(x)$, where C is an i-concept name, with $P_i \xrightarrow{C} P_j$, x is not blocked and there exists an $x' = x^{i \to j} \in V_j$, such that x' is not blocked, with $C \notin \mathcal{L}_j(x')$, **then** transmit $r^{i \to j}(x, C)$.
– CReport-rule: **if** $C \in \mathcal{L}_i(x)$, where C is $\top_j$ or a j-concept name, x is not blocked and $x^{j \leftarrow i}$ does not exist or ($x' = x^{j \leftarrow i}$ exists, x' is not blocked and $C \notin \mathcal{L}_j(x')$), **then** transmit $r^{j \leftarrow i}(x, C)$.
– r-rule: **if** $origin(i : x) = origin(j : x')$, x, x' are not blocked, and there exists k such that $P_i \in P_k^+$, $P_k \in P_j^+$ and there is no $k : x''$ with $origin(j : x') = origin(k : x'')$, **then** transmit $r^{k \leftarrow j}(x', \top_k)$.

Explanation: The $\sqcap$-, $\sqcup$-, $\exists$-, $\forall$- and CE- rules are adaptations of the corresponding $\mathcal{ALC}$ expansion rules. The r-rule serves to ensure the compositional consistency of domain relations according to tableau Property (B2). The CPush- and CReport- rules are introduced to ensure $r_{ij}(C^{\mathcal{I}_i}) = C^{\mathcal{I}_j}$, for every i-concept name C, according to tableau Property (B3). The reader will get an even better feeling for the adoption of these rules while studying the soundness and completeness lemmas for the distributed algorithm, that will be presented later.

A distributed completion graph is *complete* if no $\mathcal{ALCP}_\mathcal{C}^-$ expansion rule can be applied to it, and it is *clash-free* if there is no x in any local completion graph G_i, such that both C and $\neg_i C$ are in $\mathcal{L}_i(x)$, for some concept C.

For a satisfiability query of a concept C as witnessed by a package P_w, where C is understandable by P_w, a local completion graph G_w, with an initial node x_0, such that $origin(x_0) = x_0$ and $\mathcal{L}_w(x_0) = \{C\}$, will be created first. The $\mathcal{ALCP}_\mathcal{C}^-$ tableau expansion rules will be applied until a complete and clash-free distributed completion graph is found or until all search efforts for such a distributed completion graph fail.

4.3. Blocking and Backtracking

When only acyclic importing among packages is considered, the termination and correctness of the algorithm can be obtained by using *subset blocking* and *token passing.*

Subset blocking has been applied in the $\mathcal{ALC}$ tableau algorithm [3]. The motivation behind subset blocking is the detection of cycles in tableau expansions. Formally, we have:

Definition 5 (Subset Blocking) *For a distributed completion graph of an $\mathcal{ALCP}_\mathcal{C}^-$ ontology, a node x is* directly blocked *by a node y, if both x and y are in the same local completion graph G_i, for some i, y is a local ancestor of x, and $\mathcal{L}_i(x) \subseteq \mathcal{L}_i(y)$. Node x is* indirectly blocked *by a node y if one of x's local ancestors is directly blocked by y. Node x is* blocked *by y if it is directly or indirectly blocked by y.*

Subset blocking in $\mathcal{ALCP}_\mathcal{C}^-$ only depends on the *local* information in completion graphs, i.e., a local completion graph determines blocking regardless of whether a node has any image or preimage nodes in any other local completion graphs and irrespective of the labels of those nodes. Thus, a node is blocked only by its *local* ancestors. As we will show in Example 4, subset blocking is required to guarantee the correctness of reasoning.

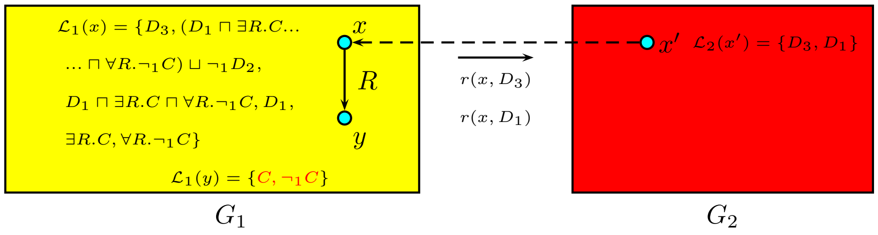
Token passing is used to coordinate expansions in different local completion graphs, as illustrated by the following example.

Example 1 : Suppose we have two packages:
$P_1 : \{\top_1 \sqsubseteq (2 : D_3), \top_1 \sqsubseteq ((2 : D_1) \sqcap \exists(1 : R).(1 : C) \sqcap \forall(1 : R).(\neg_1(1 : C))) \sqcup \neg_1(2 : D_2)\}$
$P_2 : \{(2 : D_1) \sqsubseteq (2 : D_2)\}$

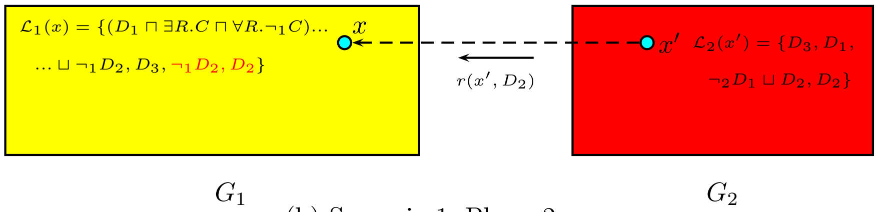
The reasoning task is to check the consistency of P_1. Figure 2 (a) and (b) show the running of $\mathcal{ALCP}_{\mathcal{C}}^{-}$ tableau expansions in one scenario, which will be referred to as Scenario 1. In (a), we first apply the CE-rule and $\sqcap$-rule, adding D_3 into $\mathcal{L}_1(x)$, which results in the firing of the CPeport-rule, the message $r^{2\leftarrow 1}(x, D_3)$ and the creation of x'. Next, due to the CE- and $\sqcup$-rule, we may choose adding $D_1 \sqcap \exists R.C \sqcap \forall R.\neg_1 C$ into $\mathcal{L}_1(x)$, which leads to a reporting message $r^{2\leftarrow 1}(x, D_1)$. Further applying expansion in G_1, we will generate node y and find a clash in $\mathcal{L}_1(y)$.

In Figure 2 (b), due to the clash, we will restore the status of node $1 : x$, as it had been before the choice in the $\sqcup$-rule was made, and try the next choice, i.e., adding $\neg_1 D_2$ into $\mathcal{L}_1(x)$. In the meantime, local completion graph G_2 may apply the CE- and $\sqcup$-rules and add D_2 into $\mathcal{L}_2(x')$. Since, by a domain relation that was established before the clash in Phase 1, x is an image node of x' and P_1 imports D_2, G_2 will apply the CPush-rule and send the message $r^{2\to 1}(x', D_2)$, which will lead to a clash in $\mathcal{L}_1(x)$. Hence, according to Scenario 1, we may assert that P_1 is not consistent since all choices in the tableau expansion of G_1 lead to clashes.

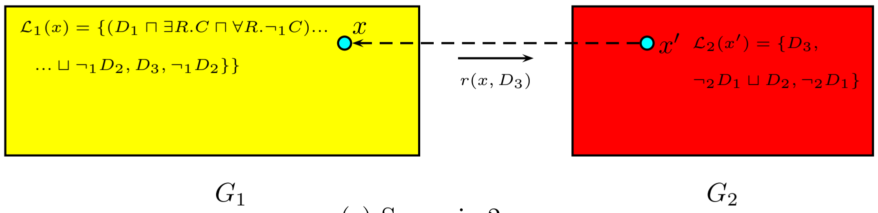
However, Figure 2 (c) shows another expansion scenario, which will be referred to as Scenario 2, that finds a consistent distributed completion graph. Hence, P_1 is actually consistent.


$\mathcal{L}_1(x) = \{D_3, (D_1 \sqcap \exists R.C \ldots$
$\ldots \sqcap \forall R.\neg_1 C) \sqcup \neg_1 D_2,$
$D_1 \sqcap \exists R.C \sqcap \forall R.\neg_1 C, D_1,$
$\exists R.C, \forall R.\neg_1 C\}$
$\mathcal{L}_1(y) = \{C, \neg_1 C\}$
x
R
y
$r(x, D_3)$
$r(x, D_1)$
x'
$\mathcal{L}_2(x') = \{D_3, D_1\}$
G_1
G_2

(a) Scenario 1, Phase 1


$\mathcal{L}_1(x) = \{(D_1 \sqcap \exists R.C \sqcap \forall R.\neg_1 C) \ldots$
$\ldots \sqcup \neg_1 D_2, D_3, \neg_1 D_2, D_2\}$
x
$r(x', D_2)$
x'
$\mathcal{L}_2(x') = \{D_3, D_1,$
$\neg_2 D_1 \sqcup D_2, D_2\}$
G_1
G_2

(b) Scenario 1, Phase 2


$\mathcal{L}_1(x) = \{(D_1 \sqcap \exists R.C \sqcap \forall R.\neg_1 C) \ldots$
$\ldots \sqcup \neg_1 D_2, D_3, \neg_1 D_2\}\}$
x
$r(x, D_3)$
x'
$\mathcal{L}_2(x') = \{D_3,$
$\neg_2 D_1 \sqcup D_2, \neg_2 D_1\}$
G_1
G_2

(c) Scenario 2

Fig. 2. The Need for Token Blocking

The problem with Scenario 1 in Example 1 is caused by the asynchronous operation of the different local reasoners. The message $r^{2\to 1}(x', D_2)$ in Phase 2 is in fact a consequence of the choice of adding $D_1 \sqcap \exists R.C \sqcap \forall R.\neg_1 C$ into $\mathcal{L}_1(x)$ and the subsequent message $r^{2\leftarrow 1}(x, D_1)$ in Phase 1. However, since local reasoners run autonomously and communication between them may be delayed, as it relies on network conditions, when the message $r^{2\to 1}(x', D_2)$ arrives at G_1, the previous choice, in which $r^{2\leftarrow 1}(x, D_1)$ was sent, has already been abandoned. Thus, the clash arising in Phase 1 is a false one, since it mixes consequences of different choices during the tableau expansion.

To avoid such problems, we resort to the techniques of *token passing* and of *clocks* in order to synchronize the creation of the local completion graphs by the local reasoners. The basic idea is to coordinate the expansion of all local completion graphs in such a way that, at any time, all node and edge labels in all local completion graphs always belong to the same sequence of non-deterministic choices.

Definition 6 (Local Clocks) *Clocks of local completion graphs are maintained in the following way:*

– *Every local completion graph G_i has a local* clock K_i *of integer type, initialized to 0.*
– *For every concept label C in $\mathcal{L}_i(x)$ where x is a node in G_i, there is a* timestamp $t_i(x, C)$ *of integer type. Informally speaking, this time stamp records the clock value of the last choice in the application of the $\sqcup$-rule before C was added into $\mathcal{L}_i(x)$, possibly in another local completion graph.*
– *For every role label R in $\mathcal{L}_i(\langle x, y\rangle)$, where $\langle x, y\rangle$ is an edge in G_i, there is a* timestamp $t_i(x, \langle x, y\rangle)$ *of integer type.*
– *If a reporting message $r^{i\rightarrow j}(x, C)$ or $r^{j\leftarrow i}(x, C)$ is sent, $t_j(j : x, C)$ will be the same as $t_i(i : x, C)$ and $K_j = \max\{K_j, t_i(i : x, C)\}$;*
– *When a new label is added in G_i by an application of CE-, $\sqcap$-, $\forall$- or $\exists$- rules, its timestamp will be the value of the clock K_i;*
– *When a new concept label is added in G_i by an application of the $\sqcup$- rule, the clock K_i is increased by 1 and the label's timestamp is set to be the new value of the clock K_i.*

Definition 7 (Token and Backtracking) *A token $\mathbb{T}$ is passed between local completion graphs. It is originally assigned to the local completion graph of the witness package. Only the local completion graph that has $\mathbb{T}$ can apply the $\sqcup$-rule. A local completion graph whose clock value is no smaller than the clock value of any other local completion graph is a* token target. *We require that* 1) *$\mathbb{T}$ only stays at a token target;* 2) *if G_i has $\mathbb{T}$ and G_i is complete, then $\mathbb{T}$ is transferred to a token target that is not complete.*[3]

A node x in G_i is said to have a t-clash *if both C and $\neg_i C$ are in $\mathcal{L}_i(x)$, for some concept C, and* $t = \max\{t_i(x, C), t_i(x, \neg_i C)\}$.

A distributed completion graph is said to be synchronized *if* 1) *all concept report messages have arrived at targets; and* 2) *all local completion graphs have stopped expansion.*

A pruning operation $\mathsf{Prune}(t)$ *(where t is the timestamp parameter) in G_i does the following:*

– *Removes all concept and role labels in G_i with timestamp $\geq t$.*
– *Removes every node with empty label set and its incoming edges.*
– *Sets K_i to the largest timestamp of concept labels in G_i after the pruning, i.e., $K_i = \max\{t_i(x, C) : x \in V_i, C \in \mathcal{L}_i(x)\}$.*

*If a t-clash occurs in G_i, G_i will broadcast a t-*clash message *to all other local completion graphs, such that the following steps will be executed in order:*

– *Stop all expansions at all local completion graphs, until the distributed completion graph is synchronized.*
– *Perform $\mathsf{Prune}(t)$ in all local completion graphs.*
– *Transfer $\mathbb{T}$ to a token target.*

The pruning operation is necessary to restore all local completion graphs to their status just before the choice which led to the clash, or to the initial status of the local tableau, if no choice at all had ever been made.

Token passing ensures that all local completion graphs are synchronized and that there is only one local completion graph that can apply non-deterministic expansions at any time. Whenever a t-clash is detected,

[3]We do not require a particular token passing protocol (i.e., when and how $\mathbb{T}$ should be transferred) on purpose. We believe it is best that it be determined based on empirical results. In what follows, for the sake of concreteness, we adopt a strategy according to which $\mathbb{T}$ may be transferred immediately after a concept reporting message, if the message target becomes a token target after the message is sent. We emphasize that this is not the only strategy that can be adopted, nor do we claim that it is the most efficient one.

consequences (i.e. nodes and edge labels) dependent on the choice at time t in all local completion graphs will be purged before any other non-deterministic choice can be made. Hence, the handling of a t-clash ensures that different choices in the searching for a clash-free distributed completion graph are always being kept separate. In Example 1, Scenario 1, after G_1 detects the clash, it will send a clash message to G_2 and all local completion graphs will be synchronized. Hence, even if the message $r^{2\to 1}(x', D_2)$ has already been sent before the clash is detected, D_2 will be purged from $\mathcal{L}_1(x)$ during pruning, before G_1 tries other choices. Hence, problems like the one encountered in Phase 2 of Scenario 1 are avoided.

Note that local completion graphs may perform expansions on different reasoning subtasks concurrently. This improves the overall efficiency and scalability of the reasoning process. Further, note that with the introduction of messages, subset blocking in $\mathcal{ALCP}^-_\mathcal{C}$ is *dynamic*: it can be established, broken and re-established. Moreover, the completeness of a local completion graph is also dynamic. A complete local completion graph may become incomplete, i.e., some expansion rules may become applicable, when a new reporting message arrives.

4.4. $\mathcal{ALCP}^-_\mathcal{C}$ Expansion Examples

Example 2 Transitive Subsumption Propagation: Given three packages:

$P_1 : \{1 : A \sqsubseteq 1 : B\}$

$P_2 : \{1 : B \sqsubseteq 2 : C\}$

$P_3 : \{2 : C \sqsubseteq 3 : D\}$

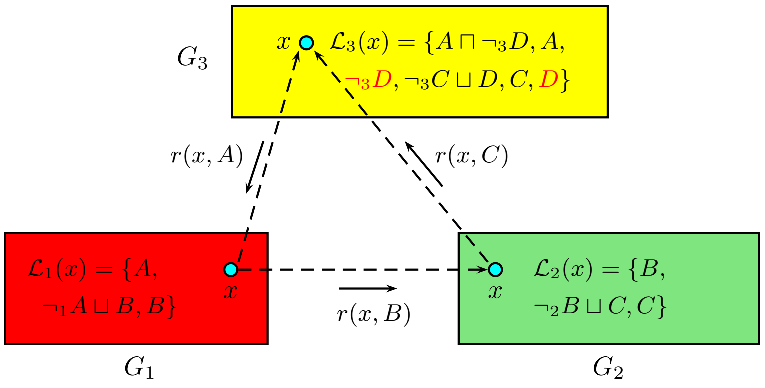

G_3
x
$\mathcal{L}_3(x) = \{A \sqcap \neg_3 D, A, \neg_3 D, \neg_3 C \sqcup D, C, D\}$
$r(x, A)$
$r(x, C)$
$\mathcal{L}_1(x) = \{A, \neg_1 A \sqcup B, B\}$
x
$r(x, B)$
x
$\mathcal{L}_2(x) = \{B, \neg_2 B \sqcup C, C\}$
G_1
G_2

Fig. 3. Transitive Subsumption Propagation in $\mathcal{ALCP}^-_\mathcal{C}$

The query is $1 : A \sqsubseteq 3 : D$ w.r.t. the witness package P_3. The expansion and message exchange between local completion graphs are shown in Figure 3. The following steps result from the execution of the algorithm:

1. G_3 is initialized with the token $\mathbb{T}$ and the node x with $\mathcal{L}_3(x) = \{A \sqcap \neg_3 D\}$; applying $\sqcap$- and CE-rules in G_3, $A, \neg_3 D$ and $\neg_3 C \sqcup D$ are added into $\mathcal{L}_3(x)$. $K_3 = 0$ and all concept labels in G_3 have the timestamp 0.
2. Since A has home package P_1, a message $r^{1\leftarrow 3}(x, A)$ is sent and G_3 transfers $\mathbb{T}$ to G_1. G_1 is initialized with $\mathcal{L}_1(x) = \{A\}$. Applying $\sqcup$- and CE- rules in G_1, $\neg_1 A \sqcup B$ and B are added into $\mathcal{L}_1(x)$. $K_1 = 1$.
3. Since P_2 imports P_1, P_3 imports P_2 and $origin(1 : x) = origin(3 : x)$, we apply the r-rule, creating $2 : x$, with $origin(2 : x) = origin(3 : x)$.
4. Applying the CPush-rule, G_1 sends the message $r^{1\to 2}(x, B)$ and $\mathbb{T}$ to G_2. Applying the $\sqcup$- and CE-rules in G_2, $\neg_2 B \sqcup C$ and C are also added into $\mathcal{L}_2(x)$. $K_2 = 2$.

5. Applying the CPush-rule in G_2, C is added to $\mathcal{L}_3(x)$ and $\mathbb{T}$ is passed to G_3. Applying the $\sqcup$-rule in G_3, $K_3 = 3$ and D is added to $\mathcal{L}_3(x)$. The 3-clash $\{D, \neg_3 D\} \subseteq \mathcal{L}_3(x)$ is detected.
6. As a result, D is now removed from $\mathcal{L}_3(x)$ and clash messages with timestamp 3 are sent to G_1 and G_2, but nothing is removed from G_1 or G_2. K_3 is set to 0. $\mathbb{T}$ is transferred to G_2, since it has the largest clock value.
7. Similarly, all other choices in applying the $\sqcup$-rule lead to clashes. Hence, no clash-free and complete distributed tableau can be found for $A \sqcap \neg_3 D$. Therefore $A \sqsubseteq D$, as witnessed by P_3.

This example shows that P-DL offers a solution to the well-known problem of non-composability of ontology mappings, that is present in DDL [28].

Example 3 Detect Inter-module Unsatisfiability: Given two packages $P_1 : \{1 : B \sqsubseteq 1 : F\}$, $P_2 : \{2 : P \sqsubseteq 1 : B, 2 : P \sqsubseteq \neg_2(1 : F)\}$, test the satisfiability of $2 : P$, as witnessed by P_2. The results shows $2 : P$ is unsatisfiable as witnessed by P_2 (Figure 4):

1. G_2 is initialized with $\mathbb{T}$ and the node x with $\mathcal{L}_2(x) = \{P\}$, $K_2 = 0$. Applying the $\sqcup$-rule and the CE-rule, $\neg_2 P \sqcup B$, $\neg_2 P \sqcup \neg_2 F$, B and $\neg_2 F$ are added into $\mathcal{L}_2(x)$ and $K_2 = 2$.
2. Since B's home package is P_1, we apply the CReport-rule, resulting in the creation of $1 : x$ and $\mathcal{L}_1(x) = \{B\}$. $\mathbb{T}$ is passed to G_1. $K_1 = 2$.
3. Applying the $\sqcup$- and CE- rules in G_1, $\neg_1 B \sqcup F$ and F are added into $\mathcal{L}_1(x)$. $K_1 = 3$.
4. Applying the CPush-rule, the message $r^{1 \to 2}(x, F)$ is sent and F is added into $\mathcal{L}_2(x)$, resulting in a clash.
5. Since $t_2(x, \neg_2 F) = 2$ and $t_2(x, F) = 3$, G_2 has a 3-clash. F is removed from $\mathcal{L}_2(x)$ and a clash message is sent to G_1. $K_2 = 2$.
6. G_1 receives the clash message and removes F from $\mathcal{L}_1(x)$. However, the next choice, i.e., adding $\neg_1 B$, also leads to a clash.
7. Similarly, all other choices in G_1 lead to clashes.

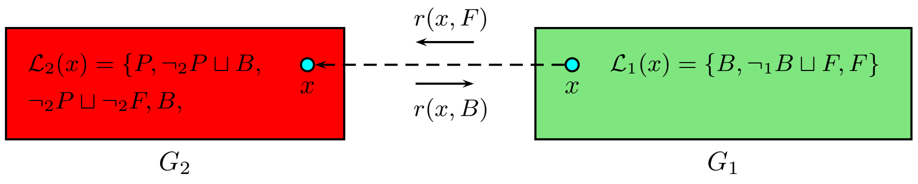

$r(x, F)$
$\mathcal{L}_2(x) = \{P, \neg_2 P \sqcup B,$
$\neg_2 P \sqcup \neg_2 F, B,$
x
x
$\mathcal{L}_1(x) = \{B, \neg_1 B \sqcup F, F\}$
$r(x, B)$
G_2
G_1

Fig. 4. Detect Inter-module Unsatisfiability in $\mathcal{ALCP}_{\mathcal{C}}^{-}$

This example shows that P-DL can also solve the inter-module unsatisfiability problem, that is present in DDL [14].

Example 4 Reasoning from the Local Point of View: Given two packages

$$P_1 : \{1 : A \sqsubseteq 1 : C\}$$

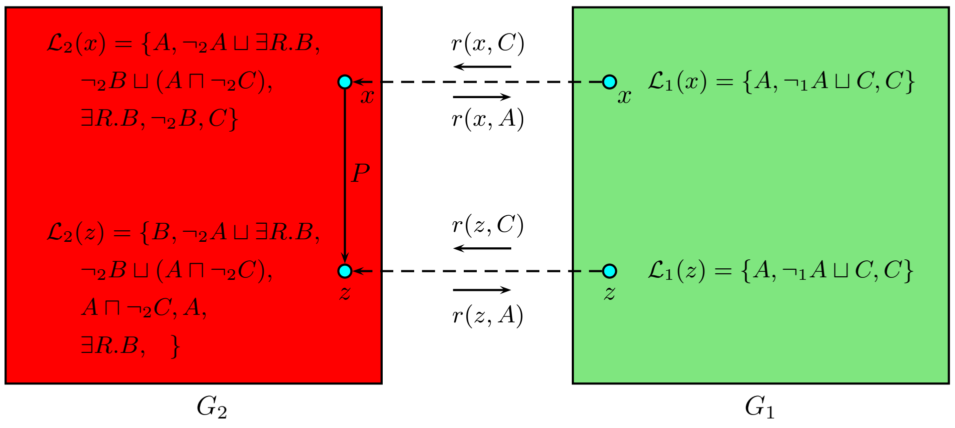
$$P_2 : \{1 : A \sqsubseteq \exists(2 : R).(2 : B), 2 : B \sqsubseteq (1 : A) \sqcap \neg_2(1 : C)\}$$

We need to test the satisfiability of $1 : A$, as witnessed by P_1 and P_2, respectively. It is easy to see that A is satisfiable as witnessed by P_1, but unsatisfiable as witnessed by P_2. Figure 5 shows one possible execution when the witness package is P_2.

- G_2 is initialized with $\mathbb{T}$, $2 : x$ and $\mathcal{L}_2(x) = \{A\}$; applying the CE-, $\sqcap$- and $\sqcup$- rules, $\neg_2 A \sqcup \exists R.B$, $\neg_2 B \sqcup (A \sqcap \neg_2 C)$, $\exists R.B$ and $\neg_2 B$ are added to $\mathcal{L}_2(x)$. $K_2 = 2$.
- A has home package P_1, whence a message $r^{1 \leftarrow 2}(x, A)$ is sent. Consequently, $1 : x$ is created, with $\mathcal{L}_1(x) = \{A\}$. $\mathbb{T}$ is not transferred to G_1 because $K_1 = 0 < K_2$.

- Applying the CE-rule in G_1, $\neg_1 A \sqcup C$ is added to $\mathcal{L}_1(x)$. Now G_1 stops, since it does not have $\mathbb{T}$.
- *In the mean time*, G_2 applies the $\exists$-, CE-, $\sqcap$ and $\sqcup$-rules, creating the node $2 : z$ with $\mathcal{L}_2(z) = \{B, \neg_2 A \sqcup \exists R.B, \neg_2 B \sqcup (A \sqcap \neg_2 C), \exists R.B, A \sqcap \neg_2 C, A, \neg_2 C\}$. The message $r^{1\leftarrow 2}(z, A)$ is sent. Node $1 : z$ is created with $\mathcal{L}_1(z) = \{A\}$. $K_1 = K_2 = 4$. $\mathbb{T}$ is transferred to G_1.
- Applying the CE- and $\sqcup$-rules in G_1, C is added to $\mathcal{L}_1(x)$ and $\mathcal{L}_1(z)$. Two messages $r^{1\to 2}(x, C)$ and $r^{1\to 2}(z, C)$ are sent. $K_1 = K_2 = 6$. $\mathbb{T}$ is transferred back to G_2.
- Since $\{C, \neg_2 C\} \subseteq \mathcal{L}_2(z)$, a 6-clash is detected in G_2. Prune(6) is preformed at G_1 and G_2.
- Similarly, all other choices lead to clashes.

This example shows that reasoning in P-DL always supports the local semantic point of view of the witness package. In this way, the same reasoning problem may have different answers from the points of view of different packages.



$\mathcal{L}_2(x) = \{A, \neg_2 A \sqcup \exists R.B, \neg_2 B \sqcup (A \sqcap \neg_2 C), \exists R.B, \neg_2 B, C\}$
x
P
$\mathcal{L}_2(z) = \{B, \neg_2 A \sqcup \exists R.B, \neg_2 B \sqcup (A \sqcap \neg_2 C), A \sqcap \neg_2 C, A, \exists R.B, \}$
z
$r(x, C)$
$r(x, A)$
$r(z, C)$
$r(z, A)$
$\mathcal{L}_1(x) = \{A, \neg_1 A \sqcup C, C\}$
x
$\mathcal{L}_1(z) = \{A, \neg_1 A \sqcup C, C\}$
z
G_2
G_1

Fig. 5. Reasoning from Local Point of View in $\mathcal{ALCP}_{\mathcal{C}}^{-}$

This example also illustrates the fact that subset blocking in $\mathcal{ALCP}_{\mathcal{C}}^{-}$ only depends on the local ancestorship rather than on the "global" ancestorship. One might have argued that, since $2 : x$ is the local ancestor of $2 : z$, $1 : x$ is a global ancestor of $1 : z$, whence it should have been allowed to block $1 : z$. However, that strategy would have resulted in incorrect blocking: after the message $r^{1\leftarrow 2}(z, A)$ is sent, $\mathcal{L}_1(1 : z) = \{A\}$, which is a subset of $\mathcal{L}_1(1 : x)$. If this had resulted in blocking $1 : z$, adding C to $\mathcal{L}_1(1 : z)$ would have been prevented and this would have led to the erroneous discovery of a consistent distributed completion graph. On the other hand, in the next section, it is shown that, with the presence of cyclic importing, global ancestorship is needed in blocking to ensure termination.

4.5. Soundness, Completeness, Termination and Complexity

In order to show that the algorithm is a decision procedure for concept satisfiability in $\mathcal{ALCP}_{\mathcal{C}}^{-}$, it is necessary to prove that the algorithm terminates, that the models that can be constructed from clash-free and complete distributed completion graphs, generated from the algorithm, are valid with respect to the semantics of the logic (soundness) and that the algorithm always finds a model if one exists (completeness).

Termination and complexity of the algorithm is obtained by proving that there is an upper bound for the total size of all local completion graphs. More specifically, we have the following lemma:

Lemma 3 *Let Σ be an $\mathcal{ALCP}_{\mathcal{C}}^{-}$ ontology and D be an $\mathcal{ALCP}_{\mathcal{C}}^{-}$ concept that is understandable by a witness package P_w in Σ. The $\mathcal{ALCP}_{\mathcal{C}}^{-}$ tableau algorithm runs in worst case non-deterministic $O\Big(2^m \times \prod_{P_j \in P_w^*} 2^{2^{n_j \times \log n_j}}\Big)$ time, where $n_i = |C_{\mathcal{T}_i}|$, $i \neq w$, $n_w = |C_{\mathcal{T}_w}| + |D|$ and $m = |P_w^*|$.*

Proof: We start with a set of observations:

- *For every node that has no local predecessor (called* local top node *henceforth), its local descendants have a tree shape.* This observation follows from the form of the expansion rules.
- *For every local top node $j : x$, $j \neq w$, x must be a preimage of a node in another local completion graph G_i, such that $P_j \mapsto P_i$.* This holds because such an x must be created by a backward concept reporting message triggered by an application of the CReport-rule or of the r-rule. Suppose, for the sake of concreteness, that the message is $r^{j \leftarrow i}(x', C)$, $i \neq j$, where $x' \in V_i$, $origin(x) = origin(x')$ and C is $\top_j$ or a j-concept name. Note that, since x does not exist before the message, C is not added to $\mathcal{L}_i(x')$ by a concept reporting message, whence it must be case that C appears in P_i. Thus, P_i imports P_j and, hence, x is a preimage of x'.
- *For any j,x, all local descendants of $j : x$ in G_j are not preimages of nodes in any other local completion graph.* This holds because a local descendant of $j : x$ is generated only by an application of the $\sqcup$-rule, while a preimage node is created only by an application of the CReport-rule or of the r-rule.

Hence, 1) each local completion graph is a forest; 2) the root of every tree, i.e., a local top node, in a local completion graph, except for the root of G_w, is "copied" from, i.e., it is the preimage of, a node in another local completion graph.

Next we prove that the size of each local completion graph, hence also the total size of the "global completion graph", is limited. For convenience, we define a function $f(x) = 2^{2^{x \times \log x}}$.

First, due to subset blocking, for any local top node in G_j, the depth of its local descendant tree is bounded by $O(2^{n_j})$ and its breadth is bounded by the number of "$\exists$" in $C_{\mathcal{T}_j}$, for $j \neq w$, or in $C_{\mathcal{T}_w} \sqcap D$, for $j = w$, which is smaller than n_j. Thus, the size of the tree is bounded by $O({n_j}^{2^{n_j}}) = O(f(n_j))$.

Since there is only acyclic importing, we can put all packages in P_w^* in an ordered list $\mathfrak{L}$, such that $\mathfrak{L}_1 = P_w$ and each package comes in $\mathfrak{L}$ before all packages in its importing transitive closure, in a way similar to topological sorting in DAG. Let $\#(\mathfrak{L}_j)$ be the subscript of the package at $\mathfrak{L}_j$. Then, we have that the size of $G_{\#(\mathfrak{L}_j)}$ is bounded by:

$$|G_{\#(\mathfrak{L}_1)}| : O(f(n_w))$$

$$|G_{\#(\mathfrak{L}_j)}| : O\Big(\sum_{k<j} |G_{\#(\mathfrak{L}_k)}| \times f(n_{\#(\mathfrak{L}_j)})\Big), \text{ for } j > 1$$

This holds because there is only one local top node in $G_{\#(\mathfrak{L}_1)} = G_w$ (the original node), and, for every $j > 1$ and $p = \#(\mathfrak{L}_j)$, the number of local top nodes in G_p is limited by $\sum\limits_{P_p \mapsto P_q} |G_q|$, i.e., by the total size of the local completion graphs of packages that directly import P_p, since all nodes in P_p must be preimage nodes of nodes in those local completion graphs. In the worst case, $\{P_q | P_p \mapsto P_q\}$ contains all packages that are before j in $\mathfrak{L}$. On the other hand, the size of a tree under a local top node in G_k is limited by $f(n_k)$.

Setting $|G_{\#(\mathfrak{L}_j)}| = t_j$ and $e_j = f(n_{\#(\mathfrak{L}_j)})$, we obtain that t_j is bounded by

$$O\big((t_1 + t_2 + ... + t_{j-1}) \times e_j\big). \qquad (2)$$

Using induction, it will now be shown that t_j is bounded by

$$O\big(2^{j-2} \times e_1 \times ... \times e_j\big), \text{ for } j > 1. \qquad (3)$$

By Equation (2), when $j = 2$, t_2 is bounded by $O(t_1 \times e_2) = O(e_1 \times e_2)$, whence Equation (3) holds. Let $j > 2$. Assuming, as the induction hypothesis, that, for every $1 < k < j$, Equation (3) holds, we have, by Equation (2), that t_j is bounded by

$$\begin{aligned} O\big((t_1 + t_2 + \cdots + t_{j-1}) \times e_j\big) &< O\big((e_1 + 2^0 e_1 e_2 + \cdots + 2^{j-3} e_1 e_2 \cdots e_{j-1}) e_j\big) \\ &< O\big((1 + 2^0 + \cdots + 2^{j-3}) \times e_1 e_2 \cdots e_j\big) \\ &= O\big(2^{j-2} e_1 e_2 \cdots e_j\big) \end{aligned}$$

This finishes the induction step and concludes the proof of Equation (3). Hence, the size of all local completion graphs is bounded by:

$$O\Big(e_1 + \sum_{2\leq j\leq m}\Big(2^{j-2}\prod_{k\leq j} e_j\Big)\Big) \leq O\Big(2^{m-1} \times \prod_{P_j\in P_w^*} f(n_j)\Big)$$
$$< O\Big(2^m \times \prod_{P_j\in P_w^*} 2^{2^{n_j\times\log n_j}}\Big)$$

Q.E.D.

Lemma 4 (Termination and Complexity) *Let Σ be an $\mathcal{ALCP}_\mathcal{C}^-$ ontology and D be an $\mathcal{ALCP}_\mathcal{C}^-$ concept, that is understandable by a witness package P_w in Σ. The $\mathcal{ALCP}_\mathcal{C}^-$ tableau algorithm runs in worst case 2*NExpTime *w.r.t. the size of D and the size of the largest package in P_w^*.*

Proof: Let $n_k = \max\{|C_{\mathcal{T}_i}|\}$ be the size of the largest package in P_w^*, $n_D = |D|$ be the size of D, and $m = |P_w^*|$ be the number of packages in the importing closure of P_w. In general, $m \ll 2^{n_k \log n_k}$. By Lemma 3, it follows that the total size of all local completion graphs is bounded by

$$O\Big(2^m \times 2^{m\times 2^{(n_k+n_D)\times\log(n_k+n_D)}}\Big) < O\Big(2^{2^{(n_k+n_D)^2}}\Big) \qquad \text{Q.E.D.}$$

.

In fact, by the proof of Lemma 3, it follows that the complexity of the $\mathcal{ALCP}_\mathcal{C}^-$ algorithm is bounded by $\text{NTime}\big(\prod_{P_j\in P_w^*} 2^{2^{n_j\log n_j}}\big) = \text{NTime}\Big(2^{\sum_{P_j\in P_w^*} 2^{n_j\log n_j}}\Big)$, where $n_i = |C_{\mathcal{T}_i}|$, for $i \neq w$, and $n_w = |C_{\mathcal{T}_w}| + |D|$. On the other hand, an equivalent reasoning task over the integrated ontology[4] using the $\mathcal{ALC}$ tableau algorithm of [3] will be bounded by $\text{NTime}\big(2^{2^{n_\Sigma\log n_\Sigma}}\big)$, where $n_\Sigma = \sum_{P_j\in P_w^*} n_j$ is the size of the integrated ontology. Since, ordinarily, $m \ll 2^{n_k\log n_k}$,

$$\sum_{P_j\in P_w^*} 2^{n_j\log n_j} \ll 2^{\sum_{P_j\in P_w^*} n_j\log n_j} < 2^{n_\Sigma\log n_\Sigma}$$

The last inequality holds because, for every $x_1 \geq 1, x_2 \geq 1$, we have $x_1\log x_1 + x_2\log x_2 - (x_1 + x_2)\log(x_1+x_2) = x_1(\log x_1 - \log(x_1+x_2)) + x_2(\log x_2 - \log(x_1+x_2)) < 0$. Thus, under the hypotheses that each module in the ontology is moderately sized and that the communication between local reasoners is reliable, it would be reasonable to expect the distributed $\mathcal{ALCP}_\mathcal{C}^-$ reasoning algorithm to terminate significantly faster than its classical counterpart applied on the integrated ontology.

In the following two lemmas, soundness and completeness of the $\mathcal{ALCP}_\mathcal{C}^-$ algorithm are proven.

Lemma 5 (Soundness) *If the $\mathcal{ALCP}_\mathcal{C}^-$ algorithm yields a complete and clash-free distributed completion graph for a concept D w.r.t. a witness package P_w, then D has a tableau w.r.t. P_w.*

Proof: Let $G = \{G_i\}$, with $G_i = (V_i, E_i, \mathcal{L}_i^g)$, be a complete and clash-free distributed completion graph generated by the $\mathcal{ALCP}_\mathcal{C}^-$ algorithm. We will obtain a tableau by "unraveling" blocked nodes and tableau relations. For a directly blocked node x, we denote by $bk(x)$ the node that directly blocks x. Thus, we have $\mathcal{L}_i^g(x) \subseteq \mathcal{L}_i^g(bk(x))$. We can define a tableau $T = \langle\{T_i\}, \{t_{ij}\}_{P_i\in P_j^+}\rangle$, with $T_i = (\mathbf{S}_i, \mathcal{L}_i^t, \mathcal{E}_i)$, for D w.r.t. P_w in the following way:

[4]A reduction to an integrated ontology is described in [5].

$$\mathbf{S}_i = \{x \in V_i |\text{ neither } x \text{ nor any image or preimage node of } x \text{ is blocked}\};$$
$$\mathcal{L}_i^t(x) = \mathcal{L}_i^g(x);$$
$$\mathcal{E}_i(R) = \{\langle x, y\rangle \in V_i \times V_i |\ y \text{ is an R-successor of } x, y \text{ is not blocked}\};$$
$$\cup\{\langle x, bk(y)\rangle \in V_i \times V_i |\ y \text{ is an R-successor of } x, y \text{ is directly blocked}\};$$
$$t_{ij} = \{\langle x, y\rangle \in \mathbf{S}_i \times \mathbf{S}_j |\ origin(x) = origin(y)\}, \text{ for } P_i \in P_j^+.$$

We show that T satisfies all tableau properties.

- Property (A0) holds due to the CE-rule.
- Property (A1) holds since G is clash-free.
- Properties (A2) and (A3) hold because of the $\sqcap$- and $\sqcup$-rules and the fact that G is complete.
- To show Property (A4), suppose that $\forall R.C \in \mathcal{L}_i^t(x) = \mathcal{L}_i^g(x)$ and $\langle x, y\rangle \in \mathcal{E}_i(R)$. Then it must be the case that either 1) $\langle x, y\rangle \in E_i$, $R \in \mathcal{L}_i^g(\langle x, y\rangle)$, whence, according to the $\forall$-rule, $C \in \mathcal{L}_i^g(y) = \mathcal{L}_i^t(y)$; or 2) there exists a y', such that $y = bk(y')$, whence $\mathcal{L}_i^g(y') \subseteq \mathcal{L}_i^g(y)$, $\langle x, y'\rangle \in E_i$, $R \in \mathcal{L}_i^g(\langle x, y'\rangle)$. Thus, according to the $\forall$-rule and the fact that G is complete, $C \in \mathcal{L}_i^g(y') \subseteq \mathcal{L}_i^g(y) = \mathcal{L}_i^t(y)$. Therefore, in both cases, Property (A4) holds.
- Property (A5) may be shown to hold by a proof dual to that of Property (A4).
- Property (B1) holds because, according to the concept reporting message, for any i, j, a node $i : x$ has at most one node of the same origin in G_j.
- For Property (B2) we have: 1) For any $P_i \in P_j^+$, $(x, y) \in t_{ij}$ iff $origin(x) = origin(y)$, whence $(x, y) \in \rho_{ij}^t$ iff $origin(x) = origin(y)$ and, therefore, $\rho_{ij}^t = t_{ij}$. 2) For all $P_i \in P_k^+$ and $P_k \in P_j^+$, $i \neq j$, if there exist $x \in \mathbf{S}_i$ and $x' \in \mathbf{S}_j$, such that $origin(x) = origin(x')$, then, according to the r-rule and the fact that G is complete, there must also exist an $x'' \in V_k$, such that $origin(x'') = origin(x) = origin(x')$. This x'' cannot be blocked, since it must be a local top node, whence $x'' \in \mathbf{S}_k$. Therefore, it follows that $t_{ij} \subseteq t_{kj} \circ t_{ik}$. On the other hand, $t_{kj} \circ t_{ik} \subseteq t_{ij}$ follows by construction.
- Finally, the "only if" direction of Property (B3) holds because of the CPush-rule and the "if" direction because of the CReport-rule.

Q.E.D.

Lemma 6 (Completeness) *If an $\mathcal{ALCP}_\mathcal{C}^-$ concept D has a distributed tableau w.r.t. a witness package P_w, then the $\mathcal{ALCP}_\mathcal{C}^-$ algorithm produces a complete and clash-free distributed completion graph for D w.r.t. P_w.*

Proof: Let $T = \langle\{T_i\}, \{t_{ij}\}_{P_i \in P_j^+}\rangle$, with $T_i = (\mathbf{S}_i, \mathcal{L}_i^t, \mathcal{E}_i)$, be a tableau for D w.r.t. P_w. Following [16], we will use T to guide the application of the non-deterministic $\sqcup$-rule in a way that yields a complete and clash-free distributed completion graph $G = \{G_i\}$, with $G_i = (V_i, E_i, \mathcal{L}_i^g)$.

To construct G, we start with a single node x_0 in the local tableau T_w, with $D \in \mathcal{L}_w^t(x_0)$. Such an x_0 exists, since T is a tableau for D w.r.t. P_w. Let $\pi \subseteq \bigcup_i (V_i \times \mathbf{S}_i)$ be a function that maps all individuals in local completion graphs to individuals in corresponding local tableaux. Initially, we have $V_w = \{x_0\}$, $\mathcal{L}_w^g(x_0) = \{D\}$, $\pi(x_0) = x_0$ and all G_i, $i \neq w$, being empty. Next, we apply $\mathcal{ALCP}_\mathcal{C}^-$ expansion rules to extend G and π, in such a way that the following conditions always (inductively) hold:

$$\begin{cases} \mathcal{L}_i^g(x) \subseteq \mathcal{L}_i^t(\pi(x)) \\ \text{if } R \in \mathcal{L}_i^g(\langle x, y\rangle), \text{ then } \langle \pi(x), \pi(y)\rangle \in \mathcal{E}_i(R) \\ \text{if } origin(i : x) = origin(j : y) \text{ in } G, \text{ then } \langle \pi(x), \pi(y)\rangle \in t_{ij} \text{ in } T, \text{ for } P_i \in P_j^+ \end{cases} \quad (4)$$

- CE-rule: **if** $C_{\mathcal{T}_i} \notin \mathcal{L}_i^g(x)$, **then** $\mathcal{L}_i^g(x) += \{C_{\mathcal{T}_i}\}$. Since, by Property (A0), $C_{\mathcal{T}_i} \in \mathcal{L}_i^t(\pi(x))$, this rule can be applied without violating Conditions (4).

– $\sqcap$-rule: **if** $C_1 \sqcap C_2 \in \mathcal{L}_i^g(x)$, x is not blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}_i^g(x)$, **then** $\mathcal{L}_i^g(x) {+}{=} \{C_1, C_2\}$. Since, by Property (A2) of $\mathcal{ALCP}_\mathcal{C}^-$ tableaux, $C_1 \sqcap C_2 \in \mathcal{L}_i^t(\pi(x))$ implies $C_1 \in \mathcal{L}_i^t(\pi(x))$ and $C_2 \in \mathcal{L}_i^t(\pi(x))$, Conditions (4) are not violated.
– $\sqcup$-rule: **if** $C_1 \sqcup C_2 \in \mathcal{L}_i^g(x)$, x is not blocked, and $\{C_1, C_2\} \cap \mathcal{L}_i^g(x) = \emptyset$, **then** $\mathcal{L}_i^g(x) {+}{=} \{C\}$, for some $C \in \{C_1, C_2\} \cap \mathcal{L}_i^t(\pi(x))$. Such a C must exist because T is a tableau and, hence, satisfies Property (A3), and $C_1 \sqcup C_2 \in \mathcal{L}_i^t(\pi(x))$, by the induction hypothesis. Hence, in this case, Conditions (4) are not violated either.
– $\forall$-rule: **if** $\forall R.C \in \mathcal{L}_i^g(x)$, x is not blocked, and there is a local R-successor y of x in G_i with $C \notin \mathcal{L}_i^g(y)$, **then** $\mathcal{L}_i^g(y) {+}{=} \{C\}$. By the induction hypothesis, $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}_i(R)$ and $\forall R.C \in \mathcal{L}_i^t(\pi(x))$, whence, by Property (A4), $C \in \mathcal{L}_i^t(\pi(y))$. Thus, Conditions (4) are not violated.
– $\exists$-rule: **if** $\exists R.C \in \mathcal{L}_i^g(x)$, x is not blocked, and x has no local R-successor y of x in G_i, with $C \in \mathcal{L}_i^g(y)$, **then** 1) create a new node y, with $orgin(y) = y$, $\mathcal{L}_i^g(\langle x, y \rangle) = \{R\}$ and $\mathcal{L}_i^g(y) = \{C\}$; 2) let $\pi(y) = y'$, where $y' \in \mathbf{S}_i$, $\langle \pi(x), y' \rangle \in \mathcal{E}_i(R)$ and $C \in \mathcal{L}_i^t(y')$. Such a y' must exist because T is a tableau and, hence, it satisfies Property (A5) and, by the induction hypothesis, $\exists R.C \in \mathcal{L}_i^t(\pi(x))$. Therefore, Conditions (4) are not violated.
– r-rule: **if** $origin(i : x) = origin(j : x')$, there exists k such that $P_i \in P_k^+$, $P_k \in P_j^+$ and there is no $k : x''$ with $origin(j : x') = origin(k : x'')$, **then** 1) transmit $r^{k \leftarrow j}(x', \top_k)$. This will create $k : x''$, such that $origin(k : x'') = origin(j : x') = origin(i : x)$; 2) let $\pi(x'') = z$, where $z \in \mathbf{S}_k$, $\langle \pi(x'), z \rangle \in t_{ik}$ and $\langle z, \pi(x') \rangle \in t_{kj}$; such a z must exist because, by the induction hypothesis, $\langle \pi(x), \pi(x') \rangle \in t_{ij}$ and, by the tableau Property (B2), $t_{ij} = t_{kj} \circ t_{ik}$. After this operation $\mathcal{L}_k^g(x'') = \emptyset$. Therefore, Conditions (4) are not violated.
– CPush-rule: **if** $C \in \mathcal{L}_i^g(x)$, where C is an i-concept name, $P_i \xrightarrow{C} P_j$, x is not blocked and there exists an $x' = x^{i \to j} \in V_j$, such that $C \notin \mathcal{L}_j^g(x')$, **then** transmit $r^{i \to j}(x, C)$. This will set $\mathcal{L}_j^g(x') {+}{=} \{C\}$. By the induction hypothesis, $\langle \pi(x), \pi(x') \rangle \in t_{ij}$, $C \in \mathcal{L}_i^t(\pi(x))$, whence, by Property (B3), $C \in \mathcal{L}_j^t(\pi(x'))$. Hence, Conditions (4) are not violated.
– CReport-rule: **if** $C \in \mathcal{L}_i^g(x)$, where C is $\top_j$ or a j-concept name, x is not blocked and there is no $x' = x^{j \leftarrow i} \in V_j$ such that $C \in \mathcal{L}_j^g(x')$, **then** 1) transmit $r^{j \leftarrow i}(x, C)$. This will create $x' = x^{j \leftarrow i}$, with $origin(x') = origin(x)$, if such an x' had not already been created, and set $\mathcal{L}_j^g(x') {+}{=} \{C\}$; 2) let $\pi(x') = x''$, if $\pi(x')$ has not yet been given, where $x'' \in \mathbf{S}_j$, $\langle x'', \pi(x) \rangle \in t_{ji}$ and $C \in \mathcal{L}_i^t(x'')$. Such a, x'' must exist because, by the induction hypothesis, $C \in \mathcal{L}_i^t(\pi(x))$ and T satisfies tableau Property (B3). Therefore, Conditions (4) are not violated in this case either.

G must be clash-free, since, if there existed i, x, C, such that $\{C, \neg_i C\} \subseteq \mathcal{L}_i^g(x)$, then, by Conditions (4), $\{C, \neg_i C\} \subseteq \mathcal{L}_i^t(\pi(x))$, which would contradict tableau Property (A1) for T. Hence, whenever an expansion rule is applicable to G, it can be applied in such a way that maintains Conditions (4). By the Termination Lemma, any sequence of rule applications must terminate. Hence, we will obtain a complete and clash-free completion graph G for D from T. Q.E.D.

By Lemmas 3-6, we obtain the following theorem, which is the main theorem of the paper.

Theorem 1 *Let Σ be an $\mathcal{ALCP}_\mathcal{C}^-$ ontology and D be an $\mathcal{ALCP}_\mathcal{C}^-$ concept, that is understandable by a witness package P_w in Σ. The $\mathcal{ALCP}_\mathcal{C}^-$ tableau algorithm is a sound, complete, and terminating decision procedure for satisfiability of D as witnessed by P_w. This decision procedure is in 2*NExpTime *w.r.t. the size of D and the size of the largest package in P_w^*.*

5. A Reasoning Algorithm for $\mathcal{ALCP}_\mathcal{C}$

5.1. Extended Subset Blocking

The reasoning algorithm for $\mathcal{ALCP}_\mathcal{C}^-$ may fail if we relax the acyclicity assumption, i.e., when applied to the P-DL $\mathcal{ALCP}_\mathcal{C}$, as illustrated by the following example.

Example 5 : Suppose we have two packages that mutually import one another:

$$P_1 : \{\top_1 \sqsubseteq \exists(1:R).(2:D)\}$$

$$P_2 : \{\top_2 \sqsubseteq \exists(2:P).(1:C)\}$$

The reasoning task is to check the consistency of the ontology as witnessed by P_1. If we employ the decision procedure for $\mathcal{ALCP}_{\mathcal{C}}^{-}$, the algorithm will not terminate, as shown in Figure 6. Since there is mutual importing, each of the local completion graphs G_1 and G_2 can send reporting messages and create new nodes in the other. Subset blocking, as given in the previous section, cannot prevent local completion graphs from exchanging messages in a cyclic fashion, which leads to non-termination.

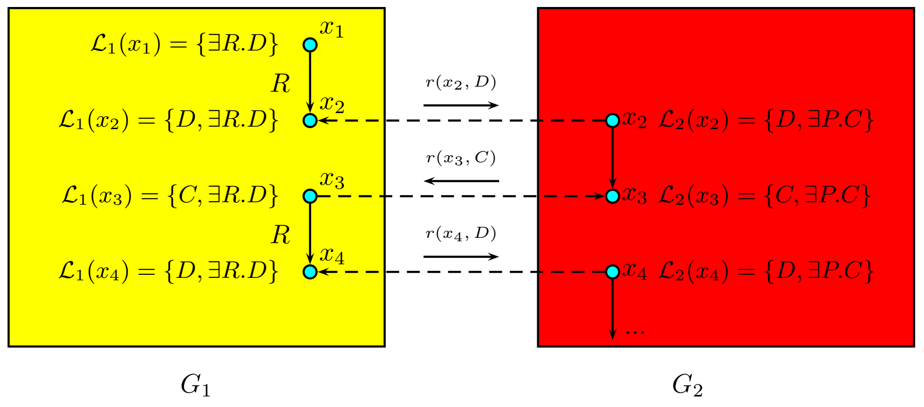

$\mathcal{L}_1(x_1) = \{\exists R.D\}$
x_1
R
$\mathcal{L}_1(x_2) = \{D, \exists R.D\}$
x_2
$\mathcal{L}_1(x_3) = \{C, \exists R.D\}$
x_3
R
$\mathcal{L}_1(x_4) = \{D, \exists R.D\}$
x_4
$r(x_2, D)$
$r(x_3, C)$
$r(x_4, D)$
x_2 $\mathcal{L}_2(x_2) = \{D, \exists P.C\}$
x_3 $\mathcal{L}_2(x_3) = \{C, \exists P.C\}$
x_4 $\mathcal{L}_2(x_4) = \{D, \exists P.C\}$
...
G_1
G_2

Fig. 6. Non-termination Caused by Cyclic Importing

Extending the $\mathcal{ALCP}_{\mathcal{C}}^{-}$ algorithm to handle cyclic importing relations requires the detection and prevention of cyclic message exchanges as well as of cyclic local expansions. In fact, termination of the $\mathcal{ALCP}_{\mathcal{C}}^{-}$ algorithm is due to the fact that a local completion graph G_i can cause the creation of a local top node, i.e., a node without a local predecessor, in another local completion graph G_j by an application of the CReport- or the r-rule if and only if the package P_i directly or indirectly imports package P_j. With the presence of cyclic importing, the creation of infinitely many local top nodes in a local completion graph may not be avoided. Thus, subset blocking may fail, since it can only ensure that the number of local descendant nodes of a local top node is limited.

Termination with cyclic importing can be regained if we can ensure that, in any local completion graph, the number of local top nodes as well as the number of local descendant nodes of each local top node are limited. This goal will be realized by an appropriate extension of subset blocking.

Definition 8 (Extended Subset Blocking) *A node x is a* global ancestor *of another node y, which may be in a different local graph, if $origin(x) \neq origin(y)$ and there is a path from x to y on the graph $G' = \bigcup_i G_i \cup \{(u,v)|origin(u) = origin(v)\}$, i.e., a path using both local edges and edges in the symmetric closure of graph relations. For any i and any x, y, such that $x \in V_i$, x is a* least global ancestor *of y in G_i if x is a global ancestor of y and there is no other $z \in V_i$ such that z is a global ancestor of y and x is a global ancestor of z.*

For a distributed completion graph of an $\mathcal{ALCP}_{\mathcal{C}}$ ontology, a node x in G_i is directly blocked *by a y in G_i if 1) y is a global ancestor of x and $\mathcal{L}_i(x) \subseteq \mathcal{L}_i(y)$, and 2) for every $j \neq i$, if there are x', y' in G_j such that $origin(x') = origin(x)$ and $origin(y') = origin(y)$, then $\mathcal{L}_i(x') \subseteq \mathcal{L}_i(y')$. Node x is* indirectly blocked *by a node y if one of x's global ancestors is directly blocked by y. Finally, node x is* blocked *by y if it is directly or indirectly blocked by y.*

Explanation: With extended subset blocking, a node x, including a local top node, can be blocked by one of its global ancestors y, if every local "copy" of x contains no more information than the corresponding local "copy" of y. In this case, expansions at x are not needed, since corresponding expansions must have been preformed at y. Hence, the creation of infinitely many local top nodes as well as of infinitely large local trees under each local top node is avoided. A more detailed analysis of this point will be presented in the termination proof of the algorithm.

For instance, in Example 5, $1 : x_4$ will be blocked by $1 : x_2$. As a result, the backward concept reporting messages $r^{2 \leftarrow 1}(x_4, D)$ will not be sent (a similar message $r^{2 \leftarrow 1}(x_2, D)$ has been sent before) and the reasoning process will terminate. On the other hand, when applying extended subset blocking in Example 4, node $1 : z$ will not be blocked by node $1 : x$, whence the necessary forward reporting message $r^{1 \rightarrow 2}(z, C)$ will not be undesirably blocked.

Labeling for Global Ancestorship: Since each local reasoner is autonomously maintained, the topology of a local completion graph may not be available to other reasoners. To keep track of the global ancestor relationship in the distributed setting, we may use a labeling schema for dynamic tree representation, since, by merging nodes of the same origin, all local completion graphs can be combined into a tree. The basic intuition is to assign localized, informative labels to each node in the distributed graph which will contain global topology information of the graph. Each node of the same origin will be assigned the same label. In this way, testing global ancestorship can be reduced to comparison of the labels of different nodes. Several labeling schemas for static and/or dynamic trees have been recently proposed [8,7,17,9]. The adoption of a particular labeling schema is to be decided during the implementation of the algorithm. It will partially depend on the communication protocol on which the algorithm will be based to achieve best performance.

5.2. Correctness and Complexity

The reasoning algorithm for $\mathcal{ALCP_C}$ is a modified version of the $\mathcal{ALCP}_\mathcal{C}^-$ algorithm, resulting by replacing subset blocking by extended subset blocking and by adding the labeling technique of the various nodes, as described previously.

Theorem 2 *Let Σ be an $\mathcal{ALCP_C}$ ontology and D an $\mathcal{ALCP_C}$ concept, that is understandable by a witness package P_w in Σ. The $\mathcal{ALCP_C}$ tableau algorithm is a sound, complete and terminating decision procedure for satisfiability of D as witnessed by P_w. This decision procedure is in 2*NExpTime *w.r.t. the size of D and the total size of packages in P_w^*.*

Proof: Only a sketch of the proof will be provided. Proofs of the soundness and completeness are similar to the proofs of Lemmas 5 and 6, respectively. So we concentrate on termination and complexity.

Termination will be proven by showing that the "combined" completion graph G', resulting from the various local completion graphs by merging all nodes of the same origin into one node, is finite. For a local completion graph G_i, let $n_i = |C_{\mathcal{T}_i}|$, for $i \neq w$, $n_w = |C_{\mathcal{T}_w}| + |D|$, $n_\Sigma = \sum_i n_i$ and $m = |P_w^*|$. Let x_0 be the initial node of G_w.

For every node in G_i, its out-degree is at most n_i, whence, for every node in G', its out-degree is bounded by n_Σ. Similarly, the size of the concept label set of each node in G_i is bounded by n_Σ. The depth of G' is a most 2^{n_Σ} due to the extended subset blocking. Hence, the total number of nodes in G' is bounded by

$$O\Big((n_\Sigma)^{2^{n_\Sigma}}\Big) = O\Big(2^{2^{n_\Sigma} \log n_\Sigma}\Big).$$

Therefore, the total number of nodes in the distributed completion graph is bounded by

$$O\Big(m \times 2^{2^{n_\Sigma} \log n_\Sigma}\Big). \qquad \text{Q.E.D.}$$

With the presence of cyclic importing, the *worst-case* time complexity of the $\mathcal{ALCP}_\mathcal{C}$ algorithm is bounded by the total size of all packages, while that of the $\mathcal{ALCP}_\mathcal{C}^-$ algorithm is only bounded by the size of the largest package involved in the reasoning task. This result indicates that avoiding cyclic importing between ontology modules will significantly improve reasoning performance.

The $\mathcal{ALCP}_\mathcal{C}$ algorithm has the same *worst-case* time complexity with the $\mathcal{ALC}$ tableau algorithm applied on the combined ontology from all modules. However, the analysis in Theorem 2 does not take into account the gain resulting from local reasoners concurrently exploring different reasoning sub-tasks. We believe that, with the proper design of communication protocols between local reasoners, the distributed $\mathcal{ALCP}_\mathcal{C}$ tableau algorithm has the potential of processing a reasoning task more efficiently than would a centralized reasoner.

6. Related Work

Partition-based Logics and Somewhere: Several authors have recently investigated distributed reasoning algorithms for modular ontologies. Partition-based Logics [2] provides an approach to automatically decompose propositional and first-order logic (FOL) into *partitions* and an algorithm for reasoning with those partitions using message passing. The Somewhere peer-to-peer data management system [1] provides a distributed query answering algorithm for a "propositional" fragment of Description Logics. On the other hand, our focus is on developing a sound and complete distributed reasoning for distributed Package-based Description Logics.

DDL: In [20,19] a tableau-based reasoning algorithm for Distributed Description Logics (DDL) with acyclic bridge rules between concepts is developed. The algorithm divides a reasoning problem w.r.t. a DDL TBox into several local reasoning problems answered by local modules. The basic idea behind this algorithm is to infer concept subsumption in one module from subsumptions in another module and inter-module *bridge rules*. For example, consider two ontology modules i and j, in which the concepts A, B and G, H respectively, are defined, together with the bridge rules $i : A \xrightarrow{\sqsupseteq} j : G$, $i : B \xrightarrow{\sqsubseteq} j : H$. If module i entails $A \sqsubseteq B$, then it is possible for module j to infer $G \sqsubseteq H$. Thus, an ontology module may submit a subsumption query, or an unsatisfiability query, to another module to complete a local reasoning task.

The algorithm is implemented in the DRAGO system [19,25], which allows multiple reasoners to communicate with one another via TCP connections to perform a reasoning task. This approach is extended in [21] to cover the distributed version of retrieval in DDL, in [19] to cover reasoning with cyclic bridge rules using fixed-point semantics, and in [11] to cover reasoning with bridge rules between roles.

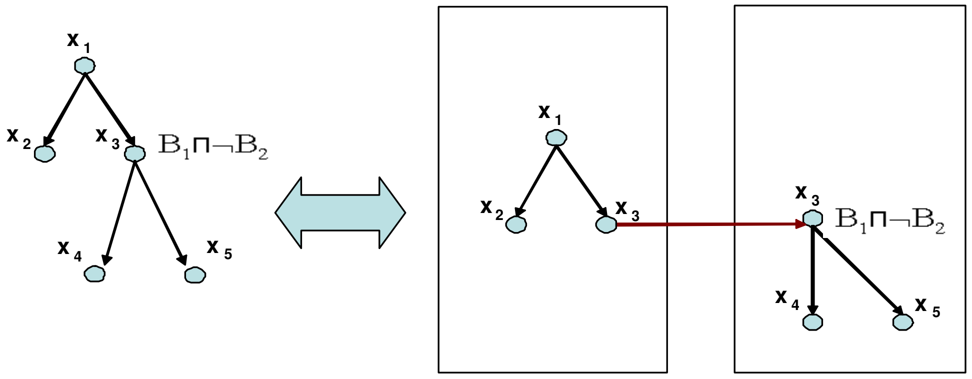


Fig. 7. Completion Graph in the DDL Tableau Algorithm

The DDL reasoning algorithm builds a virtual tree-shaped global completion graph (for the integrated ontology from all modules) by constructing multiple trees in local reasoners using local knowledge. This is in accord with the basic intuition of P-DL reasoning algorithms. However, the two approaches differ on how to decompose the virtual global completion graph. In the DDL approach (Figure 7), each local reasoner builds a “branch” of the global tree. On the other hand, in the P-DL approach (Figure 8), due to the fact that the concept languages of modules are not disjoint, a local reasoner builds a “projection” of the global tree and, as a result, some nodes may be “shared” by multiple local trees.

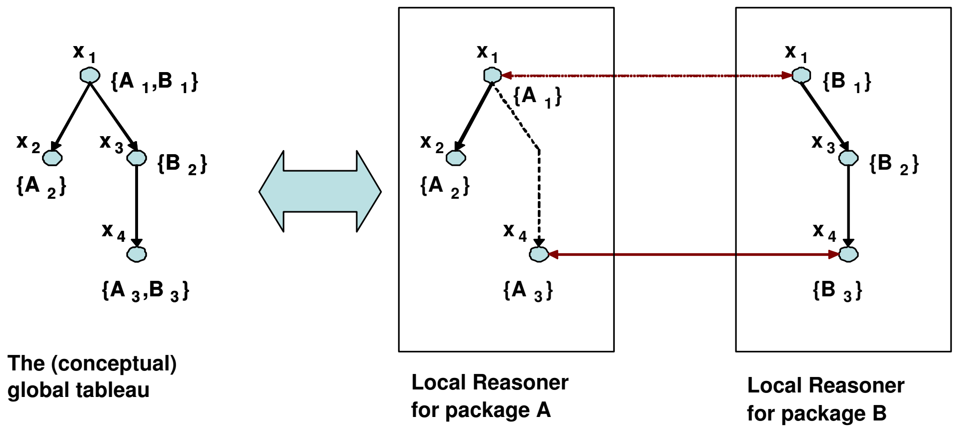


Fig. 8. Completion Graph in P-DL Tableau Algorithms

The DDL reasoning algorithm is limited in several ways. The algorithm does not support inference of bridge rules. For instance, if $L_1 = \{1 : A \sqsubseteq 1 : B\}, L_2 = \emptyset, \mathcal{B}_{12} = \{1 : B \xrightarrow{\sqsubseteq} 2 : C\}$, the algorithm cannot infer that $1 : A \xrightarrow{\sqsubseteq} 2 : C$.[5] On the contrary, P-DL solves these problems since 1) P-DL semantics ensures transitive reusability of ontology modules and 2) semantic importing in P-DL allows “inter-module” semantic relations, like DDL bridge rules, and “intra-module” semantic relations, like local concept subsumptions, to be treated in a uniform way. As a result, the P-DL reasoning algorithms, described in this paper, can handle both scenarios successfully.

$\mathcal{E}$-Connections: In [14,13,15] a tableau-based reasoning procedure for $\mathcal{E}$-Connections is presented. It generates a set of local completion graphs (typically trees) linked by $\mathcal{E}$-connection instances (cross-module role instances), as illustrated in Figure 9. Each local completion graph is associated with a color and the reasoning process is performed on the *combined completion forest* resulting by combining all those local completion graphs.

The $\mathcal{E}$-connections algorithm adopts the approach of “coloring”, but not of physically separating, local completion graphs. Hence, it is assumed in the algorithm that a local completion graph can freely access information of other local completion graphs, e.g., the node successor relationship and neighborhood, node and edge labels and blocking conditions. Therefore, no message passing or any other forms of communication between local completion graphs are required, nor are specially designed distributed backtracking strategies provided. Thus, the implementation of the algorithm in the Pellet reasoner [23] utilizes a *single* reasoner to preform reasoning tasks for an $\mathcal{E}$-connected ontology.

However, such an approach implicitly assumes the availability of *global knowledge* in all ontology modules for the reasoning in a modular ontology to be possible. This counteracts many of the benefits of having a modular ontology; in particular, scalability and the preservation of module privacy. For example, as is implied by the CE-rule of its algorithm, the Pellet implementation requires that all ontology modules be loaded into the *same* memory space. Thus, this implementation implicitly requires the integration of all ontology modules. By contrast, P-DL, and also DDL, reasoning algorithms are genuinely distributed reasoning algorithms, not requiring, either implicitly or explicitly, the integration of ontology modules.

[5]Reasoning about properties of bridge rules has been addressed in [24]. However, [24] does not provide a decision procedure for the inference of bridge rules.

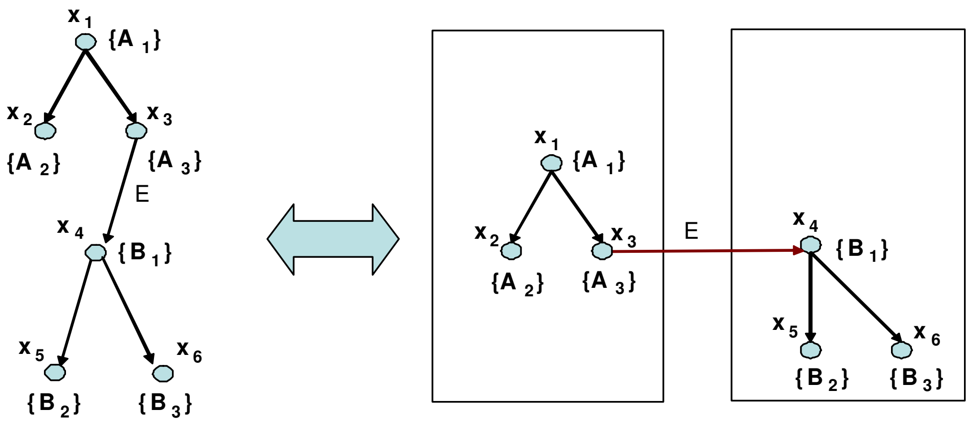


Fig. 9. Completion Graph in $\mathcal{E}$-Connections Tableau Algorithms

7. Conclusion

We have presented a distributed tableau-based reasoning algorithm for the package-based extension of the DL language $\mathcal{ALCP_C}$. The proposed algorithm offers a practical approach that

1. avoids the need for loading the entire contents of all ontology modules into a central location, by circumventing the integration of modules into a single ontology;
2. allows arbitrary reuse of knowledge among the various ontology modules, such as the presence of mutual or cyclic importing among packages, by using a message-based inter-reasoner communication strategy;
3. tackles a broader range of reasoning tasks, based on the P-DL formalism.

Work in progress is aimed at:

– extending the proposed reasoning algorithm to work with more expressive DLs such as $\mathcal{SHIQP}$, i.e., package-based $\mathcal{SHIQ}$ with concept and role importing;
– designing communication protocols between local reasoners, including ones based on handshaking and acknowledgement, clash reporting and backtracking, token passing and labeling for the global ancestor relationship;
– evaluating the performance of the implementation of the proposed algorithm based on several practical application scenarios.

Acknowledgement: This research was supported in part by grants from the US National Science Foundation (IIS-0639230).

References

[1] Philippe Adjiman, Philippe Chatalic, Francois Goasdou, Marie-Christine Rousset, and Laurent Simon. Distributed Reasoning in a Peer-to-Peer Setting: Application to the Semantic Web . *Journal of Artificial Intelligence Research*, 25:269,314, 2006.

[2] Eyal Amir and Sheila A. McIlraith. Partition-based logical reasoning. In *KR*, pages 389–400, 2000.

[3] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, 2001.

[4] Jie Bao, Doina Caragea, and Vasant Honavar. A tableau-based federated reasoning algorithm for modular ontologies. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 404–410. IEEE Press, 2006.

[5] Jie Bao, Giora Slutzki, and Vasant Honavar. A semantic importing approach to knowledge reuse from multiple ontologies. In *AAAI*, pages 1304–1309, 2007.

[6] Matteo Bonifacio, Paolo Bouquet, Paolo Busetta, Alberto Danieli, Antonia Donà, Gianluca Mameli, and Michele Nori. Keex: A peer-to-peer solution for distributed knowledge management. In *P2PKM*, 2004.

[7] Yi Chen, George A. Mihaila, Rajesh Bordawekar, and Sriram Padmanabhan. L-tree: A dynamic labeling structure for ordered xml data. In *EDBT Workshops*, pages 209–218, 2004.

[8] Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, Michel Scholl, and Sotirios Tourtounis. Optimizing taxonomic semantic web queries using labeling schemes. *J. Web Sem.*, 1(2):207–228, 2004.

[9] Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, and David Peleg. Labeling schemes for tree representation. In *IWDC*, pages 13–24, 2005.

[10] Tom Gardiner, Ian Horrocks, and Dmitry Tsarkov. Automated benchmarking of description logic reasoners. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*, volume 189, 2006.

[11] Chiara Ghidini and Luciano Serafini. Mapping properties of heterogeneous ontologies. In *1st International Workshop on Modular Ontologies (WoMo 2006), co-located with ISWC*, 2006.

[12] Erich Grädel. Why are modal logics so robustly decidable? In *Current Trends in Theoretical Computer Science*, pages 393–408. 2001.

[13] Bernardo Cuenca Grau. *Combination and Integration of Ontologies on the Semantic Web*. PhD thesis, Dpto. de Informatica, Universitat de Valencia, Spain, 2005.

[14] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Working with multiple ontologies on the semantic web. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 620–634. Springer, 2004.

[15] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining owl ontologies using epsilon-connections. *J. Web Sem.*, 4(1):40–59, 2006.

[16] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *LPAR*, pages 161–180, 1999.

[17] Amos Korman, David Peleg, and Yoav Rodeh. Labeling schemes for dynamic tree networks. *Theory Comput. Syst.*, 37(1):49–75, 2004.

[18] G. Schreiber and M. Dean. Owl web ontology language reference. http://www.w3.org/TR/2004/REC-owl-ref-20040210/, February 2004.

[19] Luciano Serafini, Alexander Borgida, and Andrei Tamilin. Aspects of distributed and modular ontology reasoning. In *IJCAI*, pages 570–575, 2005.

[20] Luciano Serafini and Andrei Tamilin. Local tableaux for reasoning in distributed description logics. In *Description Logics Workshop 2004, CEUR-WS Vol 104*, 2004.

[21] Luciano Serafini and Andrei Tamilin. Distributed instance retrieval in heterogeneous ontologies. In *Proceedings of SWAP 2005, CEUR Workshop Vol 166*, 2005.

[22] Evren Sirin and Bijan Parsia. Pellet: An OWL DL Reasoner. In *Description Logics Workshop*, 2004.

[23] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *J. Web Sem.*, 5(2), 2007.

[24] Heiner Stuckenschmidt, Luciano Serafini, and Holger Wache. Reasoning about ontology mappings. In *ECAI 2006 Workshop on Context Representation and Reasoning (CRR)*, 2006.

[25] Andrei Tamilin. *Distributed Ontological Reasoning: Theory, Algorithms, And Applications.* Phd dissertation, University of Trento, February 2007.

[26] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Description Logics*, 2004. FaCT++.

[27] Moshe Y. Vardi. Why is modal logic so robustly decidable? In *Descriptive Complexity and Finite Models*, pages 149–184, 1996.

[28] Antoine Zimmermann and Jérôme Euzenat. Three semantics for distributed systems and their relations with alignment composition. In *International Semantic Web Conference*, pages 16–29, 2006.