A Distributed Tableau Algorithm for Package-based Description Logics

Jie Bao and **Doina Caragea** and **Vasant G Honavar**[1]

Abstract. Package-based Description Logics (P-DL) is a novel formalism for modular ontologies. In P-DL, an ontology is composed of a collection of modules called *packages*. A package can partially reuse knowledge in other packages by a selective importing mechanism. This paper investigates a sound and complete tableau-based reasoning algorithm for a P-DL language $\mathcal{ALCP}_\mathcal{C}$, which extends $\mathcal{ALC}$ with acyclic concept importing between packages. The algorithm allows the reasoning process to be distributed based on local reasoning services offered by each module. The algorithm allows the reasoning process to be distributed based on local reasoning services offered by each module. Local tableaux associated with the ontology modules while physically separate, may conceptually overlap by communicating with each other via a set of messages. Our investigation shows the algorithm can avoid several semantic difficulties associated with existing approaches, such as transitive subsumption propagation and inter-module unsatisfiability detection.

1 INTRODUCTION

Because of the distributed and context-specific nature of web ontologies, there is growing interest in modular ontology languages such as Distributed Description Logics (DDL) [4], $\mathcal{E}$-connections [8, 6] and Package-extended Description Logics (P-DL) [3]. Consequently, sound and complete distributed reasoning algorithms are urgently needed to support large scale applications of ontologies. Such algorithms ideally should avoid the need to combine the ontology modules into a centralized knowledge base. Distributing the reasoning effort across the modules helps respect the autonomy of each autonomous ontology module.

DRAGO [13] (for DDL) and Pellet [14] (for $\mathcal{E}$-Connections) offer examples of efforts aimed at developing distributed reasoning algorithms.

DDL connects concepts in different modules with *bridge rules* such as onto ($\xrightarrow{\sqsupseteq}$) and into ($\xrightarrow{\sqsubseteq}$) to simulate cross-module concept subsumptions. Serafini and Tamilin (2004) [11] have presented a distributed tableau algorithm for DDL. Their algorithm divides the satisfiability problem w.r.t. a DDL TBox into several local satisfiability problems w.r.t. local TBoxes in ontology modules. The basic idea behind this algorithm is to infer subsumption in one module from subsumptions in another module and proper inter-module bridge rules. For example, if there are bridge rules $i:A \xrightarrow{\sqsupseteq} j:G, i:B \xrightarrow{\sqsubseteq} j:H$ and module i entails $A \sqsubseteq B$, then $G \sqsubseteq H$ can be inferred in module j. Further results based on this approach are reported in [10] with a new fixed-point semantics of bridge rules and a caching mechanism to store remote subsumptions, and in [12] for DDL ABox reasoning.

This algorithm is compatible with the state-of-the-art tableau-based DL reasoners and lends itself to a simple implementation as demonstrated by the DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies) [13] system. However, the algorithm has several limitations: It does not support inference of inter-module subsumption (e.g. $i:A \xrightarrow{\sqsubseteq} j:B$) or transitive subsumption propagation among multiple modules (e.g. given $i:A \xrightarrow{\sqsubseteq} j:B$ and $j:B \xrightarrow{\sqsubseteq} k:C$, infer $i:A \xrightarrow{\sqsubseteq} k:C$). Without principled ways to ensure semantic soundness of bridge rules, the algorithm may fail due to incomplete modelling. For example, If module 1 entails $\top \sqsubseteq Car$, module 2 entails $UsefulThing \sqsubseteq \neg UselessThing$, and there are bridge rules $1:Car \xrightarrow{\sqsubseteq} 2:UsefulThing$ and $1:Car \xrightarrow{\sqsubseteq} 2:UselessThing$, the algorithm can not detect the inconsistency. For another example, $1:Bird \xrightarrow{\sqsupseteq} 2:Penguin$ and $1:\neg Fly \xrightarrow{\sqsupseteq} 2:Penguin$ do not render $2:Penguin$ unsatisfiable even if module 1 entails $Bird \sqsubseteq Fly$. In general, the algorithm may not detect inter-module unsatisfiability propagation without one-to-one domain relations [10, 2], nor detect inconsistencies due to improper bridge rules.

$\mathcal{E}$-connections between DLs [7, 6] restrict the local domains of the $\mathcal{E}$-connected ontology modules to be disjoint. Roles are divided into disjoint sets of *local roles* (connecting concepts in one module) and *links* (connecting inter-module concepts). For example, two modules about people (L_1) and pets (L_2) can be connected by a link $owns$, and L_1 can use such a link to build local concepts, e.g. $1:DogOwner \sqsubseteq \exists owns.(2:Dog)$.

$\mathcal{E}$-Connections allow straightforward implementation of reasoning based on existing tableau OWL reasoners, e.g. Pellet. The tableau-based reasoning procedure as presented in [6, 5] is an extension to existing DL tableau algorithm. Instead of having a single tableau, the algorithm will generate a set of tableaux (trees) linked by $\mathcal{E}$-connection instances (cross-module role instances).

However, reasoning within the $\mathcal{E}$-Connections framework has a significant limitation, namely, the lack of inter-module subsumptions since local domains of all modules are strictly disjoint. In general, $\mathcal{E}$-Connections do not support transitive knowledge propagation among ontology modules. Current prototype implementation of the $\mathcal{E}$-Connections reasoner (as a part of Pellet), which is motivated by the "combined tableau" idea [6, 5], only "colors" but does not separate each local tableau. Therefore, a reasoning process will result in one combined ABox in a single memory space, thus equivalently forcing TBoxes of all involved modules to be locally loaded. Such a strategy defeats many benefits of modular ontologies (e.g., scaleability and local module autonomy).

[1] Artificial Intelligence Research Laboratory, Department of Computer Science, Iowa State University, Ames, IA 50011-1040, USA. {baojie, dcaragea, honavar}@cs.iastate.edu

In summary, DDL and E-Connections are motivated by, and hence are responsive to, different application scenarios. Their expressivity and reasoning power is complementary in several ways. However, both of them are also limited in several ways. Due to the strong local domain disjointness assumption adopted in those approaches, the distributed reasoning processes with such approaches may encounter semantic difficulties as shown above. They also fail to provide solutions for some critical distributed reasoning tasks, such as transitive concept subsumption across multiple modules.

P-DL [3], by relaxing the local domain disjointness assumption, allows discovery of a distributed model for a set of ontology modules that is identical to that obtainable by combining the ontology modules into a centralized ontology, a property we refer to as *exactness* of distributed reasoning relative to its centralized counterpart. This paper investigates a sound and complete tableau-based reasoning algorithm for a P-DL language $\mathcal{ALCP}_{\mathcal{C}}$, which extends $\mathcal{ALC}$ with acyclic concept importing between packages. The algorithm allows the reasoning process to be distributed based on local reasoning services offered by each module. Local tableaux associated with the ontology modules while physically separate, may conceptually overlap by communicating with each other via a set of messages. Our preliminary investigation shows that the proposed algorithm can solve many known reasoning difficulties in existing approaches. Complexity study shows the algorithm for the package-based $\mathcal{ALC}$ language has the same time complexity as that of a canonical $\mathcal{ALC}$ reasoning algorithm.

2 PACKAGE-BASED DESCRIPTION LOGICS

A P-DL ontology is composed of a set of packages [3]. Terms (such as $Dog, Animal$) and axioms (such as $Dog \sqsubseteq Animal$) are defined in specific home packages.

Definition 1 (Package) *Let $O = (S, A)$ be an ontology, where S is the set of terms and A is the set of axioms over terms in S. A package $P = (\Delta_S, \Delta_A)$ of the ontology O is a fragment of O, such that $\Delta_S \subseteq S$, $\Delta_A \subseteq A$. A term $t \in \Delta_S$ or an axiom $t \in \Delta_A$ is called a* member *of P, denoted as $t \in P$. P is called the (only)* home package *of t, denoted as $\mathcal{HP}(t) = P$.*

Terms can be names of classes (i.e., concepts), properties (i.e., roles), or instances (i.e., individuals). A package can use terms defined in another package. In other words, an existing package can be *imported* into another package.

Definition 2 (Foreign Term and Importing) *A term t that appears in a package P, but has a home package Q that is different from P is called a* foreign term *in P. We say that P* imports *$Q : t$ and denote it as $Q \xrightarrow{t} P$. If any term defined in Q is imported into P, we say that P imports Q and denote it as $Q \mapsto P$.*

The importing closure *$I_{\mapsto}(P)$ of a package P contains all packages that are directly or indirectly imported into P, such that:*

- *(direct importing) $R \mapsto P \Rightarrow R \in I_{\mapsto}(P)$*
- *(indirect importing) $Q \mapsto R$ and $R \in I_{\mapsto}(P) \Rightarrow Q \in I_{\mapsto}(P)$*

A concept C is understandable *to a package P if the concept is constructed only using terms in P and concept terms in $I_{\mapsto}(P)$. A package P is said to be* complete *w.r.t. a concept C if C is understandable to P, otherwise P is* incomplete *w.r.t. C.*

Foreign terms can be used to construct local concepts. All concepts except for atomic foreign concepts in package P_i are i-concepts. Therefore, although P_i can import concepts in P_j ($i \neq j$), i-concepts are still disjoint with j-concepts. For example, an ontology O has two packages:

$\mathbf{P_{Animal}}$

(1a) $1 : Dog \sqsubseteq 1 : Carnivore$
(1b) $1 : Carnivore \sqsubseteq 1 : Animal$
(1c) $1 : Carnivore \sqsubseteq \forall 1 : eats.(1 : Animal)$

$\mathbf{P_{Pet}}$

(2a) $2 : PetDog \sqsubseteq 1 : Dog \sqcap 2 : Pet$
(2b) $2 : PetDog \sqsubseteq \exists 1 : eats.(2 : DogFood)$

We will omit the prefix "1:" and "2:" when there is no confusion. Both $1 : Dog \sqcap 2 : Pet$ and $\exists 1 : eats.(2 : DogFood)$ are 2-concepts constructed using some foreign terms.

Definition 3 (Acyclic and Cyclic Importing) *A P-DL ontology $\{P_i\}$ has acyclic importing relation if for any $i \neq j$, $P_j \in I_{\mapsto}(P_i) \rightarrow P_i \notin I_{\mapsto}(P_j)$, otherwise it has cyclic importing relation.*

For example, if $\mathbf{P_{Animal}}$ also imports the concept Pet from $\mathbf{P_{Pet}}$, the ontology will have cyclic importing relation.

For simplicity, we do not concern ourselves here with some additional features of P-DL, such as package hierarchy and scope limitation modifiers [3]. We denote the package based extension to DL as $\mathcal{P}$. Hence, $\mathcal{ALCP}$ is the package-based version of DL $\mathcal{ALC}$. In what follows, we will examine a restricted type of package extension which only allows acyclic import of concept names, denoted as $\mathcal{P}_{\mathcal{C}}$. For example, the importing $\mathbf{P_{Animal}} \xrightarrow{eats} \mathbf{P_{Pet}}$, or mutual importing between packages, is not allowed in $\mathcal{ALCP}_{\mathcal{C}}$.

The semantics of P-DL are expressed as follows: For a package-based ontology $\langle \{P_i\}, \{P_i \xrightarrow{t} P_j\}_{i \neq j} \rangle$, a distributed model is $M = \langle \{\mathcal{I}_i\}, \{r_{ij}^t\}_{i \neq j} \rangle$, where $\mathcal{I}_i = \langle \Delta_i, (.)_i \rangle$ is the local model of package P_i, $r_{ij}^t \subseteq \Delta_i \times \Delta_j$ is the interpretation for the importing relation $P_i \xrightarrow{t} P_j$. For convenience, we denote the identity function $f(x) = x$ as r_{ii}^t for any i and t. For any such a relation r from Δ_i to Δ_j and any individual $d \in \Delta_i$, $r(d)$ denotes the set $\{d' \in \Delta_j | \langle d, d' \rangle \in r\}$. For a subset $D \subseteq \Delta_i$, $r(D)$ denotes $\cup_{d \in D} r(d)$, is the image set of D.

We require that the importing relation meet the following requirements:

- Every importing relation is one-to-one in that it maps an object of $t^{\mathcal{I}_i}$ to a single unique object in $t^{\mathcal{I}_j}$.
- Importing relations are consistent for different terms, i.e., for any $i : t_1 \neq i : t_2$ and any $x, x_1, x_2 \in \Delta_i$, $r_{ij}^{t_1}(x) = r_{ij}^{t_2}(x)$ and $r_{ij}^{t_1}(x_1) = r_{ij}^{t_2}(x_2) \neq \emptyset \rightarrow x_1 = x_2$. Therefore, each object in the model of a source package corresponds uniquely to an object in the model of any target package for any interpretation of importing relations.
- Compositional Consistency: if $r_{ik}^{i:t_1}(x) = y_1$, $r_{ij}^{i:t_2}(x) = y_2$, $r_{jk}^{j:t_3}(y_2) = y_3$, , (where t_1 and t_2 may or may not be same), and y_1, y_2, y_3 are not empty set, then $y_1 = y_3$. (Compositional consistency helps ensure that importing relations (defined in below) is inferrable.)

The above requirements help to ensure the *transitive reusability* of modules and the exactness of distributed reasoning algorithm (relative to its centralized counterpart). A concept $i : C$ is *satisfiable* w.r.t. a P-DL $O = \langle \{P_i\}, \{P_i \xrightarrow{t} P_j\}_{i \neq j} \rangle$ if there exists a distributed model of O such that $C^{\mathcal{I}_i} \neq \emptyset$. A package P_k *witnesses* subsumption $i : C \sqsubseteq j : D$ (i, j, k can be different, C, D are understandable to P_k) iff $r_{ik}^C(C^{\mathcal{I}_i}) \subseteq r_{jk}^D(D^{\mathcal{I}_j})$ holds for every model of P_k and all packages in its importing closure.

The importing approach adopted by P-DL is different from the "linking" approach adopted by DDL and $\mathcal{E}$-Connections in that it

partially relaxes the strong local model disjointness assumption that is required in the other two formalisms.

The *image domain relation* between $\mathcal{I}_i$ and $\mathcal{I}_j$ is $r_{ij} = \cup_t r_{ij}^t$ and is strictly one-to-one. Consequently, r_{ij} in a P-DL model isomorphically "copies" the relevant partial domain from $\mathcal{I}_i$ to $\mathcal{I}_j$ and establishes unambiguous communication between the two packages. Since the construction of a local model is dependent on the structure of local models of imported modules, it is possible that local models are in fact partially overlapping.

Concept bridge rules in DDL and $\mathcal{E}$-Connection links can be easily reduced to P-DL axioms [2]. On the other hand, it also offers the possibility of avoiding many of the semantic difficulties of current modular ontology language proposals. For example, knowledge in one P-DL package can be transitively reused by other packages. The answer to a P-DL reasoning problem is semantically equivalent to that obtained by reasoning over an integrated ontology [2]. In what follows, we will investigate an algorithm for constructing a P-DL model for a specific reasoning problem for the package extended version of the DL $\mathcal{ALC}$.

3 REASONING WITH $\mathcal{ALCP}_\mathcal{C}$

3.1 Preliminaries: reasoning with $\mathcal{ALC}$

Modern description logics exploit tableau algorithms [1] for deciding concept satisfiability w.r.t. a knowledge base. A tableau algorithms for a specific DL language contains the following main elements:

- A *completion graph*, also called *tableau* that represents a model of the DL language. Such a completion graph has the "tree model" property.
- A set of *tableau expansion rules* to construct a complete and consistent completion graph.
- A set of *blocking rules* to detect infinite cyclic models and ensure termination.
- A set of *clash conditions* to detect logic contradictions.

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 (i.e. a completion graph) is found, C is satisfiable, otherwise C is unsatisfiable. Before the reasoning process starts, the concepts in O and C should be transformed into the *Negation Normal Form* (NNF), i.e., with negation only occurs in front of atomic concepts. 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 $C_\mathcal{T} = \prod_{(C_i \sqsubseteq D_i) \in \mathcal{T}} (\neg C_i \sqcup D_i)$. Any individual x in any model of $\mathcal{T}$ will be an instance of $C_\mathcal{T}$.

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

- if $C \in L(x)$, then $\neg C \notin L(x)$,
- if $C_1 \sqcap C_2 \in L(x)$, then $C_1 \in L(x)$ and $C_2 \in L(x)$,
- if $C_1 \sqcup C_2 \in L(x)$, then $C_1 \in L(x)$ or $C_2 \in L(x)$,
- if $\forall R.C \in L(x)$ and $R \in L(\langle x, y \rangle)$, then $C \in L(y)$,
- if $\exists R.C \in L(x)$, then there is some y such that $R \in L(\langle x, y \rangle)$ and $C \in L(y)$.

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

- $\sqcap$-rule: if $C_1 \sqcap C_2 \in L(x)$, x is not blocked, $\{C_1, C_2\} \not\subseteq L(x)$, then $L(x) = L(x) \cup \{C_1, C_2\}$
- $\sqcap$-rule: if $C_1 \sqcup C_2 \in L(x)$, x is not blocked, $\{C_1, C_2\} \cap L(x) = \emptyset$, then $L(x) = L(x) \cup \{C_1\}$ or $L(x) = L(x) \cup \{C_2\}$
- $\exists$-rule: if $\exists R.C \in L(x)$, x is not blocked, and x has no R-successor y with $C \in L(y)$, then create a new node y with $L(\langle x, y \rangle) = \{R\}$ and $L(y) = \{C\}$
- $\forall$-rule: if $\forall R.C \in L(x)$, x is not blocked, and there is a R-successor y of x with $C \notin L(y)$, then $L(y) = L(y) \cup \{C\}$

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 $L(x) \subseteq L(y)$, x is blocked. No expansion rule will be applied to a blocked node.

An $\mathcal{ALC}$ tableau contains a **clash** if there is $\{C, \neg C\} \in L(x)$ 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.

3.2 Distributed Tableaux for $\mathcal{ALCP}_\mathcal{C}$

The main idea behind the $\mathcal{ALCP}_\mathcal{C}$ tableau algorithm is to construct multiple federated local tableaux instead of a single global tableau. The connection between local tableaux is enabled by the image relations, i.e. a local tableau is able to create "image" nodes of its nodes in another local tableau. Formally, we have:

Definition 4 ($\mathcal{ALCP}_\mathcal{C}$ Distributed Tableau) *A distributed tableau for an $\mathcal{ALCP}_\mathcal{C}$ ontology $\{P_i\}$ is a tuple $\langle \{T_i\}, \{r_{ij}\}_{i \neq j} \rangle$, where T_i is a local $\mathcal{ALC}$ tableau for the package P_i, r_{ij} is the image relation between T_i and T_j, such that it creates one-to-one mappings from a subset of nodes in T_i to a subset of nodes in T_j. For any i-node x and j-node y, $(x, y) \in r_{ij}$, x is said the pre-image of y and y is the image of x. The local label $L_i(x)$ of x in a local tableau T_i only contains i-concepts.*

The P-DL semantics require that all images of a node in any local tableau should be uniquely identified. This can be done by keeping the original name of a node the same as its image nodes' names. For example, a node $i : x$ in a local tableau T_i can be copied in another local tableau T_j with the same name $i : x$ (we will omit the prefix when there is no confusion). As we will show later, such a property of image nodes helps to deliver nodes transitively across multiple local tableaux, and is essential to solve several known semantic difficulties in existing modular ontology languages.

It should be noted that the "copying" of nodes across multiple local tableaux does not require that they are completely identical. If a node x is copied in both local tableaux T_i and T_j, local labels $L_i(x)$ and $L_j(x)$ are strictly disjoint since i-concepts and j-concepts are disjoint. Some nodes in a local tableau may also not be shared by other local tableaux. Therefore, the partial sharing of nodes does not mean the sacrifice of local semantics.

The distributed tableau is not a combined tableau since all local tableaux will be autonomously created and maintained by the local reasoners provided for different packages. The local tableaux reside on their own reasoning servers and communicate in a peer-to-peer fashion. The communication between a local tableau T_j and a tableau T_i is supported by the following set of message primitives:

- **Membership** $m(y, C)$: given an individual y and an i-concept C, querying if there is a pre-image or image y' of y in T_i, such that $C \in L_i(y')$
- **Reporting** $r(y, C)$: given an individual y and an i-concept C, if there is a pre-image or image y' of y in T_i, $C \notin L_i(y')$, then $L_i(y') = L_i(y') \cup \{C\}$; if there is no existing pre-image or image y' of y, create a y' with $L(y') = \{C\}$ and add (y', y) to the image relation r_{ij}
- **Clash** $\perp(y)$: an individual y in T_j contains a clash.
- **Model** $\top(y)$: no expansion rules can be applied on y or any of its descendants in T_j.

If $i = j$, such messages are reduced to local operations: $m(y, C)$ is reduced to querying if $C \in L_j(y)$, $r(y, C)$ is reduced to adding C to $L_j(y)$, $\perp(y)$ is reduced to a local inconsistency report and $\top(y)$ is reduced to a local completion report. For convenience, we do not distinguish a local operation from a remote message operation.

To simplify the communication process we have:

Definition 5 (Concept Destination) *An atomic concept C or its negation $\neg C$'s destination is C's home package $\mathcal{HP}(C)$. A complex concept C's destination is the package in which it is generated. Destination of C is denoted as $\delta(C)$.*

A membership message $m(y, C)$ or reporting message $r(y, C)$ operation is always sent to the destination package of C, i.e., $\delta(C)$.

3.3 $\mathcal{ALCP}_{\mathcal{C}}$ Tableau Expansions

The $\mathcal{ALCP}_{\mathcal{C}}$ expansions rules are derived from the $\mathcal{ALC}$ expansion rules as follows: each module is only locally internalized, instead of being globally internalized w.r.t. a combined TBox; a local tableau can create "copies" of its local nodes in another local tableau (as needed) during an expansion. For a local tableau T_i, the expansion rules are:

- $\sqcap$-rule: if $C_1 \sqcap C_2 \in L_i(x)$, x is not blocked, then
  (1) if $m(x, C_1)$=false, do $r(x, C_1)$
  (2) if $m(x, C_2)$=false, do $r(x, C_2)$
- $\sqcup$-rule: if $C_1 \sqcup C_2 \in L_i(x)$, x is not blocked, but $m(x, C_1) \wedge m(x, C_2)$=false, then do $r(x, C_1)$ or $r(x, C_2)$
- $\exists$-rule: if $\exists R.C \in L_i(x)$, x is not blocked, and x has no R-successor y with $m(y, C)$=true, then create a new node y with $L_i(\langle x, y\rangle) = \{R\}$ and do $r(y, C)$
- $\forall$-rule: if $\forall R.C \in L_i(x)$, x is not blocked, and there is a R-successor y of x with $m(y, C)$=false, then do $r(y, C)$.
- CE-rule: if $C_{\mathcal{T}_i} \notin L_i(x)$, then $L_i(x) = L_i(x) \cup C_{\mathcal{T}_i}$, where $C_{\mathcal{T}_i}$ is the internalized concept for the TBox of P_i.

A distributed tableau is said to be *distributively complete* if no $\mathcal{ALCP}_{\mathcal{C}}$ expansion rule can be applied on any of its local tableau, and it is *distributively consistent* if all of its local tableaux are consistent.

For a satisfiability query involving a concept C sent to the package P_j (called the *witness package*), where C is understandable to P_j, we will first create its local tableau with an initial node x_0 ($\{C_{\mathcal{T}_j} \sqcap C\} \in L_j(x_0)$), and apply the $\mathcal{ALCP}_{\mathcal{C}}$ tableau expansion rules until a distributively complete and consistent tableau is found, or all search processes for such a distributed tableau fail.

Since different packages may stand for different semantic points of view, a satisfiability problem may have different answers when witnessed by different packages. In what follows, we assume that the satisfiability problem of a concept C is only witnessed by a package P_j that is complete w.r.t. C (i.e. C is understandable to P_j).

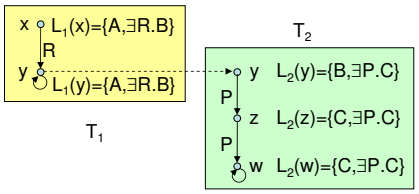

Figure 1. Distributed Tableaux Example

An example of such a distributed tableau is given in Figure 1. We have two packages: $P_1 : \top \sqsubseteq 1 : A, \top \sqsubseteq \exists(1 : R).(2 : B)$, $P_2 : \top \sqsubseteq (2 : P).(2 : C)$. P_1 imports concept B from P_2. The reasoning task is to check the consistency of P_1, therefore P_1 is the witness package. T_1 and T_2 are local tableaux for the two packages, with y as a shared node. Note that $L_1(y)$ and $L_2(y)$ are disjoint. The blocking strategy (e.g. y is blocked by x in T_1 since $L_1(y) \subseteq L_1(x)$) is also localized and is presented in the below.

3.4 Termination Strategy with Acyclic Importing

The termination of the algorithm can be obtained with subset blocking when there is only acyclic importing among packages. A P-DL ontology with acyclic importing relation has the property that local tableaux for the ontology have only unidirectional reporting messages. Formally, we have:

Definition 6 (Message Descendant) *A local tableau T_i is a direct message descendant of T_j, $i \neq j$, if there is at least one reporting message sent from T_j to T_i; T_i is a message descendant of T_j if it is a direct message descendant of T_j, or it is a direct message descendant of a T_j's message descendant.*

Lemma 1 *Given a P-DL ontology $\{P_i\}$, a local tableau T_i is a message descendant of a T_j, only if $P_i \in I_{\mapsto}(P_j)$*

This lemma follows from the observation that for any pair of T_i and T_j, if there is a reporting message from T_j to T_i, there must be an i-concept that is occurring in T_j, therefore package P_j must import P_i. This property can be easily extended to multiple packages and hence the lemma. This immediately leads to the following lemma:

Lemma 2 *Given a distributed tableau $\langle\{T_i\}, \{r_{ij}\}_{i \neq j}\rangle$ of an $\mathcal{ALCP}_{\mathcal{C}}$ ontology, if a local tableau T_i is T_j's message descendant, then T_j must not be T_i's message descendant.*

The blocking strategy for $\mathcal{ALCP}_{\mathcal{C}}$ works as follows:

Definition 7 (Subset Blocking for $\mathcal{ALCP}_{\mathcal{C}}$) *For a distributed tableau $\{T_i\}$ of an $\mathcal{ALCP}_{\mathcal{C}}$ ontology $\{P_i\}$, a node x is blocked by a node y, iff both x and y are in the same local tableau T_i, y is a local ancestor of x, and $L_i(x) \subseteq L_i(y)$.*

The correctness of such a blocking strategy is guaranteed by the following facts: For any node x in a local tableau T_i, if it has image nodes in another local tableau T_j ($i \neq j$), since the ontology has acyclic importing relation, neither T_j nor any of its message descendants will create new nodes in T_i with reporting messages. Therefore, if there is a path from x to another node y in T_i, the path only contains nodes in T_i and will not transverse via any image relations to other local tableaux. Thus, we can detect and prevent infinite expansion in T_i only with local topology information in T_i.

The blocking strategy allows expansion of a node even if its pre-image and image nodes in other local tableaux are locally blocked. For example, in Figure 1, while y is blocked in T_1, its image node in T_2 is further expanded. Such a strategy prevents infinite creation of image nodes in other local tableaux, but allows existing image nodes to be further validated.

The coordination of different local tableaux for $\mathcal{ALCP}_{\mathcal{C}}$ with acyclic importing operates as follows:

- **Waiting**: If a node x has sent a reporting message to another local tableau, x is temporarily blocked until it receives a clash($\bot$) message from any of its image node, or model($\top$) messages from all of its image nodes.
- **Clash Message**: If a node x contains a local clash (e.g. with both C and $\neg C$ in $L_i(x)$), or it receives a clash report, 1) if there is no other search choice (e.g. $\exists$-rule choices) that can be applied to x, send clash reports $\bot(x)$ to x's local direct ancestor and all pre-image nodes, and destroy all image relations from x; 2) if there are other search choices, restore the state (including image relations) of x to the state before the last choice, and try the next choice.
- **Model Message**: if no $\mathcal{ALCP}_\mathcal{C}$ expansion rules can be applied to x, x has no clash nor receives any clash message, there is no image node of x, or all image nodes return model messages, then send model messages $\top(x)$ to x's local direct ancestor and all pre-image nodes.

The intuition behind clash and model messages is that a node has a clash if it has local clash or *any* of its image node has a clash, and it is complete if it is locally complete and *all* its image nodes are complete. Such a strategy is driven by the fact that image nodes are "copies" of the pre-image node. The algorithm terminates when the root node in the local tableau of the witness package receives a clash message (i.e., the given concept is unsatisfiable) or a model message (i.e., the given concept is satisfiable).

3.5 $\mathcal{ALCP}_\mathcal{C}$ Expansion Examples

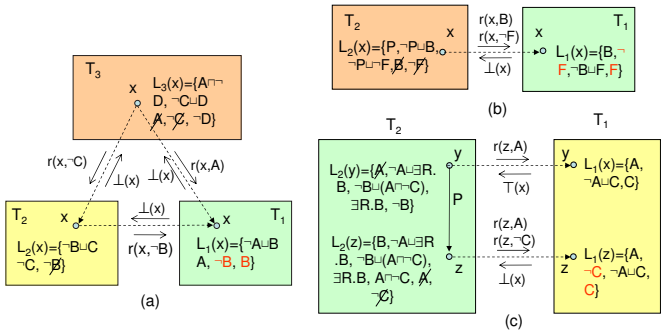

Figure 2. $\mathcal{ALCP}_\mathcal{C}$ Expansion Examples

Example 1: Transitive Subsumption Propagation. Given three packages: $P_1 : \{1 : A \sqsubseteq 1 : B\}$, $P_2 : \{1 : B \sqsubseteq 2 : C\}$, $P_3 : \{2 : C \sqsubseteq 3 : D\}$, test subsumption $1 : A \sqsubseteq 3 : D$ witnessed by P_3. The expansion and message exchange between local tableaux is shown in Figure 2 (a). Since there is no consistent distributed tableau for $A \sqcap \neg D$, $A \sqsubseteq D$ is witnessed by P_3.

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

Example 3: Reasoning from Local Point of View. Given two packages $P_1 : \{1 : A \sqsubseteq 1 : C\}$, $P_2 : \{1 : A \sqsubseteq \exists 2 : R.(2 : B), 2 : B \sqsubseteq 1 : A \sqcap (\neg 1 : C)\}$, test satisfiability of $1 : A$ witnessed by P_1 and P_2, respectively. It is easy to see A is satisfiable when witnessed by P_1, but unsatisfiable when witnessed by P_2 (Figure 2 (c) shows one possible search result). This example shows reasoning in P-DL always stands for the local semantic point of view of the witness packages, therefore the same reasoning problem can have different answers when witnessed from different packages. Note that in Figure 2 (c), z in T_2 is not further expanded due to the waiting strategy and the following clash message from T_1.

4 SOUNDNESS, COMPLETENESS AND COMPLEXITY

The basic intuition behind the $\mathcal{ALCP}_\mathcal{C}$ expansion rules is to reduce the inconsistency checking of distributed tableaux to only local tableau inconsistency checking. Since a local tableaux T_i is actually a representation for an ABox (denoted as $\mathcal{A}_i$) of a package P_i, an $\mathcal{ALCP}_\mathcal{C}$ expansion rule is equivalent to adding new facts to the ABox. For example, a local tableau with nodes x, y, $C \in L(x)$ and an R-edge from x to y is equivalent to an ABox $\{C(x), R(x, y)\}$, for some concept name C and role name P. A reporting message $r(x, C)$ adds new facts $C(x')$ and $x \rightarrow x'$ to the destination package of C, where $x \rightarrow x'$ is an image relation instance. We will denote a fact using lowercase Greek letters, e.g., α, β.

The result of applying an $\mathcal{ALCP}_\mathcal{C}$ expansion rule can be classified into three types using the ABox-type representation:

- *Augmenting*: New facts are inferred from existing facts in a local ABox (i.e. local tableau) and are added to the same ABox i.e. $\mathcal{A}' = \mathcal{A} \cup \{\alpha\}, \mathcal{A} \models \alpha$.
- *Searching*: Different candidates for inferred facts are added to the original local ABox i.e. $\mathcal{A}_1 = \mathcal{A} \cup \{\alpha\}, \mathcal{A}_2 = \mathcal{A} \cup \{\beta\}, \mathcal{A} \models (\alpha$ or $\beta)$.
- *Reporting*: New inferred facts are added to another local ABox, i.e. $\mathcal{B}' = \mathcal{B} \cup \{\alpha\}, \mathcal{A} \models \alpha$.

Since $\mathcal{ALCP}_\mathcal{C}$ expansions will send any concept fact to the ABox of its destination package, and inconsistency only occurs for the co-existence of $C(x)$ and $\neg C(x)$, if there is a global inconsistency, it must result in a local consistency in a certain ABox (i.e. local tableau). If there is an inconsistent local ABox, the entire set of ABoxes (i.e. the distributed tableau) must also be inconsistent. We denote $(\mathcal{A}, \mathcal{B})$ as the union of two ABoxes $\mathcal{A}, \mathcal{B}$.

Lemma 3 *If an ABox $\mathcal{A}$ is expanded by the $\mathcal{ALCP}_\mathcal{C}$ tableau expansion rules via*

- *augmenting from $\mathcal{A}$ to $\mathcal{A}'$, then $\mathcal{A}$ is consistent iff $\mathcal{A}'$ is consistent;*
- *searching from $\mathcal{A}$ to $\mathcal{A}_1$ or $\mathcal{A}_2$, then $\mathcal{A}$ is consistent iff $\mathcal{A}_1$ is consistent or $\mathcal{A}_2$ is consistent;*
- *reporting from $\mathcal{A}, \mathcal{B}$ to $\mathcal{A}, \mathcal{B}'$, then $(\mathcal{A}, \mathcal{B})$ is consistent iff $(\mathcal{A}, \mathcal{B}')$ is consistent.*

Given a set of distributively complete and consistent ABoxes (and hence also local tableaux) $\{\mathcal{A}_i\}$ for a P-DL $\{P_i\}$, we can build a distributed model as follows:

- For each ABox $\mathcal{A}_i$, the domain of local individuals Δ_i is the set of all individuals occurring in it.
- For each concept name $C \in P_i$, $C^{\mathcal{I}_i} = \{x | C(x) \in \mathcal{A}_i\}$
- For each role name $R \in P_i$, $R^{\mathcal{I}_i} = \{(x, y) | R(x, y) \in \mathcal{A}_i\}$
- For image relation r_{ij}, $r_{ij} = \{(x, y) | x \in \mathcal{A}_i, (x \rightarrow y) \in \mathcal{A}_j\}\}$

Consequently, a concept C is satisfiable w.r.t. a witness package P if and only if the $\mathcal{ALCP}_\mathcal{C}$ expansion rules can find a common distributed model for C, P and P's importing closure. Formally, we have:

Lemma 4 (Soundness and Completeness) *A distributively complete and consistent distributed tableaux can be obtained from the application of finite $\mathcal{ALCP}_\mathcal{C}$ expansion rules for the satisfiability problem of a concept C witnessed by a package P_j w.r.t. a P-DL ontology $\{P_i\}$, if and only if there exists a common distributed model for C and $\{I_{\mapsto}(P_j), P_j\}$.*

The complexity of reasoning consists can be expressed in terms of complexity of local reasoning processes and the communication process. The communication space cost is mainly due to the storage requirement for image domain relations. In the worst case each local tableau completely copies all nodes from other local tableux. If there are m nodes in total in all local tableaux, and there are n such local tableaux, the storage needed for image domain relations will be of the order of $O(m^2/n)$. Since n is finite and m can be polynomially bounded by the size of the combined terminology set of all packages and the testing concept (the well-known dept-first search strategy for $\mathcal{ALC}$ is still applicable), the upper bound on communication space complexity is polynomial w.r.t. the size of the whole P-DL ontology and the testing concept. If the ontology modules are well designed, much of the reasoning is likely to be localized within individual packages, resulting in space requirement that is significantly lower than the worst case upper bound.

The communication time cost is measured by the total number of messages exchanged between local tableaux. Assuming the distributed tableaux is *conceptually* reduced to a single combined tableau, a message operation can be reduced to a local operation. For example, $m(x, C)$ is reduced to testing if $C \in L(x)$ and $r(x, C)$ is reduced to adding C to $L(x)$. Consequently, the number of such message operations is bounded by the total number of equivalent local operations in an $\mathcal{ALC}$ reasoning process w.r.t a combined knowledge base of all packages. Therefore, we have the following lemma:

Lemma 5 (Complexity) *The run time complexity of the tableau-based distributed reasoning algorithm presented in this paper for $\mathcal{ALCP}_\mathcal{C}$ P-DL is no greater the run time complexity of the classical tableau-based algorithm for $\mathcal{ALC}$ in the worst case w.r.t. the size of the input concept and the total size of the combined terminology set of all packages.*

It is known the reasoning complexity for $\mathcal{ALC}$ -satisfiability w.r.t. cyclic TBoxes is EXPTIME [9]. Therefore, $\mathcal{ALCP}_\mathcal{C}$ also has the same reasoning complexity upper bound. The practical performance of the algorithm can be improved by caching strategy. For example, if $m(x, C)$ returns true, the calling package can store the result locally thus avoid repeated queries.

5 CONCLUSIONS

We have presented a distributed tableau-based reasoning algorithm for the package-based extension of the DL language $\mathcal{ALCP}_\mathcal{C}$. The approach adopted in this algorithm is novel in several respects in that it:

1. Strictly avoids combining the local ontology modules in a centralized memory space: Unlike the tableau algorithm that is based on classical DL semantics, this approach enables distributed reasoning with localized semantics and each local reasoning module can operate in a peer-to-peer fashion.
2. Relaxes the local domain disjointness assumption adopted in existing modular ontology reasoning algorithms, thereby avoiding some of the semantic difficulties associated with existing approaches.
3. 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, without having higher complexity of reasoning.

Work in progress is aimed at overcoming some of the limitations of the distributed reasoning algorithm for $\mathcal{ALCP}_\mathcal{C}$ P-DL described in this paper:

- Module Expressivity: Web ontology languages such as OWL are based on expressive extensions of $\mathcal{ALC}$, e.g., $\mathcal{SHOIQ}(D)$. Although we restricted ourselves to $\mathcal{ALC}$ in this paper, we believe that the strategy used in this paper can be extended to many expressive extensions of $\mathcal{ALC}$.
- Cyclic Importing: Cyclic importing among ontology modules is unavoidable in real-world applications. We are currently working on a new version of the reasoning algorithm that is able to handle such cases and allows bidirectional message flow between local reasoners with more a complicated backtracking strategy.

ACKNOWLEDGEMENT: This research is supported by grants from the National Science Foundation (0219699) and the National Institutes of Health (GM066387) and the Iowa State University Center for Integrative Animal Genomics to Vasant Honavar.

REFERENCES

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

[2] J. Bao, D. Caragea, and V. Honavar. On the semantics of linking and importing in modular ontologies (extended version). Technical report, TR-408 Computer Sicence, Iowa State University, 2006.

[3] J. Bao, D. Caragea, and V. Honavar. Towards collaborative environments for ontology construction and sharing. In *International Symposium on Collaborative Technologies and Systems (CTS 2006)*. 2006.

[4] A. Borgida and L. Serafini. Distributed description logics: Directed domain correspondences in federated information sources. In *CoopIS/DOA/ODBASE*, pages 36–53, 2002.

[5] B. C. Grau, B. Parsia, and E. Sirin. Tableau algorithms for e-connections of description logics. Technical report, University of Maryland Institute for Advanced Computer Studies (UMIACS), TR 2004-72, 2004.

[6] B. C. Grau, B. Parsia, and E. Sirin. Working with multiple ontologies on the semantic web. In *International Semantic Web Conference*, pages 620–634, 2004.

[7] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-connections of description logics. In *Description Logics Workshop, CEUR-WS Vol 81*, 2003.

[8] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-connections of abstract description systems. *Artif. Intell.*, 156(1):1–73, 2004.

[9] K. Schild. Terminological cycles and the propositional -calculus. In *KR*, pages 509–520, 1994.

[10] L. Serafini, A. Borgida, and A. Tamilin. Aspects of distributed and modular ontology reasoning. In *IJCAI*, pages 570–575, 2005.

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

[12] L. Serafini and A. Tamilin. Distributed instance retrieval in heterogeneous ontologies. In *Proceedings of SWAP 2005, CEUR Workshop Vol 166*, 2005.

[13] L. Serafini and A. Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *ESWC*, pages 361–376, 2005.

[14] E. Sirin and B. Parsia. Pellet: An OWL DL Reasoner. In *Description Logics Workshop*, 2004.