A Tableau-based Federated Reasoning Algorithm for Modular Ontologies*

Jie Bao[1], Doina Caragea[2], Vasant Honavar[1]
[1]Artificial Intelligence Research Laboratory,
Department of Computer Science,
Iowa State University, Ames, IA 50011-1040, USA
[2]Department of Computing and Information Sciences
Kansas State University, Manhattan, KS 66506, USA
[1]{baojie, honavar}@cs.iastate.edu, [2]dcaragea@ksu.edu

Abstract

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 modular ontology language that extends description logics. 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 approach 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.

1 Introduction

There is a growing recent interest in ontology language features to support modular ontologies as well as approaches to reasoning with multiple ontology modules including: Distributed Description Logics (DDL)[6], $\mathcal{E}$-connections [9, 8] and Package-based Description Logics (P-DL) [5, 4]. Of particular interest in this context are algorithms for reasoning with multiple, distributed and autonomous ontology modules. Reasoning with ontologies in such a setting presents several challenges:

- The reasoning task involves not a single ontology, but a collection of ontologies about a domain of interest that are created and maintained by autonomous groups.
- In many cases, integrating distributed ontologies into one consistent centralized ontology is not possible for several reasons: the ontologies may be large and communication overhead is too expensive; the autonomous entities that control an ontology may be unwilling to share it in its entirety although be willing to answer queries to the ontology. In such a setting, it is not feasible to reduce the problem of reasoning over distributed ontology modules to the problem of reasoning over a single centralized ontology.
- In general, an ontology may reuse terms defined in another ontology. Mutual or cyclic reuse is also common.

Several authors have recently investigated distributed reasoning algorithms for modular ontologies. Serafini et. al. [11, 10] describe a tableau-based reasoning algorithm for **DDL**. 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* that relate concepts in one module to concepts in another module. For example, cnsider ontology modules i and j in which the concepts A, B and G, H respectively are defined, given the bridge rules $i : A \xrightarrow{\sqsupseteq} j : G$, $i : B \xrightarrow{\sqsubseteq} j : H$ and module i entails $A \sqsubseteq B$, then it is possible for module j to infer that $G \sqsubseteq H$.

Grau et al. [8, 7] present a tableau-based reasoning procedure for $\mathcal{E}$**-Connections**. $\mathcal{E}$-connections divides roles into disjoint sets of *local roles* (connecting concepts in one module) and *links* (connecting concepts in different modules). 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. $1 : DogOwner \sqsubseteq \exists owns.(2 : Dog)$. The tableau-based reasoning procedure for $\mathcal{E}$-Connections, implemented in the Pellet reasoner, generates a set of tableaux (trees) linked by $\mathcal{E}$-connection instances (cross-module role instances).

Bao et.al. [3] describe a distributed reasoning algorithm for **P-DL** with *acyclic importing*. This algorithm adopts a federated approach to reasoning using distributed storage of a global tableau. Some of local tableaux may share some nodes (i.e. "image" nodes) and communicate by sending messages to each other. Thus, search for a *model* of the ontology is distributed across the local tableaux.

*A longer version of the paper is available as a technical report at http://archives.cs.iastate.edu/documents/disk0/00/00/04/67/index.html

However, existing approaches to reasoning with modular ontologies suffer from several limitations. Both DDL and $\mathcal{E}$-Connections, because of their limited expressivity, lack support for certain types of reasoning tasks. For example, DDLs have no support for inter-module role relations, whereas $\mathcal{E}$-connections lack inter-module subsumptions. Both DDL and P-DL reasoning algorithms do not allow mutual or cyclic references (bridge rules in DDL, term importing in P-DL) of concepts among ontology modules.

Current implementation of the $\mathcal{E}$-Connections reasoner, motivated by the "combined tableau" idea [8, 7], only "colors" local tableaux without separating them. Therefore, reasoning relies on one (combined) ABox thereby forcing the TBoxes of all modules to be loaded (through internalization) into the reasoner. The strategy actually loads all ontology modules into a single memory space thus makes a de facto ontology integration, which sacrifices many of the benefits of modular ontologies (e.g. scalability).

Against this background, we present an improved federated reasoning algorithm that overcomes many of these limitations and offers several advantages over existing approaches. It strictly avoids combining the local ontology modules in a centralized memory space using distributed reasoning with localized P-DL semantics thereby allowing local reasoning modules to operate in an asynchronous, peer-to-peer fashion. It supports reasoning with both inter-module subsumption and inter-module role relations and allows arbitrary references of concepts among ontology modules. 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 [4].

2 Package-based Description Logics

This section briefly reviews basic features of Package-based Description Logics (P-DL) as given in [5, 4]. In P-DL, an ontology is composed of a collection of modules called *packages*. Each term (name of a concept, property or individual) or axiom is associated with a *home package*. A package can use terms defined in other packages:

Definition 1 (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 packages such that:*

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

Definition 2 (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, an ontology O with acyclic importing is:

$\mathbf{P_{Animal}}$

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

$\mathbf{P_{Pet}}$

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

By reusing terms defined in $\mathbf{P_{Animal}}$, the ontology is able to model both inter-module concept subsumption (e.g. axiom 2a) and role relations (e.g. axiom 2b). We denote the package extension to Description Logics (DL) as $\mathcal{P}$. For example, $\mathcal{ALCP}$ is the package-based version of DL $\mathcal{ALC}$. In what follows, we will examine a restricted type of package extension which only allows import of concept names, denoted as $\mathcal{P_C}$.

For a package-based ontology $\langle \{P_i\}, \{P_i \rightarrow P_j\}_{i \neq j} \rangle$, a distributed model is $M = \langle \{\mathcal{I}_i\}, \{r_{ij}\}_{i \neq j} \rangle$, where $\mathcal{I}_i = \langle \Delta_i, (.)_i \rangle$ is the local model of package P_i, $r_{ij} \subseteq \Delta_i \times \Delta_j$ is the interpretation for the *image domain relation* $P_i \rightarrow P_j$. $(x, y) \in r_{ij}$ indicates an individual $y \in \Delta_j$ is an "image" (or copy) of an individual $x \in \Delta_i$. Therefore, local models of P-DL can be partially overlapping.

To ensure module transitive reusability and reasoning correctness, we require that every image domain relation has the following properties:

- It is one-to-one: for any $x \in \Delta_i$, there is at most one $y \in \Delta_j$, such that $(x, y) \in r_{ij}$.
- It is compositional consistent: $r_{ij} = r_{ik} \circ r_{jk}$, where $\circ$ denotes function composition. Therefore, semantic relations between terms in i and terms in k can be inferred even if k doesn't directly import terms from i.

For a relation r_{ij} and any individual $d \in \Delta_i$, $r_{ij}(d)$ denotes the set $\{d' \in \Delta_j | \langle d, d' \rangle \in r\}$. For a subset $D \subseteq \Delta_i$, $r_{ij}(D)$ denotes $\cup_{d \in D} r_{ij}(d)$, is the image set of D.

A concept $i : C$ is *satisfiable* w.r.t. a P-DL $O = \langle \{P_i\}, \{P_i \rightarrow P_j\}_{i \neq j} \rangle$ if there exists a distributed model of O such that $C^{\mathcal{I}_i} \neq \emptyset$. O *entails* subsumption $i : C \sqsubseteq j : D$ (i may or may not be the same as j), denoted as $O \vDash i : C \sqsubseteq_P j : D$ iff $r_{ij}(C^{\mathcal{I}_i}) \subseteq D^{\mathcal{I}_j}$ holds in every model of O.

3 Distributed Reasoning for P-DL

We extend the tableau-based approach to distributed reasoning with P-DL modules introduced in [3] to a more general setting wherein arbitrary importing (e.g. cyclic or mutual importing) among packages is allowed, and the tableau search process is preformed in a parallel, asynchronous fashion. We demonstrate the strategy with the package-extended version of a representative DL $\mathcal{ALC}$ that allows importing of concepts between packages, i.e. $\mathcal{ALCP_C}$.

3.1 $\mathcal{ALC}$ Reasoning

We first briefly introduce the tableau algorithm for traditional DLs. e.g. $\mathcal{ALC}$. A tableau is a representation of a model of a logic language, and in particular, of an ontology. Popular representation forms of a tableau include ABox and Completion Graph [2], while each of them can be transformed into the other. In this paper, we adopt the ABox representation since it is more explicit for incremental tableau storage needed for our algorithm.

An ABox contains a set of *facts* in the form of $C(x)$, $P(x, y)$, $x = y$ or $x \neq y$, where x, y are individuals, C is a concept name, and P is a property name. To test the satisfiability of a concept C w.r.t. a TBox $\mathcal{T}$, an initial ABox $\mathcal{A}_0$ is created as $(C \sqcap C_\mathcal{T})(x_0)$, where $C_\mathcal{T}$ is the *internalization concept* of $\mathcal{T}$: $C_\mathcal{T} = \bigsqcap_{(C_i \sqsubseteq D_i)\in\mathcal{T}} (\neg C_i \sqcup D_i)$. Each individual x in any ABox of $\mathcal{T}$ will be an instance of $C_\mathcal{T}$.

New facts can be inferred from existing facts based on *tableau expansion rules* and added to the ABox. Assuming that all concepts are in Negation Normal Form (NNF), the $\mathcal{ALC}$ tableau expansion rules for traditional reasoning process (i.e. on a single ontology) are:

- $\sqcap$-rule: if ABox $\mathcal{A}$ contains $(C_1 \sqcap C_2)(x)$ but not both $C_1(x)$ and $C_2(x)$, then $\mathcal{A}'$=$\mathcal{A} \cup \{C_1(x), C_2(x)\}$
- $\sqcup$-rule: if ABox $\mathcal{A}$ contains $(C_1 \sqcup C_2)(x)$ but neither $C_1(x)$ or $C_2(x)$, then $\mathcal{A}_1$=$\mathcal{A} \cup \{C_1(x)\}$, $\mathcal{A}_2$=$\mathcal{A} \cup \{C_2(x)\}$
- $\exists$-rule: if $\mathcal{A}$ contains $(\exists R.C)(x)$ but no individual y such that $C(y)$ and $R(y, z)$ in $\mathcal{A}$, then $\mathcal{A}'$=$\mathcal{A} \cup \{C(y), R(x, y)\}$ where y is an individual name not occurring in original $\mathcal{A}$.
- $\forall$-rule: if $\mathcal{A}$ contains $(\forall R.C)(x), R(x, y)$ but no $C(y)$, then $\mathcal{A}'$= $\mathcal{A} \cup \{C(y)\}$

An ABox clash corresponds to the scenario: $\{C(x), \neg C(x)\} \subseteq \mathcal{A}$ (for any individual x and any concept name C). An ABox is *consistent* if it contains no clash, and is *complete* if no expansion rule can be applied on it. Note that the $\sqcup$-rule is nondeterministic in that it generate multiple possible new facts. The algorithm needs to try different choices i.e., *search* for different possible models. Once a chosen path leads to an inconsistency, the algorithm needs to backtrack to the ABox state before the choice, and try other remaining choices.

A concept C is said to be satisfiable w.r.t. a TBox $\mathcal{T}$ if and only if the algorithm finds a consistent and complete ABox for both C and $C_\mathcal{T}$.

3.2 Incremental Distributed Tableau Storage

Tableau-based reasoning for modular ontologies [11, 10, 8, 7, 3] usually exploits multiple local tableaux instead of a single tableau. This supports the localized semantics requirement for modular ontologies [4], i.e., that there is no required *global model*. Thus, reasoning is carried out to obtain a set of connected local models for a modular ontology.

Each of the existing approaches assumes different properties of local models, and as a consequence, requires different procedures for constructing such local models and local tableaux. DDL and $\mathcal{E}$-connections reasoning algorithms [11, 10, 8, 7] assume domains (the set of individuals) of local tableaux are disjoint, while P-DL reasoning algorithm [3] allows them to be partially overlapping. Advantages of the later approach include support for inter-module subsumption and transitive reusability of modules [3].

In this paper, we assume incremental instantiation of ABoxes to simplify the description of the algorithm.

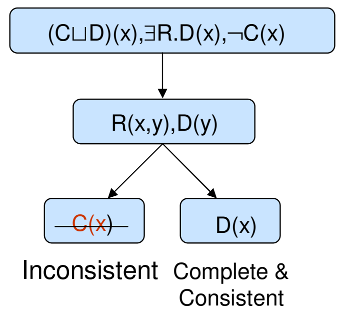

Figure 1. Incremental Instantiation of ABoxes

We represent an ABox by a series of *nodes*, where each node contains one or more *facts*. The *root* node contains all the initial facts in the ABox. By applying the tableau expansion rules, starting with the root node, we can successively generate new inferred facts. The inferred facts are added as to the successor of the current node, called its *expansion successor*. The edge linking a node to its successor is called an *expansion edge*. Multiple choices for expansion (e.g. using the $\sqcup$-rule), result in multiple successors. Recursive application of the tableau expansion rules yields an *ABox Tree*, with each node in the tree representing an ABox that contains all the facts on the path to that node from the root node. When the algorithm terminates, each leaf in the tree corresponds to either an inconsistent ABox or a complete and consistent ABox (See Figure 1 for an example).

For each node n, let $\Delta(n)$ denoting the tree that n belongs to; $f(n)$ is the set of facts in n; and $\mathcal{A}(n)$ is the ABox containing all the facts in n and all of its ancestors up to the root node. Thus, if a node n is a successor of node m, $\mathcal{A}(n) = \mathcal{A}(m) \cup f(n)$. Each package P_i participating in the reasoning process has associated with it, an initial node n_i^0. Each individual introduced to the ABox tree of P_i must be an instance of $C_{\mathcal{T}_i}$.

Given a P-DL ontology $O = \{P_i\}$, we can obtain an ABox forest wherein each package has associated with it exactly one ABox tree. A distributed ABox $\mathcal{A}_d$ (i.e., a distributed model) of O is represented by a set of complete and consistent leaf nodes $\{n_i\}$, one from each ABox tree, where $\mathcal{A}(n_i)$ is a local ABox (i.e. a local model), and $\mathcal{A}_d = \cup_i \mathcal{A}(n_i)$.

Thus, each ABox tree is maintained by the corresponding local reasoner. The reasoning process is carried out by a federation of such local reasoners. Since each ABox tree is only locally internalized, integration of the ontology modules into a centralized ontology or of local models into a centralized model is strictly avoided.

3.3 Distributed Tableau Expansion

To construct a distributed model for an $\mathcal{ALCP_C}$ ontology, we start with a list of initial ABox nodes corresponding to each package in the ontology. New facts can be added to the ABox forest by applying tableau expansion rules similar to that of $\mathcal{ALC}$. However, the traditional $\mathcal{ALC}$ expansion procedure needs to be modified in several important aspects. We refer to the resulting expansion rules as $\mathcal{ALCP_C}$ expansion rules.

First of all, new facts should be sent to a "destination" ABox tree to reduce the cost of detecting a clash. Since a concept can be imported into another package, it is possible that a fact $C(x)$ is generated from an expansion in an ABox of a package that is not C's home package. Therefore, $C(x)$ and $\neg C(x)$) can be generated in different local ABoxe trees in which case, a clash cannot be *locally* detected. However, global check for such clashes is expensive. Hence, we adopt a strategy that is designed to minimize the cost of detecting clashes. We start by introducing some relevant definitions:

Definition 3 (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 tree in which it is generated. Destination of C is denoted as $\delta(C)$.*

Each generated fact $C(x)$ from any ABox tree node will be sent to an ABox tree of the destination of C, i.e., $\delta(C)$. The destination ABox tree of a fact f is denoted by $\delta(f)$. Thus, all clashes can be detected locally. Note that there is no role importing in $\mathcal{ALCP_C}$, therefore a role fact $P(x, y)$ is always generated in (and stays in) the ABox tree of P's home package.

We refer to a fact that is sent from one ABox tree to another as a *fact message*, and we add a *message edge* from the sending node to the receiving node. In such cases, two copies of the fact are kept in the two nodes. For example, in Figure 2 at Time 4, $B_1(x)$ is generated in the ABox tree T_A (of package A), but B_1 has home package B. Therefore, $B_1(x)$ is sent to the ABox tree T_B of B, and finally results in a local clash that is detected locally in T_B.

Since a fact (e.g., $C(x)$) may be shared by two ABox trees (e.g. T_i, T_j), an individual name (e.g., x) may also appear in the two trees. We denote such a shared individual name in different ABox trees with prefixes such as $i : x$ and $j : x$. However, we assume those names can still be identified as variances for the same individual.

The termination of the algorithm can still be ensured using the *subset blocking* [2]: for an ABox tree node n, the application of the $\exists$-rule is blocked to an individual x by an individual y iff $\{D|D(x) \in \mathcal{A}(n)\} \subseteq \{D'|D'(y) \in \mathcal{A}(n)\}$.

Note that the algorithm we presented so far is equivalent to the completion graph-based $\mathcal{ALCP_C}$ reasoning algorithm in [3] if we allow only acyclic importing between packages. Thus, based on Lemmas 1 and 2 of [3], if only acyclic importing between packages is allowed, the message edges between ABox trees are guaranteed to be uni-directional: once an ABox tree t_1 receives a fact from t_2, there is no path in the ABox forest (linked by message edges) from a node in t_1 ending in a node in t_2, hence there is no risk of message loop. However, if cyclic importing is allowed, in order to guarantee termination of the algorithm, we need to find a way to prevent message looping.

3.4 Handling Cyclic Importing

Cyclic importing presents additional difficulties in message exchange among ABox trees because it may lead to ABox trees waiting for each other in a cycle or a deadlock. How can we avoid such a deadlock?

To develop some intuition regarding this problem, consider the logical meaning of edges in the ABox forest. If a fact f is generated by applying expansion rules at a node n, f is actually the logical consequence of some facts in the ABox $\mathcal{A}(n)$. For example, in Figure 1, the fact $D(x)$ is one possible logical consequence of $(C \sqcup D)(x)$. Therefore, if a new fact f that is a (direct of indirect) logical consequence of $\mathcal{A}(n)$ is to be added on the ABox tree, it should be added as a child of node n. For example, in Figure 2 Time 5, a fact $A_2(x)$ is generated in the ABox tree T_B while the destination of A_2 is T_A. However, since an ancestor of $A_2(x)$ has received a fact $B_1(x)$ from T_A, $A_2(x)$ is an indirect logical consequence of $B_1(x)$. Hence, $A_2(x)$ should be added to T_A under the node containing $B_1(x)$. We refer to an ABox graph containing both expansion edges and message edges as an *ABox graph*.

Further note that an ABox graph is a representation of global tableaux, while each branch in a local ABox tree stands for a search choice in finding such a global tableau. Thus, when adding new facts to the graph, the distinction between the different search choices must be maintained. In other words, different reasoning subtasks should be kept separate.

The preceding considerations suggest the following strategy for avoiding message looping or deadlock in the presence of cyclic importing:

- Let each node n maintain a contact list $lst(n)$ of nodes from other ABox trees.
- Initial contact list of a root node is initialized with the list of root nodes of other ABox trees.

- If a new node n is generated under a node m in the same ABox tree $\Delta(m)$, $lst(n) \Leftarrow lst(m)$.
- If a node n in an ABox tree T generates a new fact f such that $\delta(f) \neq T$, f is sent to a new node l under m on the destination tree $\delta(f)$, where $m \in lst(n)$. $lst(l)$ is obtained by merging $lst(n)$ and $lst(m)$: if both n and m contain contacts from an ABox tree, discard one if it is an ancestor of the other on the ABox graph)

This strategy ensures that a node always has at most one contact node from each of the other ABox trees:

Lemma 1 *For a node n in an ABox tree T, for any package $p, p \neq \Delta(n)$, there is at most one node m in the ABox tree of p such that m is in n's contact list :* $|\{m | m \in lst(n), \Delta(m) = p\}| \leqslant 1$.

Proof sketch: Suppose there are two nodes m_1 and m_2 from one ABox tree of p that are both in $lst(n)$. Then there must be two paths from m_1 to n_1 and from m_2 to n_2, where n_1 and n_2 are on the path from n to its root. Without loss of generality, suppose n_2 is a descendent of n_1. Then there must be an ancestor of n_2, say n_3, such that $n_3 \in lst(m_2)$. Hence, $n_1 = n_3$ or n_3 must be a descendent of n_1 and there must be a path from n_3 through some ancestor of m_2 to m_2. However, such nodes must be m_1's descendants and hence m_2 must be a descendant of m_1. According to our contact merging rule, only m_2 will be kept in n's contact list.

The contact list update rule described above ensures that the contact list contains only the most "recent" message sender from ABox trees of other packages. We also denote $lst_i(n)$ as the contact of n on the ABox tree i.

From Lemma 1 it follows that:

Lemma 2 *If a node has two ancestors n, m in an ABox graph, it must be the case that n is an ancestor of m on the ABox graph, or m is an ancestor of n on the ABox graph (but not both).*

This lemma implies that when adding new facts to the graph, the distinction between the different search choices (tree branches) must be maintained across all local reasoners. For example, the Figure 2 Time 6, $B_2(x)$ in T_B has two ancestors $B_1(x)$ and $B_2(x)$ in T_A, and $B_1(x)$ is a local ancestor of $B_2(x)$. Therefore, the set of all ancestor nodes of a node n on the ABox graph contains facts associated with a single search branch.

Thus, there is in effect, a *virtual global* ABox (directed) *graph* that corresponds to a conceptually integrated ontology. This graph can be decomposed into multiple smaller local ABox *trees* (by copying some nodes as needed). There must be no (directed) loop on the ABox graph, thereby ensuring the termination of the algorithm:

Lemma 3 (Termination) *Let C_0 be an $\mathcal{ALCP_C}$ -concept description in NNF. There cannot be an infinite sequence of $\mathcal{ALCP_C}$ rule applications.*

We summarize the expansion rules for $\mathcal{ALCP_C}$ in what follows, starting with some notations: for any node n on an ABox tree k, $\mathcal{A}_k(n)$ is the ABox represented by n; for any fact f, $m(n, f)$ is a query from n for f's existence in its destination, i.e., if $f \in \mathcal{A}_{\delta(f)}(lst_{\delta(f)}(n))$; $r(n, f)$ is an action that sends a fact f to its destination, i.e., creates a new node containing f under $lst_{\delta(f)}(n)$. When $\delta(f) = n$, $m(n, f)$ is reduced to a local query that if $f \in \mathcal{A}_k(n)$, and $r(n, f)$ is reduced to a local action that adds a new node containing f under n. The $\mathcal{ALCP_C}$ expansion rules are:

- $\sqcap$-rule: if $\mathcal{A}_k(n)$ contains fact $(C_1 \sqcap C_2)(x)$, x is not blocked in $\mathcal{A}_k(n)$, then do $r(n, C_i(x))$ if $m(n, C_i(x))$= false, for $i = 1, 2$
- $\sqcup$-rule: if $\mathcal{A}_k$ contains fact $(C_1 \sqcup C_2)(x)$, x is not blocked in $\mathcal{A}_k(n)$, but $m(n, C_1(x)) \vee m(n, C_2(x))$=false, then do $r(n, C_1(x))$ or $r(n, C_2(x))$
- $\exists$-rule: if $\mathcal{A}_k$ contains fact $(\exists R.C)(x)$, x is not blocked in $\mathcal{A}_k(n)$, and $R \in P_k$, for any $R(x, z) \in \mathcal{A}_k$, we have $m(n, C(z))$=false, then do $r(n, R(x, y))$ and $r(n, C(y))$ where y is a new individual name.
- $\forall$-rule: if $\mathcal{A}_k$ contains $(\forall R.C)(x), R(x, y)$, x is not blocked in $\mathcal{A}_k(n)$, $R \in P_k$, and $m(n, C(y))$= false, then do $r(n, C(y))$.

3.5 Asynchronous Federated Reasoning

We need several types of messages to complete our description of the search for a complete and consistent global tableau, in which the targets of all messages are all n's contacts and n's parent node on the local ABox tree ($\Delta(n)$):

- If a local clash is found in $\mathcal{A}(n)$, mark n as $\perp$, send clash messages.
- If *all* expansion successors of n are marked as $\perp$, or *any* of the message successors is marked as $\perp$, mark n as $\perp$; send clash messages.
- If $\mathcal{A}(n)$ is locally complete, mark n as $\top$, send model messages .
- If *any* expansion successors of n are marked as $\top$, and *all* message successors is marked as $\top$, mark n as $\top$, send model messages .

In the case of centralized tableau reasoning, given any set of three nodes x, y, z, where x is the expansion parent of y and z , it must be the case that $\mathcal{A}(x)$ has a clash iff $\mathcal{A}(y)$ has a clash and $\mathcal{A}(z)$ has a clash. However, this is not necessarily true in the distributed setting. For example, even when all of x's successors make the tableux inconsistent, if none of message recipients of x find a clash, it is possible for them to send back to x a different fact and open a new branch under x (also see the example in Figure 2 Time 7, node $B_1(x)$ in T_A, its only successor $A_2(x)$ contains a clash, while on Time 8 a new branch $A_3(x)$ is created by a message from T_B). Unlike in the centralized setting where

$\mathcal{A}(x)$ is consistent iff $\mathcal{A}(y)$ is consistent or $\mathcal{A}(z)$ is consistent, in the distributed setting, remote inconsistencies might be discovered on x's copies in other ABox trees although x is locally consistent.

Once a node sends a fact message to one or more ABox trees, *we do not require* that the node wait until an answer is received from other ABox trees. A search branch may be closed on the basis of a local clash or clash messages received from other ABox trees. Therefore, local reasoners for each of the ABox trees may work on different reasoning subtasks concurrently to make the best use of the computational resources available to each of them. Thus, the algorithm presented here also relaxes the requirement adopted in [3] for waiting for a response from the recipient of a fact message before proceeding.

Figure 2 (Time 1 -Time 7) illustrates the working of the algorithm presented in this paper. The example ontology contains two packages A and B. Package A contains axioms $\top \sqsubseteq A_1 \sqcap \neg A_3$, $A_1 \sqsubseteq B_1$, $A_2 \sqsubseteq B_2$; package B contains axioms $B_1 \sqsubseteq A_2 \sqcup A_3$, $B_2 \sqsubseteq A_3$. Thus, the two packages mutually import each other. The reasoning task is to check the consistency of the ontology. The result is that the ontology is inconsistent.

4 Complexity

It is easy to show that the worst case time complexity in the proposed distributed reasoner is no worse than that of a centralized reasoner on $\mathcal{ALC}$, i.e. EXPTIME for consistency of $\mathcal{ALC}$ -ABoxes [2]. It can be shown by a *conceptual* reduction from $\mathcal{ALCP_C}$ expansions to $\mathcal{ALC}$ expansions: Note that all local ABox trees are generated and maintained at a single location, the given $\mathcal{ALCP_C}$ expansion rules will be reduced to $\mathcal{ALC}$ expansion rules. Thus, the *total* number of tableau expansions required to find an $\mathcal{ALCP_C}$ model in *all* local reasoners is the same as the number of tableau expansions required to find an $\mathcal{ALC}$ model for an integrated ontology of all ontology modules.

However, since different reasoners may *concurrently* explore different tableau search choices in $\mathcal{ALCP_C}$ expansions, in practice, the proposed distributed reasoner can be significantly faster than its centralized centralized counterpart.

The space complexity of the proposed algorithm might be a source of concern when the individual reasoners operate in an asynchronous fashion (i.e., the individual reasoners do not wait for responses to messages that they have sent to other reasoners before proceeding). In this mode of operation, the local reasoners depart from the strictly depth-first search (with PSPACE complexity) [1]. While this allows each reasoner to make the best use of its computational resources, the departure from strictly depth-first search can lead to a worst case space complexity that is EXPSPACE (same as that of $\mathcal{ALC}$). Alternatively, each local reasoner can implement a mixed strategy of switching to the synchronous (i.e. waiting for responses before proceeding) mode when available memory is limited, thus requires only polynomial space.

In practice, however, the proposed $\mathcal{ALCP_C}$ reasoning algorithm can be more memory efficient than a centralized reasoner, since each package is only locally internalized. Therefore, once a new individual x is introduced in an ABox tree for package i, only $C_{\mathcal{T}_i}(x)$ is added to the ABox tree, while in the centralized case a more complex fact $C_{\mathcal{T}}(x)$ is added, where $C_{\mathcal{T}}$ is the internalization concept for the combined TBox $\mathcal{T}$ of all packages.

In conclusion, localizing reasoning subtasks within multiple local reasoners, reduces the time and space required by each local reasoner relative to that required by a centralized reasoner working on an integrated version of the modular ontology. Thus, the distributed reasoning algorithm can potentially make it possible to reason with ontologies that are much larger than those that can be accomodated by a single centralized reasoner.

5 Soundness and Completeness

The proposed algorithm reduces the problem of checking inconsistency of a model to a combination of checking inconsistency of local ABoxes. Given ABoxes $\mathcal{A}, \mathcal{B}, ...$, we denote by $(\mathcal{A}, \mathcal{B}, ..)$ the ABox obtained by merging the respective ABoxes, with shared facts and shared individual names are merged.

The $\mathcal{ALCP_C}$ expansion rules expand one or more existing ABoxes. We have the following lemma:

Lemma 4 *If a set of locally complete ABox* $\{\mathcal{A}_i\}$, $i = 1..., m$ *is generated by the* $\mathcal{ALCP_C}$ *tableau expansion rules,* $(\mathcal{A}_1, ..., \mathcal{A}_m)$ *is consistent iff* $\forall i, \mathcal{A}_i$ *is consistent.*

The preceding lemma follows from the observation that $\mathcal{ALCP_C}$ expansions will send any concept fact to the ABox of its destination package, and inconsistency is detected when both $C(x)$ and $\neg C(x)$ appear in some local ABox. Global inconsistency must necessarily result in a local consistency in some ABox; and a locally inconsistent ABox implies that the set of ABoxes taken together must also be inconsistent. Thus we have:

Lemma 5 (Soundness) *Assume that* S_0 *is obtained from the finite set of ABoxes* S *by application of an* $\mathcal{ALCP_C}$ *transformation rule. Then* S *is consistent iff* S_0 *is consistent.*

Completeness of the algorithm follows from the observation that a P-DL model can be induced by a set of distributed ABoxes. It is easy to verify that a P-DL model is obtained if we transform each ABox $\mathcal{A}_i$ into a local model (as we usually do to prove the completeness of $\mathcal{ALC}$ expansion rules), and if there is a fact message $C(x)$ sent from $\mathcal{A}_i$ to $\mathcal{A}_j$, add a pair $(i : x, j : x)$ to the image domain relation between Δ_i and Δ_j. Thus we have:

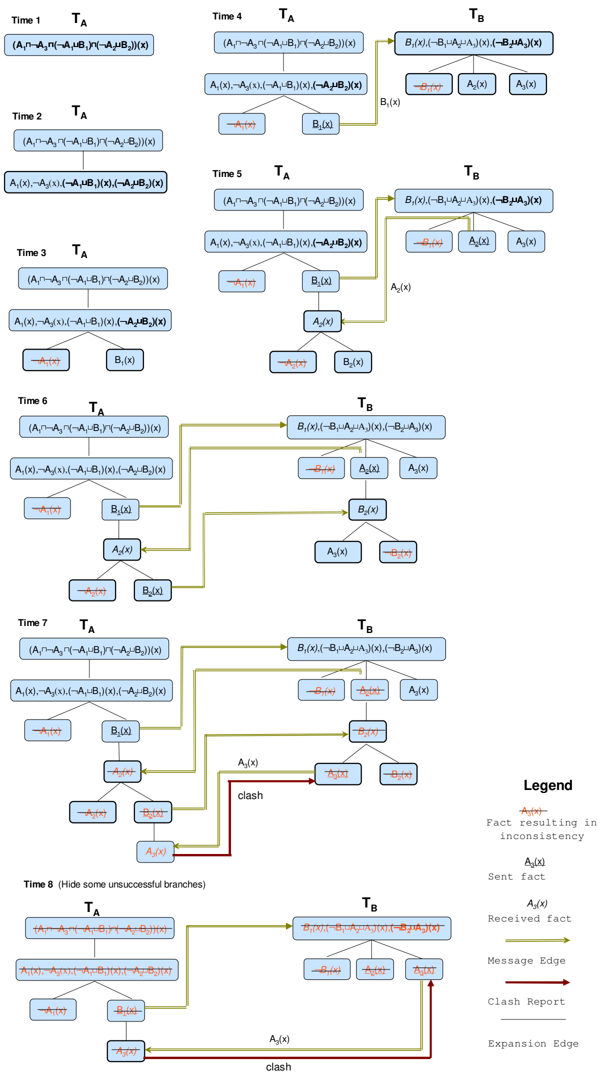

Figure 2. Reasoning with Mutual Importing

Lemma 6 (Completeness) *Any complete and clash-free $\mathcal{ALCP_C}$ Global ABox $\{\mathcal{A}_i\}$ has a model.*

6 Summary and Discussion

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

1. by strictly avoiding integration of modules into a single ontology, avoids the need for integrating the ontology modules in a centralized knowledge base.
2. by bsing a message-based inter-reasoner communication strategy, enhances the reusability of ontology modules, incliuding in particular, mutual or cyclic importing among packages thereby overcoming an important limitation of earlier approaches that were limited to acyclic importing [3].
3. by allowing reasoning to proceed in an asynchronous, peer-to-peer fashion, enables local reasoners to concurrently work on different reasoning subtasks thereby improving the efficiency and scalability of reasoning.
4. by exploiting on the P-DL formalism, tackles a broader range of reasoning tasks, including those involving both inter-module subsumption and role relations.

Work in progress is aimed at:

- Extending the proposed reasoning algorithm to work with more expressive DLs such as $\mathcal{SHIQP_C}$ (i.e. package-based $\mathcal{SHIQ}$ with concept importing).
- Detailed performance evaluation of implementations of the proposed algorithm in several practical application scenarios.

Acknowledgement: This research was supported by grants from the US NSF (0219699, 0639230) to Vasant Honavar.

References

[1] F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, and D. M. et.al., editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press, 2003.

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

[3] J. Bao, D. Caragea, and V. Honavar. A distributed tableau algorithm for package-based description logics. In *the 2nd International Workshop On Context Representation And Reasoning (CRR 2006), co-located with ECAI 2006*. 2006.

[4] J. Bao, D. Caragea, and V. Honavar. Modular ontologies - a formal investigation of semantics and expressivity. In *R. Mizoguchi, Z. Shi, and F. Giunchiglia (Eds.): Asian Semantic Web Conference 2006, LNCS 4185*, pages 616–631, 2006.

[5] 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)*, pages 99–108. IEEE Press, 2006.

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

[7] 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.

[8] 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.

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

[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.