The tableau algorithm for $\mathcal{ALC}$

Petr Křemen

version 0.2

The algorithm, introduced originally in [3] in slightly different form, aims at constructing a model of an $\mathcal{ALC}$ ontology $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$. During the algorithm run, each (possibly infinite) candidate model is represented by a (necessarily finite) completion graph.

A completion graph is a labeled oriented graph $G = \langle V_G, E_G, L_G \rangle$, where each $x \in V_G$ is labeled with a set $L_G(x)$ of concepts, and each edge $\langle x, y \rangle \in E_G$ is labeled with a set $L_G(\langle x, y \rangle)$ of roles. Furthermore, a completion graph $G$

- contains a direct clash, if $\{A, \neg A\} \subseteq L_G(x)$ for some named concept $A$, or $\bot \in L_G(x)$, or $\neg \top \in L_G(x)$

- is complete w.r.t. to the set $J$ of rules, if no completion rule from $J$ can be applied on it.

The tableau algorithm evolves a set $\mathcal{S}$ of completion graphs (corresponding to partial candidate models) according to completion rules in $J$. Whenever a rule application causes a clash in a completion graph, the graph is discarded (which prunes candidate models corresponding to the graph). The algorithm terminates when no more rules can be applied on a clash-free completion graph (ontology is consistent), or when no completion graph remains to explore (ontology is inconsistent).

1 Tableau algorithm for $\mathcal{ALC}$ with empty TBox

First focus on the case when TBox is empty, i.e. $\mathcal{T} = \emptyset$. In this case the set $J$ of applicable completion rules is depicted in Table 1. The procedure is as follows:

1. (PREPROCESSING) All concepts in $\mathcal{O}$ have to be transformed into Negational Normal Form. This means to “move negation in front of named concepts” using equivalences, like $\neg (C_1 \cap C_2) \equiv \neg C_1 \cup \neg C_2$, or $\neg (\exists R \cdot C) \equiv \forall R \cdot \neg C$.

2. (INITIALIZATION) Initial state of the algorithm is $\mathcal{S}_0 = \{G_0\}$, where $G_0 = \langle V_{G_0}, E_{G_0}, L_{G_0} \rangle$ is a completion graph, that “corresponds to $\mathcal{A}$”, i.e.

   - $V_{G_0}$ contains all named individuals occurring in some axiom of $\mathcal{A}$
Table 1: Completion rules used for expanding a set of ALC completion graphs (and not considering TBox). $G = (V_G, E_G, L_G)$ is the completion graph chosen in the current iteration.

The algorithm does not prescribe the order, in which the rules are selected. Of course, this can significantly influence performance. E.g. non-deterministic rules (⊔-rule in case of ALC) should be performed only when no other rule is applicable, to prevent generating additional completion graphs first, all of which need to be tested in CONSISTENCY/MODEL TEST steps of the algorithm. For details on other tableau optimization technique, see e.g. Chapter 9 in [1].

Correctness and Completeness I will not repeat the full proof of correctness and completeness of the algorithm, that was already presented in [1] and [3]. I only sketch main rationale behind the algorithm that helps to better understand its idea. Correctness is a direct consequence of the semantics of completion rules. E.g. if there was a model $I$ corresponding to $G$ and $A_1 \cap A_2 \in L_G(a)$ for some $a$, then, following the
semantics of ALC, \(a^T \in (A_1 \cap A_2)^T\) and \(a^T \in A_1^T \cap A_2^T\). This is ensured by putting both \(A_1\) and \(A_2\) into the \(L_G(a)\) by the \(\cap -\) rule. For the other direction and other rules the idea is similar.

Completeness is shown by constructing a canonical model \(I\) of \(O\) for each complete completion graph that does not contain a direct clash, as follows:

- The interpretation domain \(\Delta^I\) contains all graph vertices,
- for each named concept \(A\) we define \(A^I = \{a | A \in L_G(a)\}\),
- for each named role \(R\) we define \(R^I = \{\langle a_1, a_2 \rangle | R \in L_G(\langle a_1, a_2 \rangle)\}\).

Induction according to the axiom types and complex concept structure shows that it is indeed a model of the original ontology. E.g. if \(C(a) \in O\), then \(a^I \in C^I\) holds for any complete graph \(G\): (i) if \(C = A\) is a named concept, then indeed \(a^T \in A^T\) because \(A \in L_G(a)\) (this is, how \(G_0\) was constructed in the INITIALIZATION step of the algorithm) and \(L_G(a) \cup L_G(\langle a \rangle)\), (ii) if \(C = A_1 \cap A_2\) where both \(A_1\) and \(A_2\) are named concepts, then \(a^T \in A_1^T \cap A_2^T\) and thus \(a^T \in A_1^T\) and \(a^T \in A_2^T\) because \(\{A_1, A_2\} \subseteq L_G\) where \(G\) is a completion graph that resulted from application the \(\cap\)-rule. If this rule was not applied, then \(G \supseteq G'\) is not complete, which contradicts our assumption. For the other axiom types and concept constructs the idea is similar.

In case of ALC, there is a correspondence between a model and a complete completion graph simple, as presented. However, tableaux algorithms for expressive description logics require more complex structures and more complex transformations to achieve such correspondence, see e.g. [1] or [2] for more details.

Example 1 Let’s check consistency of an ALC ontology \(O = \{\alpha\}\), where \(\alpha\) is \(C(\text{PillarScour})\) and \(C\) is

\[
(\exists\text{isFailureOf} \cdot \text{Column} \cap \exists\text{isFailureOf} \cdot \text{Pillar} \cap \neg\exists\text{isFailureOf} \cdot (\text{Pillar} \cap \text{Column}))
\]

This axiom says that \(\text{PillarScour}\) is failure of some bridge and it is a failure of some pillar but it is not a failure of an object that is a bridge and a pillar at the same time.

The first step is to transform the complex concept into Negational Normal Form. This produces \(\alpha_2\) that is \(C_2(\text{PillarScour})\) where \(C_2\) is

\[
(\exists\text{isFailureOf} \cdot \text{Column} \cap \exists\text{isFailureOf} \cdot \text{Pillar} \cap \forall\text{isFailureOf} \cdot (\neg\text{Pillar} \cap \neg\text{Column}))
\]

\(\alpha_2\) is semantically equivalent with \(\alpha\) (i.e. \(\{\alpha\} \models \{\alpha_2\}\) and \(\{\alpha_2\} \models \{\alpha\}\)), but it has negations only before named concepts.

The initial state of the algorithm is \(S_0 = \{G_0\}\), where graph \(G_0 = \langle\{\text{PillarScour}\}, \emptyset, \{\text{PillarScour} \mapsto \{C_2\}\}\rangle\) (1)

is shown in Figure [1].

At this point, the algorithm passes four sequences of the steps CONSISTENCY TEST \(\rightarrow\) MODEL TEST \(\rightarrow\) RULE APPLICATION of the tableau algorithm, that

3
can be denoted as a state evolution during the step RULE APPLICATION (the label over the arrow denotes the rule that was used):

\[
\begin{align*}
\{G_0\} & \xrightarrow{\cap\text{-rule}} \{G_1\} \xrightarrow{\exists\text{-rule}} \{G_2\} \xrightarrow{\exists\text{-rule}} \{G_3\} \xrightarrow{\forall\text{-rule}} \{G_4\},
\end{align*}
\]

where \(G_4\) is depicted in Figure 2.

So far, only deterministic rules (i.e., those that do not increase the number of completion graphs) have been used. Looking carefully at Figure 2, it is clear that the only rule that remains applicable is the \(\sqcup\text{-rule}\). The rule can be applied on the concept \((\neg\text{Column} \sqcup \neg\text{Pillar})\) in the label of either vertex 0 or 1. Picking e.g. 0 and applying \(\sqcup\text{-rule}\) produces a new state \(\{G_5, G_6\}\) depicted in Figure 3.

Graph \(G_5\) contains a direct clash, as \(\text{Column}\) and \(\neg\text{Column}\) is in the label of vertex 0, and thus \(G_5\) is discarded, as it cannot be transformed to a model (e.g., the canonical model defined in relation to the completeness of the algorithm). Thus, \(G_6\) is picked and \(\sqcup\text{-rule}\) is applied, which results in a new state \(\{G_7, G_8\}\), as shown in Figure 4.

While \(G_7\) contains a direct clash in vertex 1, completion graph \(G_8\) is complete with respect to the \(\mathcal{ALC}\) completion rules and does not contain a direct clash. Thus, a canonical model \(I_1 = (\Delta I_1, \cdot I_1)\) can be constructed from \(G_8\) as follows:
Figure 3: Graphs $G_5$ and $G_6$ produced by application of the $\sqcup$-rule on vertex 0.

$$\Delta^{I_1} = \{\text{PillarScour}, 0, 1\}$$

$$\text{isFailureOf}^{I_1} = \{\{\text{PillarScour}, 0\}, \{\text{PillarScour}, 1\}\}$$

$$\text{Pillar}^{I_1} = \{1\}$$

$$\text{Column}^{I_1} = \{0\}$$

$$\text{PillarScour}^{I_1} = \{\text{PillarScour}\}$$

$I_1$ is not the only interpretation of $O$ - another model of $O$ might be $I_2$, for which $\text{Column} = \{0, \text{PillarScour}\}$ and coincides with $I_1$ on the rest. This documents the fact that a complete completion graph corresponds to one (canonical) model, but might
Figure 4: Graphs $G_7$ and $G_8$ produced by application of the $\sqsubseteq$-rule on vertex 1.

correspond to other models as well.

2 Tableau algorithm for general $\mathcal{ALC}$

In the general case, the situation is slightly complicated. The TBox knowledge is included in the algorithm by an extra rule

\begin{align*}
\sqsubseteq\text{-rule} & \\
\text{if: } (C_1 \sqsubseteq C_2) \in T \text{ and } (\neg C_1 \sqcup C_2) \notin L_G(a) \text{ for some } a \text{ that is not blocked.} & \\
\text{then: } S' = S \cup \{G'\} \setminus \{G\}, \text{ where } G' = (V_G, E_G, L_{G'}). & \\
L_{G'} = L_G \text{ except } L_{G'}(a) = L_G(a) \cup \{\neg C_1 \sqcup C_2\}. &
\end{align*}
For this rule to be applicable, each equivalence axiom \( C_1 \equiv C_2 \in \mathcal{T} \) has to be replaced by two subsumption axioms \( C_1 \subseteq C_2 \) and \( C_2 \subseteq C_1 \) in the PREPROCESSING step of the algorithm. Unfortunately, such simple extension of the tableau algorithm need not terminate. E.g. consider ontology \( \mathcal{O} = \{ \text{Object} \sqsubseteq \exists\text{hasPart} . \text{Object, Object(CharlesBridge)} \} \).

### 2.1 Blocking

To ensure termination of the algorithm it is necessary to detect cycles of the generated model that might occur due to the application of the \( \subseteq \)-rule. The cycles are detected using *blocking* that ensures that a completion graph, although representing possibly infinite model, is always finite. This is ensured by preventing the inference rules to generate node and edge patterns that “repeat regularly”. The notion of regularity is different for each description logic; for \( \mathcal{ALC} \) so called subset blocking \(^1\) is used:

A vertex \( a_1 \) in a completion graph \( G \), but not occuring in \( \mathcal{A} \), is blocked by a vertex \( a_2 \), if there is an oriented path in \( G \) from \( a_2 \) to \( a_1 \) and a \( L_G(a_1) \subseteq L_G(a_2) \).

Then all rules are applicable only on an individual, only if this individual is not blocked. As a result we get a set of completion rules in \( \mathcal{ALC} \) with general TBox, as shown in Table 2.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \subseteq )-rule</td>
<td>if ( (C_1 \subseteq C_2) \in \mathcal{T} ) and ( \neg C_1 \sqcap C_2 \notin L_G(a) ) for some ( a ) that is not blocked.</td>
<td>then ( S' = S \cup { G' } \setminus { G } ), where ( G' = \langle V_G, E_G, L_G' \rangle ). ( L_G' = L_G ) except ( L_G'(a) = L_G(a) \cup { \neg C_1 \sqcap C_2 } ).</td>
</tr>
<tr>
<td>( \sqcap )-rule</td>
<td>if ( (C_1 \sqcap C_2) \in L_G(a) ), for some ( a ) that is not blocked and ( { C_1, C_2 } \notin L_G(a) ).</td>
<td>then ( S' = S \cup { G' } \setminus { G } ), where ( G' = \langle V_G, E_G, L_G' \rangle ). ( L_G' = L_G ) except ( L_G'(a) = L_G(a) \cup { C_1, C_2 } ).</td>
</tr>
<tr>
<td>( \sqcup )-rule</td>
<td>if ( (C_1 \sqcup C_2) \in L_G(a) ), for some ( a ) that is not blocked and ( { C_1, C_2 } \cap L_G(a) = \emptyset ).</td>
<td>then ( S' = S \cup { G_1, G_2 } \setminus { G } ), where ( G_k = \langle V_G, E_G, L_G \rangle ). ( L_G' = L_G ) except ( L_G'(a) = L_G(a) \cup { C_1, C_2 } ) for ( k \in { 1, 2 } ).</td>
</tr>
<tr>
<td>( \exists )-rule</td>
<td>if ( \exists R : C \in L_G(a_1) ), for some ( a_1 ) that is not blocked, and there is no ( a_2 \in V_G ), such that both ( R \in L_G(a_1, a_2) ) and ( C \in L_G(a_2) ).</td>
<td>then ( S' = S \cup { G' } \setminus { G } ), where ( G' = \langle V_G \cup { a_2 }, E_G \cup { (a_1, a_2) }, L_G \rangle ). ( L_G' = L_G ) except ( L_G'(a_2) = { C } ), ( L_G'(a_1, a_2) = { R } ).</td>
</tr>
<tr>
<td>( \forall )-rule</td>
<td>if ( \forall R : C \in L_G(a_1) ), for some ( a_1 ) that is not blocked and there is ( a_2 \in V_G ), such that ( R \in L_G(a_1, a_2) ), but not ( C \in L_G(a_2) ).</td>
<td>then ( S' = S \cup { G' } \setminus { G } ), where ( G' = \langle V_G, E_G, L_G' \rangle ). ( L_G' = L_G ) except ( L_G'(a_2) = L_G(a_2) \cup { C } ).</td>
</tr>
</tbody>
</table>

Table 2: Completion rules used for expanding a set of \( \mathcal{ALC} \) completion graphs. \( G = (V_G, E_G, L_G) \) is the completion graph chosen in the current iteration.
References

