Tableau Proof Techniques for DLs

Reasoning Services for DLs

- Terminating, complete and efficient algorithms for deciding **satisfiability** – and all the other reasoning services – are available for various DLs.
- Most of these algorithms are based on **tableau proof techniques**.
- Such algorithms have been shown to be **efficient** for real knowledge bases, even if the problem in the corresponding logic is in PSPACE or EXPTIME. Most popular DL reasoners today are based on tableau techniques e.g., FACT++ (`http://owl.man.ac.uk/factplusplus/`) and Pellet (`http://clarkparsia.com/pellet/`).

Tableau Proof Techniques

We will give a short introduction of tableau proof techniques for

- Propositional logic (PL)
- First-order logic (FOL)

before we move to the case of description logics.

What we want to demonstrate is that tableau techniques have been standard proof techniques in other logics before they were used by DL researchers. Regarding DLs, there are also close connections to tableau techniques for modal logics but we will not introduce them here in any detail.

In the literature, the term **semantic tableau** is also used.

Tableau Proof Techniques - PL

Tableau are **refutation systems** for PL (like **resolution**). To prove that a formula P is a **tautology** (or **valid**), we start with $\neg P$ and produce a **contradiction**.

The procedure for doing this involves **expanding** $\neg P$ so that inessential details of its logical structure are cleared away.

In tableau proofs, such an expansion takes the form of a **tree**, where nodes are labeled with formulas.

Each **branch** of this tree should be thought of as representing the **conjunction** of the formulas appearing on it, and the tree itself as representing the **disjunction** of its branches.

Uniform Notation for PL

Theorem. (Unique Parsing) Every propositional formula is in exactly one of the following categories:

1. atomic (propositional symbol, $\top$ or $\bot$).
2. $\neg X$, for a unique propositional formula X.
3. $(X \circ Y)$ for a unique binary symbol $\circ$ and unique propositional formulas X and Y.

Uniform Notation for PL (cont'd)

Based on the unique parsing theorem, we can group all propositional formulas of the forms $(X \circ Y)$ and $\neg(X \circ Y)$ into two categories, those that act **conjunctively**, which we call **α-formulas**, and those that act **disjunctively**, which we call **β-formulas**:

α	α_1	α_2
$X \wedge Y$	X	Y
$\neg(X \vee Y)$	$\neg X$	$\neg Y$
$\neg(X \supset Y)$	X	$\neg Y$

β	β_1	β_2
$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$X \vee Y$	X	Y
$X \supset Y$	$\neg X$	Y

Uniform notation allows us to have a large number of basic connectives, and still not do unnecessary work in proofs.

Tableau Expansion Rules

The following **tableau expansion rules** are used to manipulate trees (transform a tree into another) in tableau proofs:

$$\frac{\neg\neg P}{P} \qquad \frac{\neg\top}{\bot} \qquad \frac{\neg\bot}{\top} \qquad \frac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\beta_1 \mid \beta_2}$$

How are these rules used?

Example

Let us assume that we want to show that the formula

$$(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S)).$$

is a tautology. The following tree is a tableau proof of this formula.

Example (cont'd)

$$\neg[(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))]$$

$$P \supset (Q \supset R)$$

$$\neg((P \vee S) \supset ((Q \supset R) \vee S))$$

$$P \vee S$$

$$\neg((Q \supset R) \vee S)$$

$$\neg(Q \supset R)$$

$$\neg S$$

$$\neg P \qquad\qquad Q \supset R$$

$$P \qquad S$$

Closed Tableau

A branch θ of a tableau is called **closed** if both X and $\neg X$ occur on θ for some propositional formula X, or if $\perp$ occurs on θ.

If A and $\neg A$ occur on θ where A is atomic, or if $\perp$ occurs, θ is said to be **atomically closed**.

A tableau is **(atomically) closed** if every branch is (atomically) closed.

A **tableau proof** of X is a closed tableau for $\{\neg X\}$.

Soundness and Completeness

Theorem. (Soundness) If a sentence ϕ of PL has a tableaux proof then ϕ is a tautology.

Theorem. (Completeness) If a sentence ϕ of PL is a tautology then ϕ has a tableau proof.

The Example Revisited

$$\neg[(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))]$$

$$P \supset (Q \supset R)$$

$$\neg((P \vee S) \supset ((Q \supset R) \vee S))$$

$$P \vee S$$

$$\neg((Q \supset R) \vee S)$$

$$\neg(Q \supset R)$$

$$\neg S$$

$$\neg P \qquad\qquad Q \supset R$$

$$P \qquad S$$

Example (cont'd)

Notice that all branches of the tableau in this example are **closed**. In one of the branches, closure was on a non-atomic formula.

Thus, the tableau is **closed** and the given formula

$$(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$$

is a **tautology**.

Uniform Notation for FOL

The uniform notation we introduced for PL can be extended to FOL. The additional machinery is that all **quantified** formulas and their negations are grouped into two categories, those that act **universally**, which are called γ**-formulas**, and those that act **existentially**, which are called δ**-formulas**. For each variety and for each term t, an **instance** is defined.

γ	$\gamma(t)$
$(\forall x)\Phi$	$\Phi\{x/t\}$
$\neg(\exists x)\Phi$	$\neg\Phi\{x/t\}$

δ	$\delta(t)$
$(\exists x)\Phi$	$\Phi\{x/t\}$
$\neg(\forall x)\Phi$	$\neg\Phi\{x/t\}$

Tableau Proofs for FOL

In informal proofs, **new constant symbols** are routinely used.

The formal counterpart is **parameters**, constant symbols not part of our original language.

In tableau proofs, we will use sentences of L^{par}, the extension of the given language L by the addition of a countable list of **new parameters**.

Tableau Expansion Rules for FOL

In the case of FOL, we have the PL **tableau expansion rules** plus the following two:

$$\frac{\gamma}{\gamma(t)}$$

(for any closed term t of L^{par})

$$\frac{\delta}{\delta(p)}$$

(for a **new** parameter p of L^{par})

Example

Let us assume that we want to prove that the FOL formula

$$(\forall x)(P(x) \vee Q(x)) \supset ((\exists x)P(x) \vee (\forall x)Q(x))$$

is valid. The following tree is a tableau proof of this formula. The resulting tableau is closed.

Example (cont'd)

$$\neg((\forall x)(P(x) \vee Q(x)) \supset ((\exists x)P(x) \vee (\forall x)Q(x)))$$
$$(\forall x)(P(x) \vee Q(x))$$
$$\neg((\exists x)P(x) \vee (\forall x)Q(x))$$
$$\neg(\exists x)P(x)$$
$$\neg(\forall x)Q(x)$$
$$\neg Q(c)$$
$$\neg P(c)$$
$$P(c) \vee Q(c)$$

$$P(c) \qquad\qquad Q(c)$$

Soundness and Completeness

Theorem. (Soundness) If a FOL sentence ϕ has a tableau proof then ϕ is valid.

Theorem. (Completeness) If a sentence ϕ of FOL is valid, then ϕ has a tableau proof.

Because FOL is not decidable, tableau proofs may not always terminate. The source of this difficulty is the γ rule.

Trivial example: Suppose we have a tableau branch containing both $(\exists x)\neg P(x)$ and $(\forall y)P(y)$. We might apply the δ-rule to the first formula, adding $\neg P(c)$, where c is a new parameter. But then using the γ-rule on the second, we might add one after the other $P(t_1), P(t_2), \ldots$ where $t_1, t_2, \ldots$ are all distinct closed terms different from c. In this way, we never produce the obvious closure.

Deciding Satisfiability in DL Using Tableau

Tableau proofs are decision procedures for solving the problem of satisfiability in a DL.

If a formula is **satisfiable**, the procedure will constructively exhibit a **model** of the formula.

The basic idea (as in PL and FOL) is to incrementally build such a model by looking at the formula and decomposing it in a top/down fashion. The procedure exhaustively looks at all the possibilities.

If a formula is **unsatisfiable**, the procedure can eventually prove that **no model** could be found.

$\mathcal{ALC}$ - Revision

Syntax	Semantics	Terminology
A	$A^{\mathcal{I}} \subseteq \Delta$	atomic concept
R	$R^{\mathcal{I}} \subseteq \Delta \times \Delta$	atomic role
$\top$	Δ	top (universal) concept
$\bot$	$\emptyset$	bottom concept
$\neg C$	$\Delta \setminus C^{\mathcal{I}}$	concept complement
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	concept conjunction
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	concept disjunction
$\forall R.C$	$\{x \mid (\forall y)((x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}})\}$	universal restriction
$\exists R.C$	$\{x \mid (\exists y)((x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$	existential restriction

$\mathcal{ALC}$ - Examples

- $\texttt{Person} \sqcap \neg\texttt{Female}$
- $\texttt{Female} \sqcup \texttt{Male}$
- $\forall\texttt{child.Person}$
- $\exists\texttt{child.Person}$
- $\exists\texttt{child.Person} \sqcap \forall\texttt{child.Person}$
- $(\texttt{Female} \sqcap \forall\texttt{child.Person})(\texttt{ANNA})$
- $\texttt{child}(\texttt{BOB}, \texttt{ANNA})$

Tableau Proofs for $\mathcal{ALC}$ Concept Satisfiability

Given an $\mathcal{ALC}$ concept C, the tableau algorithm for **concept satisfiability** tries to construct a finite interpretation $\mathcal{I}$ that satisfies C i.e., it contains an element a such that $a \in C^{\mathcal{I}}$.

We follow the paper by Baader and Sattler (2001) given in the readings, and use an ABox assertion $C(a)$ to encode this.

In some papers of the literature, a **constraint system** is used to implement the tableau (the two approaches are equivalent).

Tableau Proofs: the High-Level Algorithm

1. We start with the ABox assertion $C(a)$.
2. We add formulas to the tableau by applying certain **transformation rules**. Transformation rules are either **deterministic** or **nondeterministic** (result in **branches**).
3. We apply the transformation rules until either a **contradiction** is generated in **every branch**, or **there is a branch where no more rule is applicable**.

   In the former case C is unsatisfiable. In the latter case, C is satisfiable and this branch gives a **non-empty model** of C.

Negation Normal Form

For the tableau techniques to work, the formula in question has to be transformed into negation normal form.

Definition. A formula is in **negation normal form** if negation appears only in front of atomic concepts.

Applying the following equivalences, we can transform any $\mathcal{ALC}$ formula into an equivalent one in negation normal form:

- $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$
- $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$
- $\neg(\forall R.C) \equiv \exists R.\neg C$
- $\neg(\exists R.C) \equiv \forall R.\neg C$

Transformation Rules: the AND rule

The transformation rules come straightforwardly from the semantics of constructors.

If in an **arbitrary interpretation** $\mathcal{I}$, whose domain contains an **arbitrary** element a, we have that $a \in (C \sqcap D)^{\mathcal{I}}$, then from the semantics we know that a should be in the intersection of $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$, i.e. it should be in both $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$.

We can use **ABox assertions** to encode this in a transformation rule as follows.

The AND rule (or $\rightarrow_{\sqcap}$-rule)

If

- $(C \sqcap D)(a)$ is in $\mathcal{A}$, but
- $C(a)$ and $D(a)$ are not both in $\mathcal{A}$

then

$$\mathcal{A} := \mathcal{A} \cup \{C(a), D(a)\}$$

The OR Rule (or $\rightarrow_{\sqcup}$-rule)

Similarly, we have the following rule.

If

- $(C \sqcup D)(a)$ is in $\mathcal{A}$, but
- neither $C(a)$ nor $D(a)$ is in $\mathcal{A}$

then

$$\mathcal{A} := \mathcal{A} \cup \{C(a)\}$$

or

$$\mathcal{A} := \mathcal{A} \cup \{D(a)\}$$

This rule forces us to introduce **sets of ABoxes** as a formal tool to represent tableau proofs.

The SOME rule (or $\rightarrow_{\exists}$-rule)

From the semantics, we have the following. If in an arbitrary interpretation $\mathcal{I}$, whose domain contains an arbitrary element a, we have that $a \in (\exists R.C)^{\mathcal{I}}$, then there must be an element b (not necessarily distinct from a) such that $(a, b) \in R^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}$.

We can use **ABox assertions** to encode this in a transformation rule as follows.

The SOME rule (cont'd)

If

- $(\exists R.C)(a)$ is in $\mathcal{A}$ and
- there is no individual c such that both $R(a, c)$ and $C(c)$ are in $\mathcal{A}$

then

$$\mathcal{A} := \mathcal{A} \cup \{R(a, b), C(b)\}$$

where b is an a **new individual** not occurring in $\mathcal{A}$.

The FORALL Rule (or $\rightarrow_\forall$-rule)

Similarly, we have the following rule.

If

- $(\forall R.C)(a)$ is in $\mathcal{A}$
- $R(a, b)$ is in $\mathcal{A}$, and
- $C(b)$ is not in $\mathcal{A}$

then

$$\mathcal{A} := \mathcal{A} \cup \{C(b)\}$$

Some Definitions

Definition. An ABox is called **complete** if none of the above transformation rules applies to it.

While building a tableau proof, we can look for evident **contradictions** to see if the tableau is not satisfiable. We call these contradictions **clashes**.

Definition. An ABox $\mathcal{A}$ contains a **clash** if

- $\{\bot(a)\} \subseteq \mathcal{A}$, or
- $\{C(a), (\neg C)(a)\} \subseteq \mathcal{A}$

for some individual a and concept C.

Definition. An ABox is called **closed** if it contains a clash, and **open** otherwise.

Note: ABoxes correspond to **branches** in a tableau so the definitions can be given for tableaux too.

Tableau Proofs: the Algorithm Revisited

1. We start with the ABox assertion $C(a)$.
2. We add formulas to the tableau by applying the previous rules.
3. We apply the rules until either a **contradiction** is generated in **every branch** (all branches are closed ABoxes), or **there is a branch where no contradiction appears and no rule is applicable** (this branch is an open and complete ABox).

   In the former case C is **unsatisfiable**. In the latter case, C is **satisfiable** and this branch gives a **non-empty model** of C.

Examples

Let us check whether the concept

$$(\forall \texttt{child}.\texttt{Male}) \sqcap (\exists \texttt{child}.\neg\texttt{Male})$$

is satisfiable. The tableau method will proceed as follows:

$\Big((\forall \texttt{child}.\texttt{Male}) \sqcap (\exists \texttt{child}.\neg\texttt{Male})\Big)(\texttt{a})$

$(\forall \texttt{child}.\texttt{Male})(\texttt{a})$ $\sqcap$-*rule*

$(\exists \texttt{child}.\neg\texttt{Male})(\texttt{a})$

$\texttt{child}(\texttt{a},\texttt{b})$ $\exists$-*rule*

$\neg\texttt{Male}(\texttt{b})$

$\texttt{Male}(\texttt{b})$ $\forall$-*rule*

Clash!

The tableau is complete and there is a contradiction in every branch, thus the given concept is **unsatisfiable.**

Examples (cont'd)

Let us check whether the concept

$$(\forall \texttt{child}.\texttt{Male}) \sqcap (\exists \texttt{child}.\texttt{Male})$$

is satisfiable. The tableau method will proceed as follows:

$\Big((\forall \texttt{child}.\texttt{Male}) \sqcap (\exists \texttt{child}.\texttt{Male})\Big)(\texttt{a})$

$(\forall \texttt{child}.\texttt{Male})(\texttt{a})$ $\sqcap$-*rule*

$(\exists \texttt{child}.\texttt{Male})(\texttt{a})$

$\texttt{child}(\texttt{a}, \texttt{b})$ $\exists$-*rule*

$\texttt{Male}(\texttt{b})$

$\texttt{Male}(\texttt{b})$ $\forall$-*rule*

The above tableau with one branch (ABox) is complete and open, thus the given formula is **satisfiable.** Can you find a model?

Satisfiability of ABoxes

Naturally, we can also check the satisfiability of ABoxes using tableau techniques.

Example: Consider the ABox consisting of the following formulas:

$$(\texttt{Parent} \sqcap \forall \texttt{child}.\texttt{Male})(\texttt{JOHN})$$

$$\neg \texttt{Male}(\texttt{MARY})$$

$$\texttt{child}(\texttt{JOHN}, \texttt{MARY})$$

Satisfiability of ABoxes (cont'd)

The tableau technique in this case gives us:

$$
\begin{array}{rll}
(\texttt{Parent} \sqcap \forall \texttt{child}.\texttt{Male})(\texttt{JOHN}) & & \\
(\neg \texttt{Male})(\texttt{MARY}) & & \\
\texttt{child}(\texttt{JOHN}, \texttt{MARY}) & & \\
\texttt{Parent}(\texttt{JOHN}) & & \sqcap\textit{-rule} \\
(\forall \texttt{child}.\texttt{Male})(\texttt{JOHN}) & & \\
\texttt{Male}(\texttt{MARY}) & & \forall\textit{-rule} \\
\textbf{Clash!} & &
\end{array}
$$

Thus the ABox is **unsatisfiable.**

Soundness of the Tableau Method for $\mathcal{ALC}$

The tableau method does not add unnecessary contradictions.

Deterministic rules always **preserve the satisfiability** of any ABox involved in the proof, and nondeterministic rules allow always a choice of application that preserves satisfiability.

Termination of the Tableau Method for $\mathcal{ALC}$

Termination can be proved by using the following argument.

All rules but $\rightarrow_\forall$ are never applied twice on the same ABox assertion.

The $\rightarrow_\forall$ rule is never applied to an individual a more times than the number of the **direct successors** of a, which is bounded by the length of a concept (for a definition of the concept of direct successors see the formal proof).

Finally, each rule application to a formula $C(a)$ adds formulas $D(b)$ such that D is a strict subexpression of C.

Completeness of the Tableau Method for $\mathcal{ALC}$

If $\mathcal{A}$ is a **complete** and **open** ABox (i.e., a branch) in a tableau proof of $C(a)$ then $\mathcal{A}$ is **satisfiable**.

The following is a **canonical interpretation** $\mathcal{I}$ of $\mathcal{A}$ that can be obtained from the tableau:

- The domain $\Delta^\mathcal{I}$ of $\mathcal{I}$ consists of the individuals occurring in $\mathcal{A}$.
- For each atomic concept P, we define $P^\mathcal{I}$ to be $\{x|\ P(x) \in \mathcal{A}\}$.
- For each atomic role R, we define $R^\mathcal{I}$ to be $\{(x,y)|\ R(x,y) \in \mathcal{A}\}$.

Using this interpretation, it is possible to construct an interpretation for C such that $C^\mathcal{I}$ is nonempty. In other words, C is satisfiable.

Example

Let us revisit the complete and open Abox we used to prove that the concept

$$(\forall \texttt{child}.\texttt{Male}) \sqcap (\exists \texttt{child}.\texttt{Male})$$

is satisfiable:

$\Big((\forall \texttt{child}.\texttt{Male}) \sqcap (\exists \texttt{child}.\texttt{Male})\Big)(\texttt{a})$

$(\forall \texttt{child}.\texttt{Male})(\texttt{a})$ $\sqcap$-*rule*

$(\exists \texttt{child}.\texttt{Male})(\texttt{a})$

$\texttt{child}(\texttt{a},\texttt{b})$ $\exists$-*rule*

$\texttt{Male}(\texttt{b})$

$\texttt{Male}(\texttt{b})$ $\forall$-*rule*

Example (cont'd)

A model $\mathcal{I}$ of the Abox has domain $\Delta(\mathcal{I}) = \{\texttt{a}, \texttt{b}\}$ and

$$\texttt{Male}^{\mathcal{I}} = \{\texttt{b}\}, \quad \texttt{child}^{\mathcal{I}} = \{(\texttt{a}, \texttt{b})\}.$$

The concept

$$(\forall \texttt{child}.\texttt{Male}) \sqcap (\exists \texttt{child}.\texttt{Male})$$

is satisfiable because

$$((\forall \texttt{child}.\texttt{Male}) \sqcap (\exists \texttt{child}.\texttt{Male}))^{\mathcal{I}} = \{\texttt{a}\} \neq \emptyset$$

Tableau Techniques for TBoxes

So far we have used tableau techniques to deal with **concept satisfiability** and **ABox satisfiability**.

The literature gives us tableaux techniques for dealing with TBoxes as well.

The easiest case is when we have **acyclic terminologies.**

Acyclic Terminologies

A TBox is called an **acyclic terminology** if it is a set of concept definitions that do not contain multiple or cyclic definitions.

Multiple definitions are terminological axioms of the form

$$A \equiv B_1, \ldots, A \equiv B_n$$

for distinct concept expressions $B_1, \ldots, B_n$.

Cyclic definitions are terminological axioms of the form

$$A_1 \equiv C_1, \ldots, A_n \equiv C_n$$

where A_i occurs in C_{i-1} $(1 < i \leq n)$ and A_1 occurs in C_n.

If the acyclic terminology $\mathcal{T}$ contains a concept definition $A \equiv C$ then A is called its **defined name** and C its **defining concept**.

Acyclic Terminologies (cont'd)

Reasoning with acyclic terminologies can be reduced to **reasoning without TBoxes** by unfolding the definitions: this is achieved by repeatedly replacing defined names by their defining concepts until no more defined names exist.

Unfolding might lead to an **exponential blow-up** in the size of the produced ABox.

Example

TBox:

$$\texttt{MixedTeam} \equiv \texttt{Team} \sqcap (\exists \texttt{hasMember.Male}) \sqcap (\exists \texttt{hasMember.Female})$$

$$\texttt{Male} \equiv \neg \texttt{Female}$$

ABox:

$$\texttt{MixedTeam}(\texttt{FC})$$

$$(\forall \texttt{hasMember.Male})(\texttt{FC})$$

The above knowledge base is unsatisfiable.

Example (cont'd)

After unfolding the definition of `MixedTeam`, we have:

$$(\texttt{Team} \sqcap (\exists \texttt{hasMember.Male}) \sqcap (\exists \texttt{hasMember.Female}))(\texttt{FC})$$

$$(\forall \texttt{hasMember.Male})(\texttt{FC})$$

After unfolding the definition of `Male`, we have:

$$(\texttt{Team} \sqcap (\exists \texttt{hasMember.}(\neg \texttt{Female})) \sqcap (\exists \texttt{hasMember.Female}))(\texttt{FC})$$

$$(\forall \texttt{hasMember.}(\neg \texttt{Female}))(\texttt{FC})$$

Example (cont'd)

The closed tableau showing unsatisfiability is as follows:

$$(\texttt{Team} \sqcap (\exists \texttt{hasMember.}(\neg \texttt{Female})) \sqcap (\exists \texttt{hasMember.Female}))(\texttt{FC})$$

$$(\forall \texttt{hasMember.}(\neg \texttt{Female}))(\texttt{FC})$$

$$(\texttt{Team} \sqcap (\exists \texttt{hasMember.}(\neg \texttt{Female})))(\texttt{FC})$$

$$((\exists \texttt{hasMember.Female}))(\texttt{FC})$$

$$\texttt{hasMember}(\texttt{FC}, \texttt{a})$$

$$\texttt{Female}(\texttt{a})$$

$$\neg \texttt{Female}(\texttt{a})$$

Clash!

General TBoxes

For tableau proofs involving general TBoxes, see the paper by Baader and Sattler (2001) in the readings and references therein.

Readings

- F. Baader. Description Logics. In Reasoning Web: Semantic Technologies for Information Systems, 5th International Summer School 2009, volume 5689 of Lecture Notes in Computer Science, pages 1-39. Springer-Verlag, 2009.
  Available from
  `http://lat.inf.tu-dresden.de/research/papers.html`.
- F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. Studia Logica, 69:5-40, 2001.
  Available from
  `http://www.cs.man.ac.uk/~sattler/ulis-ps.html`.
- (Optional). Melvin Fitting. First-Order Logic and Automated Theorem Proving. 2nd edition. Springer, 1996.
  This is a good introduction to tableau proofs for PL and FOL.

Acknowledgements

The slides on tableau techniques for DLs have been prepared by modifying slides by Enrico Franconi, University of Bolzano-Bozen, Italy.

See `http://www.inf.unibz.it/~franconi/dl/course/` for Enrico's course on DLs.

Some other courses on DLs are listed on `http://dl.kr.org/courses.html`.