Overview of the Tutorial

- **History and Basics**: Syntax, Semantics, ABoxes, Tboxes, Inference Problems and their interrelationship, and Relationship with other (logical) formalisms
- **Applications of DLs**: ER-diagrams with i.com demo, ontologies, etc. including system demonstration
- **Reasoning Procedures**: simple tableaux and why they work
- **Reasoning Procedures II**: more complex tableaux, non-standard inference problems
- **Complexity issues**
- **Implementing/Optimising DL systems**
Description Logics

- family of logic-based knowledge representation formalisms well-suited for the representation of and reasoning about
  - terminological knowledge
  - configurations
  - ontologies
  - database schemata
    - schema design, evolution, and query optimisation
    - source integration in heterogeneous databases/data warehouses
    - conceptual modelling of multidimensional aggregation
  - ...

- descendents of semantics networks, frame-based systems, and KL-ONE
- aka terminological KR systems, concept languages, etc.

Architecture of a Standard DL System
**Introduction to DL I**

A Description Logic - mainly characterised by a set of constructors that allow to build complex concepts and roles from atomic ones,

- concepts correspond to classes / are interpreted as sets of objects,
- roles correspond to relations / are interpreted as binary relations on objects,

Example: Happy Father in the DL $\mathcal{ALC}$

\[ \text{Man} \sqcap (\exists \text{has-child. Blue}) \sqcap \]
\[ (\exists \text{has-child. Green}) \sqcap \]
\[ (\forall \text{has-child. Happy} \sqsubseteq \text{Rich}) \]

**Introduction to DL: Syntax and Semantics of $\mathcal{ALC}$**

Semantics given by means of an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$:

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept</td>
<td>$A$</td>
<td>Human</td>
<td>$A^\mathcal{I} \subseteq \Delta^\mathcal{I}$</td>
</tr>
<tr>
<td>atomic role</td>
<td>$R$</td>
<td>likes</td>
<td>$R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$</td>
</tr>
</tbody>
</table>

For $C$, $D$ concepts and $R$ a role name

- conjunction: $C \sqcap D$  Human $\sqcap$ Male  $C^\mathcal{I} \cap D^\mathcal{I}$
- disjunction: $C \sqcup D$  Nice $\sqcup$ Rich  $C^\mathcal{I} \cup D^\mathcal{I}$
- negation: $\neg C$  $\neg$ Meat  $\Delta^\mathcal{I} \setminus C^\mathcal{I}$
- exists restrict: $\exists R.C$  $\exists$has-child.Human  \[ \{ x \mid \exists y. \langle x, y \rangle \in R^\mathcal{I} \land y \in C^\mathcal{I} \} \]
- value restrict: $\forall R.C$  $\forall$has-child.Blond  \[ \{ x \mid \forall y. \langle x, y \rangle \in R^\mathcal{I} \Rightarrow y \in C^\mathcal{I} \} \]
**Introduction to DL: Other DL Constructors**

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>number restriction</td>
<td>$(\geq n , R)$</td>
<td>$(\geq 7 \text{ has-child})$</td>
<td>${ x \mid { y. \langle x, y \rangle \in R^I } \geq n }$</td>
</tr>
<tr>
<td></td>
<td>$(\leq n , R)$</td>
<td>$(\leq 1 \text{ has-mother})$</td>
<td>${ x \mid { y. \langle x, y \rangle \in R^I } \leq n }$</td>
</tr>
<tr>
<td>inverse role</td>
<td>$R^-$</td>
<td>has-child^-</td>
<td>${ (x, y) \mid \langle y, x \rangle \in R^I }$</td>
</tr>
<tr>
<td>trans. role</td>
<td>$R^*$</td>
<td>has-child^*</td>
<td>$(R^I)^*$</td>
</tr>
<tr>
<td>concrete domain</td>
<td>$u_1, \ldots, u_n., P$</td>
<td>$\text{h-father}\cdot,\text{age, age.}$</td>
<td>${ x \mid \langle u_1^I, \ldots, u_n^I \rangle \in P }$</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Many different DLs/DL constructors have been investigated.

**Introduction to DL: Knowledge Bases: TBoxes**

**For terminological knowledge:** TBox contains

- **Concept definitions**: $A \doteq C$ (A a concept name, C a complex concept)
  - Father $\doteq$ Man $\cap \exists\text{has-child}.\,\text{Human}$
  - Human $\doteq$ Mammal $\cap \forall\text{has-child}^-\cdot\,\text{Human}$
  - $\leadsto$ introduce macros/names for concepts, can be (a)cyclic

- **Axioms**: $C_1 \sqsubseteq C_2$ (C, complex concepts)
  - $\exists\text{favourite}.\,\text{Brewery} \sqsubseteq \exists\text{drinks}.\,\text{Beer}$
  - $\leadsto$ restrict your models

An interpretation $\mathcal{I}$ satisfies

- a concept definition $A \doteq C$ iff $A^\mathcal{I} = C^\mathcal{I}$
- an axiom $C_1 \sqsubseteq C_2$ iff $C_1^\mathcal{I} \sqsubseteq C_2^\mathcal{I}$
- a TBox $\mathcal{I}$ if $\mathcal{I}$ satisfies all definitions and axioms in $\mathcal{T}$
  - $\leadsto$ $\mathcal{I}$ is a model of $\mathcal{T}$
Introduction to DL: Knowledge Bases: ABoxes

For assertional knowledge: ABox contains

Concept assertions  
\( a : C \)  
(a an individual name, \( C \) a complex concept)  
John : Man \( \sqcap \forall \text{has-child}.(\text{Male} \sqcap \text{Happy}) \)

Role assertions  
\( \langle a_1, a_2 \rangle : R \)  
(a_i individual names, \( R \) a role)  
\( \langle \text{John}, \text{Bill} \rangle : \text{has-child} \)

An interpretation \( \mathcal{I} \) satisfies

a concept assertion  
\( a : C \)  
iff  
\( a^\mathcal{I} \in C^\mathcal{I} \)

a role assertion  
\( \langle a_1, a_2 \rangle : R \)  
iff  
\( \langle a_1^\mathcal{I}, a_2^\mathcal{I} \rangle \in R^\mathcal{I} \)

an ABox  
\( \mathcal{A} \)  
iff  
\( \mathcal{I} \) satisfies all assertions in \( \mathcal{A} \)

\( \sim \)  
\( \mathcal{I} \) is a model of \( \mathcal{A} \)

Introduction to DL: Basic Inference Problems

Subsumption:  
\( C \sqsubseteq D \)  
Is \( C^\mathcal{I} \subseteq D^\mathcal{I} \) in all interpretations \( \mathcal{I} \)?

w.r.t. TBox \( \mathcal{T} \):  
\( C \sqsubseteq_T D \)  
Is \( C^\mathcal{I} \subseteq D^\mathcal{I} \) in all models \( \mathcal{I} \) of \( \mathcal{T} \)?

\( \sim \)  
structure your knowledge, compute taxonomy

Consistency:  
Is \( C \) consistent w.r.t. \( \mathcal{T} \)?  
Is there a model \( \mathcal{I} \) of \( \mathcal{T} \) with \( C^\mathcal{I} \neq \emptyset \)?

of ABox \( \mathcal{A} \):  
Is \( \mathcal{A} \) consistent?  
Is there a model of \( \mathcal{A} \)?

of KB \( (\mathcal{T}, \mathcal{A}) \):  
Is \( (\mathcal{T}, \mathcal{A}) \) consistent?  
Is there a model of both \( \mathcal{T} \) and \( \mathcal{A} \)?

Inference Problems are closely related:

\( C \sqsubseteq_T D \)  
iff  
\( C \sqcap \neg D \) is inconsistent w.r.t. \( \mathcal{T} \),
(no model of \( \mathcal{I} \) has an instance of \( C \sqcap \neg D \))

\( C \) is consistent w.r.t. \( \mathcal{T} \)  
iff  
not \( C \sqsubseteq_T A \sqcap \neg A \)

\( \sim \)  
Decision Procedures for consistency (w.r.t. TBoxes) suffice
For most DLs, the basic inference problems are *decidable*, with complexities between $P$ and $\text{ExpTime}$.

**Why is decidability important?** Why does semi-decidability not suffice?

If subsumption (and hence consistency) is undecidable, and

- subsumption is semi-decidable, then consistency is not semi-decidable
- consistency is semi-decidable, then subsumption is not semi-decidable

- Quest for a “highly expressive” DL with “practicable” inference problems

where expressiveness depends on the application
practicability changed over the time

---

**Introduction to DL: History**

Complexity of Inferences provided by DL systems over the time

<table>
<thead>
<tr>
<th>Time Period</th>
<th>Complexity Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>late '80s</td>
<td>Undecidable</td>
</tr>
<tr>
<td>early '90s</td>
<td>ExpTime</td>
</tr>
<tr>
<td>mid '90s</td>
<td>PSpace</td>
</tr>
<tr>
<td>late '90s</td>
<td>PTime</td>
</tr>
</tbody>
</table>

- KL-ONE, NIKL
- Classic (AT&T)
- Crack, Kris
- Fact, DLP, Race

Investigation of Complexity of Inference Problems/Algorithms starts
Introduction to DL: State-of-the-implementation-art

In the last 5 years, DL-based systems were built that

- can handle DLs far more expressive than $\mathcal{ALC}$ (close relatives of converse-DPDL)
  - Number restrictions: “people having at most 2 cats and exactly 1 dog”
  - Complex roles: inverse (“has-child” — “child-of”),
    transitive closure (“offspring” — “has-child”),
    role inclusion (“has-daughter” — “has-child”), etc.

- implement provably sound and complete inference algorithms
  (for ExpTime-complete problems)
- can handle large knowledge bases
  (e.g., Galen medical terminology ontology: 2,740 concepts, 413 roles, 1,214 axioms)
- are highly optimised versions of tableau-based algorithms
- perform (surprisingly well) on benchmarks for modal logic reasoners
  (Tableaux’98, Tableaux’99)

Relationship with Other Logical Formalisms: First Order Predicate Logic

Most DLs are decidable fragments of FOL: Introduce

a unary predicate $A$ for a concept name $A$

a binary relation $R$ for a role name $R$

Translate complex concepts $C$, $D$ as follows:

- $t_x(A) = A(x)$, $t_y(A) = A(y)$,
- $t_x(C \cap D) = t_x(C) \land t_x(D)$, $t_y(C \cap D) = t_y(C) \land t_y(D)$,
- $t_x(C \cup D) = t_x(C) \lor t_x(D)$, $t_y(C \cup D) = t_y(C) \lor t_y(D)$,
- $t_x(\exists R.C) = \exists y.R(x, y) \land t_y(C)$, $t_y(\exists R.C) = \exists x.R(y, x) \land t_x(C)$,
- $t_x(\forall R.C) = \forall y.R(x, y) \Rightarrow t_y(C)$, $t_y(\forall R.C) = \forall x.R(y, x) \Rightarrow t_x(C)$.

A TBox $\mathcal{T} = \{C_i \vdash D_i\}$ is translated as

$$\Phi_\mathcal{T} = \forall x. \bigwedge_{1 \leq i \leq n} t_x(C_i) \Leftrightarrow t_x(D_i)$$
$C$ is consistent iff its translation $t_x(C)$ is satisfiable,

$C$ is consistent w.r.t. $T$ iff its translation $t_x(C) \land \Phi_T$ is satisfiable,

$C \sqsubseteq D$ iff $t_x(C) \Rightarrow t_x(D)$ is valid

$C \sqsubseteq_T D$ iff $\Phi_t \Rightarrow \forall x.(t_x(C) \Rightarrow t_x(D))$ is valid.

$\rightarrow \mathcal{ALC}$ is a fragment of FOL with 2 variables (L2), known to be decidable

$\rightarrow \mathcal{ALC}$ with inverse roles and Boolean operators on roles is a fragment of L2

$\rightarrow$ further adding number restrictions yields a fragment of C2

(L2 with “counting quantifiers”), known to be decidable

$\star$ in contrast to most DLs, adding transitive roles (binary relations/transitive closure operator) to L2 leads to undecidability

$\star$ many DLs (like many modal logics) are fragments of the Guarded Fragment

$\star$ most DLs (like many modal logics) are fragments of the Guarded Fragment

$L2$ is NExpTime-complete, most DLs are in ExpTime

DLs and Modal Logics are closely related:

$\mathcal{ALC} \equiv \text{multi-modal K:}$

\[
\begin{align*}
C \sqcap D & \equiv C \land D, \\
\neg C & \equiv \neg C, \\
\exists R.C & \equiv \langle R \rangle C, \\
\forall R.C & \equiv [R]C
\end{align*}
\]

transitive roles $\equiv$ transitive frames (e.g., in K4)

regular expressions on roles $\equiv$ regular expressions on programs (e.g., in PDL)

inverse roles $\equiv$ converse programs (e.g., in C-PDL)

number restrictions $\equiv$ deterministic programs (e.g., in D-PDL)

$\rightarrow$ no TBoxes available in modal logics

$\rightarrow$ “internalise” axioms using a universal role $u$: $C \vdash D \equiv [u](C \Leftrightarrow D)$

$\rightarrow$ no ABox available in modal logics $\rightarrow$ use nominals
Applications of Description Logics

Application Areas

Terminological KR and Ontologies
- DLs initially designed for terminological KR (and reasoning)
- Natural to use DLs to build and maintain ontologies

Semantic Web
- Semantic markup will be added to web resources
  - Aim is “machine understandability”
- Markup will use Ontologies to provide common terms of reference with clear semantics
- Requirement for web based ontology language
  - Well defined semantics
  - Builds on existing Web standards (XML, RDF, RDFS)
- Resulting language (DAML+OIL) is based on a DL (SHIQ)
- DL reasoning can be used to, e.g.,
  - Support ontology design and maintenance
  - Classify resources w.r.t. ontologies
Application Areas II

- **Configuration**
  - **Classic** system used to configure telecoms equipment
  - Characteristics of components described in DL KB
  - Reasoner checks validity (and price) of configurations

- **Software information systems**
  - LaSSIE system used DL KB for flexible software documentation and query answering

- **Database applications**
  - ...

Database Schema and Query Reasoning

- **DLR** (n-ary DL) can capture semantics of many conceptual modelling methodologies (e.g., EER)
- Satisfiability preserving mapping to *SHIQ* allows use of DL reasoners (e.g., FaCT, RACER)
- DL Abox can also capture semantics of conjunctive queries
  - Can reason about query containment w.r.t. schema
- DL reasoning can be used to support
  - Schema design, evolution and query optimisation
  - Source integration in heterogeneous databases/data warehouses
  - Conceptual modelling of multidimensional aggregation
- E.g., **I.COM** Intelligent Conceptual Modelling tool (Enrico Franconi)
  - Uses FaCT system to provide reasoning support for EER
Terminological KR and Ontologies

- General requirement for medical terminologies
- Static lists/taxonomies difficult to build and maintain
  - Need to be very **large** and highly interconnected
  - Inevitably contain many **errors** and **omissions**
- Galen project aims to replace static hierarchy with DL
  - **Describe** concepts (e.g., spiral fracture of left femur)
  - Use DL classifier to **build taxonomy**
- Needed expressive DL **and** efficient reasoning
  - Descriptions use transitive/inverse roles, GCIs etc.
  - Very large KBs (tens of thousands of concepts)
    - Even prototype KB is very large ($\approx$3,000 concepts)
    - Existing (incomplete) classifier took $\approx$24 hours to classify KB
    - FaCT system (sound and complete) takes $\approx$60 seconds
Reasoning Support for Ontology Design

DL reasoner can be used to support design and maintenance

Example is OilEd ontology editor (for DAML+OIL)
- Frame based interface (like Protegé, OntoEdit, etc.)
- Extended to clarify semantics and capture whole DAML+OIL language
  - Slots explicitly existential or value restrictions
  - Boolean connectives and nesting
  - Properties for slot relations (transitive, functional etc.)
  - General axioms

Reasoning support for OilEd provided by FaCT system
- Frame representation translated into $SHIQ$
- Communicates with FaCT via CORBA interface
- Indicates inconsistencies and implicit subsumptions
- Can make implicit subsumptions explicit in KB

DAML+OIL Medical Terminology Examples

E.g., DAML+OIL medical terminology ontology
- Transitive roles capture transitive partonomy, causality, etc.
  - Smoking $\sqsubseteq \exists causes.Cancer$ plus Cancer $\sqsubseteq \exists causes.Death$
  - $\Rightarrow$ Cancer $\sqsubseteq$ FatalThing

- GCIs represent additional non-definitional knowledge
  - Stomach-Ulcer $\equiv$ Ulcer $\sqcap \exists hasLocation.Stomach$ plus
    - Stomach-Ulcer $\sqsubseteq \exists hasLocation.Lining-Of-Stomach$
    - $\Rightarrow$ Ulcer $\sqcap \exists hasLocation.Stomach$ $\sqsubseteq$ OrganLiningLesion

- Inverse roles capture e.g. causes/causedBy relationship
  - Death $\sqcap \exists causedBy.Smoking$ $\sqsubseteq$ PrematureDeath
  - $\Rightarrow$ Smoking $\sqsubseteq$ CauseOfPrematureDeath

- Cardinality restrictions add consistency constraints
  - BloodPressure $\sqsubseteq \exists hasValue.(High $\sqcup$ Low) \sqsubseteq 1 hasValue$ plus
    - High $\sqsubseteq \neg$Low $\Rightarrow$ HighLowBloodPressure $\sqsubseteq \bot$
Reasoning Procedures: Deciding Consistency of $\mathcal{ALCN}$ Concepts

As a warm-up, we describe a **tableau-based algorithm** that

- decides consistency of $\mathcal{ALCN}$ concepts,
- tries to build a (tree) model $\mathcal{I}$ for input concept $C_0$,
- breaks down $C_0$ syntactically, inferring constraints on elements in $\mathcal{I}$,
- uses **tableau rules** corresponding to operators in $\mathcal{ALCN}$ (e.g., $\rightarrow\sqcap$, $\rightarrow\exists$)
- works non-deterministically, in PSpace
- stops when clash occurs
- terminates
- returns “$C_0$ is consistent” iff $C_0$ is consistent
Reasoning Procedures: Tableau Algorithm

- works on a tree (semantics through viewing tree as an ABox):
  nodes represent elements of $\Delta^I$, labelled with sub-concepts of $C_0$
edges represent role-successorships between elements of $\Delta^I$
- works on concepts in negation normal form: push negation inside using de Morgan’ laws and
  \[
  \neg(\exists R.C) \leadsto \forall R.\neg C \quad \neg(\forall R.C) \leadsto \exists R.\neg C
  \]
  \[
  \neg(\leq n R) \leadsto (\geq (n + 1)R) \quad \neg(\geq n R) \leadsto (\leq (n - 1)R) \quad (n \geq 1)
  \]
  \[
  \neg(\geq 0 R) \leadsto A \sqcap \neg A
  \]
- is initialised with a tree consisting of a single (root) node $x_0$ with $L(x_0) = \{C_0\}$:
- a tree $T$ contains a clash if, for a node $x$ in $T$,
  \[
  \{A, \neg A\} \subseteq L(x) \quad \text{or} \quad \{(\geq m R), (\leq n R)\} \subseteq L(x) \quad \text{for } n < m
  \]
- returns “$C_0$ is consistent” if rules can be applied s.t. they yield clash-free, complete (no more rules apply) tree

---

Reasoning Procedures: $\mathcal{ALC}$ Tableau Rules

<table>
<thead>
<tr>
<th>$x\bullet {C_1 \sqcap C_2, \ldots}$</th>
<th>$\rightarrow \sqcap$</th>
<th>$x\bullet {C_1 \sqcap C_2, C_1, C_2, \ldots}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x\bullet {C_1 \sqcup C_2, \ldots}$</td>
<td>$\rightarrow \sqcup$</td>
<td>$x\bullet {C_1 \sqcup C_2, C, \ldots}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for $C \in {C_1, C_2}$</td>
</tr>
<tr>
<td>$x\bullet {\exists R.C, \ldots}$</td>
<td>$\rightarrow \exists$</td>
<td>$x\bullet {\exists R.C, \ldots}$</td>
</tr>
<tr>
<td>$\downarrow R$</td>
<td></td>
<td>$y\bullet {C}$</td>
</tr>
<tr>
<td>$y\bullet {\ldots}$</td>
<td>$\rightarrow \forall$</td>
<td>$x\bullet {\forall R.C, \ldots}$</td>
</tr>
<tr>
<td>$\downarrow R$</td>
<td></td>
<td>$y\bullet {\ldots, C}$</td>
</tr>
</tbody>
</table>
\[ x \cdot \{ (\geq n \ R), \ldots \} \rightarrow_{\geq} x \cdot \{ (\geq n \ R), \ldots \} \]

\[ R \]

\[ x \cdot \{ (\leq n \ R), \ldots \} \rightarrow_{\leq} x \cdot \{ (\leq n \ R), \ldots \} \]

\[ R \]


**Reasoning Procedures: Soundness and Completeness**

**Lemma** Let \( C_0 \) be an \( \mathcal{ALCN} \) concept and \( T \) obtained by applying the tableau rules to \( C_0 \). Then

1. the rule application terminates,
2. if \( T \) is clash-free and complete,
   then \( T \) defines (canonical) (tree) model for \( C_0 \), and
3. if \( C_0 \) has a model \( \mathcal{I} \), then the rules can be applied such that they yield a clash-free and complete \( T \).

**Corollary**

1. The tableau algorithm is a (PSpace) decision procedure for consistency (and subsumption) of \( \mathcal{ALCN} \) concepts
2. \( \mathcal{ALCN} \) has the tree model property
Proof of the Lemma

1. (Termination) The algorithm “monotonically” constructs a tree whose
   depth is linear in $|C_0|$: quantifier depth decreases from node to succs.
   breadth is linear in $|C_0|$ (even if number in NRs are coded binarily)

2. (Canonical model) Complete, clash-free tree $T$ defines a (tree) pre-model $I$:

   - nodes $x$ correspond to elements $x \in \Delta^T$
   - edges $x \xrightarrow{R} y$ define role-relationship
   - $x \in A^T$ iff $A \in \mathcal{L}(x)$ for concept names $A$

   $\leadsto$ Easy to that $C \in \mathcal{L}(x) \Rightarrow x \in C^T$ — if $C \neq (\geq n \ R)$
   If $(\geq n \ R) \in \mathcal{L}(x)$, then $x$ might have less than $n \ R$-successors, but
   the $\rightarrow\geq$-rule ensures that there is $\geq 1 \ R$-successor...

3. (Completeness) Use model $I$ of $C_0$ to steer application of non-deterministic rules
   $(\rightarrow_{\cup}, \rightarrow_{\leq})$ via mapping

   $$\pi : \text{Nodes of Tree} \rightarrow \Delta^T \quad \text{with} \quad C \in \mathcal{L}(x) \Rightarrow \pi(x) \in C^T.$$  

   This easily implies clash-freenes of the tree generated.
To make the tableau algorithm run in PSpace:

1. observe that branches are independent from each other
2. observe that each node (label) requires linear space only
3. recall that paths are of length $\leq |C_0|$
4. construct/search the tree depth first
5. re-use space from already constructed branches

$\Rightarrow$ space polynomial in $|C_0|$ suffices for each branch/for the algorithm

$\Rightarrow$ tableau algorithm runs in NPspace (Savitch: NPspace = PSpace)

Reasoning Procedures: Extensibility

This tableau algorithm can be modified to a PSpace decision procedure for

- $\mathcal{ALC}$ with qualifying number restrictions $\geq n R C$ and $\leq n R C$
- $\mathcal{ALC}$ with inverse roles has-child$^-$
- $\mathcal{ALC}$ with role conjunction $\exists (R \cap S).C$ and $\forall (R \cap S).C$
- TBoxes with acyclic concept definitions:
  - unfolding (macro expansion) is easy, but suboptimal: may yield exponential blow-up
  - lazy unfolding (unfolding on demand) is optimal, consistency in PSpace decidable
Language extensions that require more elaborate techniques include

- **TBoxes with general axioms** $C_i \sqsubseteq D_i$:
  - each node must be labelled with $\neg C_i \cup D_i$
  - quantifier depth no longer decreases
  - $\leadsto$ termination not guaranteed

- **Transitive closure of roles**:
  - node labels $(\forall R^* . C)$ yields $C$ in all $R^n$-successor labels
  - quantifier depth no longer decreases
  - $\leadsto$ termination not guaranteed

  Use **blocking** (cycle detection) to ensure termination
  (but the right blocking to retain soundness and completeness)
Non-Termination

As already mentioned, for $\mathcal{A}\mathcal{LC}$ with general axioms basic algorithm is non-terminating

E.g. if $\text{human} \sqsubseteq \exists \text{has-mother}. \text{human} \in T$, then

~$\text{human} \sqcup \exists \text{has-mother}. \text{human}$ added to every node

\[
\mathcal{L}(w) = \{\text{human}, (\neg \text{human} \sqcup \exists \text{has-mother}. \text{human}), \exists \text{has-mother}. \text{human}\}
\]

Blocking

When creating new node, check ancestors for equal (superset) label

If such a node is found, new node is blocked
Blocking with More Expressive DLs

- Simple subset blocking may not work with more complex logics
- E.g., reasoning with inverse roles
  - Expanding node label can affect predecessor
  - Label of blocking node can affect predecessor
  - E.g., testing $C \sqcap \exists S . C$ w.r.t. Tbox

$$T = \{ \top \sqsubseteq \forall R^-(\forall S^- . \neg C), \top \sqsubseteq \exists R.C \}$$

### Dynamic Blocking

- Solution (for inverse roles) is **dynamic blocking**
  - Blocks can be established broken and re-established
  - Continue to expand $\forall R.C$ terms in blocked nodes
  - Check that cycles satisfy $\forall R.C$ concepts

Diagram of dynamic blocking with labels for nodes and arrows indicating blocking.
Non-finite Models

- With number restrictions some satisfiable concepts have only non-finite models
- E.g., testing \( \neg C \) w.r.t. \( T = \{ \top \subseteq \exists R. C, \top \subseteq \leq 1 R^- \} \)

\[
\begin{align*}
\mathcal{L}(w) &= \{ \neg C, \exists R. C, \leq 1 R^- \} \\
\mathcal{L}(x) &= \{ C, \exists R. C, \leq 1 R^- \} \\
\mathcal{L}(y) &= \{ C, \exists R. C, \leq 1 R^- \}
\end{align*}
\]

model must be non-finite

Inadequacy of Dynamic Blocking

- With non-finite models, even dynamic blocking not enough
- E.g., testing \( \neg C \) w.r.t. \( T = \{ \top \subseteq \exists R. (C \cap \exists R^- . \neg C), \top \subseteq \leq 1 R^- \} \)

\[
\begin{align*}
\mathcal{L}(w) &= \{ \neg C, \exists R.(C \cap \exists R^- . \neg C), \leq 1 R^- \} \\
\mathcal{L}(x) &= \{ (C \cap \exists R^- . \neg C), \exists R.(C \cap \exists R^- . \neg C), \leq 1 R^-, C, \exists R^- . \neg C \} \\
\mathcal{L}(y) &= \{ (C \cap \exists R^- . \neg C), \exists R.(C \cap \exists R^- . \neg C), \leq 1 R^-, C, \exists R^- . \neg C \}
\end{align*}
\]

\( \exists R^- . \neg C \in \mathcal{L}(y) \) not satisfied

Inconsistency due to \( \leq 1 R^- \in \mathcal{L}(y) \) and \( C \in \mathcal{L}(x) \)
Double Blocking I

Problem due to $\exists R^- . \neg C$ term only satisfied in predecessor of blocking node

\[
\mathcal{L}(w) = \{ \neg C, \exists R. (C \cap \exists R^- . \neg C), \leq 1R^- \}
\]

Solution is **Double Blocking** (pairwise blocking)
- Predecessors of blocked and blocking nodes also considered
- In particular, $\exists R . C$ terms satisfied in predecessor of blocking node must also be satisfied in predecessor of blocked node $\neg C \in \mathcal{L}(w)$

Double Blocking II

Due to pairwise condition, block no longer holds
- Expansion continues and contradiction discovered

\[
\mathcal{L}(w) = \{ \neg C, \exists R. (C \cap \exists R^- . \neg C), \leq 1R^- \}
\]

\[
\mathcal{L}(x) = \{(C \cap \exists R^- . \neg C), \exists R. (C \cap \exists R^- . \neg C), \leq 1R^-, C, \exists R^- . \neg C, \neg C \}
\]

\[
\mathcal{L}(y) = \{(C \cap \exists R^- . \neg C), \exists R. (C \cap \exists R^- . \neg C), \leq 1R^-, C, \exists R^- . \neg C \}
\]

Clash
We left out a variety of complexity results for

- concept consistency of other DLs
  (e.g., those with “concrete domains”)

- other standard inferences
  (e.g., deciding consistency of ABoxes w.r.t. TBoxes)

- “non-standard” inferences such as
  - matching and unification of concepts
  - rewriting concepts
  - least common subsumer (of a set of concepts)
  - most specific concept (of an ABox individual)
Implementing DL Systems

Naive Implementations

Problems include:

- **Space usage**
  - Storage required for tableaux datastructures
  - Rarely a serious problem in practice

- **Time usage**
  - Search required due to non-deterministic expansion
  - **Serious** problem in practice
  - Mitigated by:
    - Careful *choice of algorithm*
    - Highly *optimised implementation*
Careful Choice of Algorithm

- Transitive roles instead of transitive closure
  - Deterministic expansion of $\exists R.C$, even when $R \in R_+$
  - (Relatively) simple blocking conditions
  - Cycles always represent (part of) cyclical models

- Direct algorithm/implementation instead of encodings
  - GCI axioms can be used to “encode” additional operators/axioms
  - Powerful technique, particularly when used with FL closure
  - Can encode cardinality constraints, inverse roles, range/domain,
    ...
    - E.g., (domain $R.C) \equiv \exists R. \top \sqsubseteq C$
  - (FL) encodings introduce (large numbers of) axioms
  - BUT even simple domain encoding is disastrous with large numbers of roles

Highly Optimised Implementation

Optimisation performed at 2 levels

- Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use enhanced traversal that exploits information from previous tests
  - Also use structural information from KB
    - E.g., to select order in which to classify concepts

- Computing **subsumption** between concepts
  - Objective is to minimise cost of single subsumption tests
  - Small number of hard tests can dominate classification time
  - Recent DL research has addressed this problem (with considerable success)
Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

- Pre-processing optimisations
  - Aim is to simplify KB and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms

- Algorithmic optimisations
  - Main aim is to reduce search space due to non-determinism
  - Integral part of implementation
  - But often generally applicable to search based algorithms

Pre-processing Optimisations

Useful techniques include

- Normalisation and simplification of concepts
  - Refinement of technique first used in KRIS system
  - Lexically normalise and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)

- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into “definition” axioms
  - Definition axioms efficiently dealt with by lazy expansion

- Avoidance of potentially costly reasoning whenever possible
  - Normalisation can discover “obvious” (un)satisfiability
  - Structural analysis can discover “obvious” subsumption
Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R. C \rightarrow \neg \forall R. \neg C$
  - $C \sqcup D \rightarrow \neg (\neg C \sqcap \neg D)$

- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \rightarrow A \sqcap C \sqcap D$
  - $\forall R. \top \rightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \rightarrow \bot$

- Lazily unfold concepts in tableaux algorithm
  - Use names/pointers to refer to complex concepts
  - Only add structure as required by progress of algorithm
  - Detect clashes between lexically equivalent concepts

\{HappyFather, $\neg \text{HappyFather}\} \rightarrow \text{clash}
\{\forall \text{has-child.}(\text{Doctor} \sqcup \text{Lawyer}), \exists \text{has-child.}(\neg \text{Doctor} \sqcap \neg \text{Lawyer})\} \rightarrow \text{search}

Absorption I

- Reasoning w.r.t. set of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to every node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes search space already $2^{100}$
  - GALEN (medical terminology) KB contains hundreds of axioms

- Reasoning w.r.t. “primitive definition” axioms is relatively efficient
  - For CN $\sqsubseteq D$, add $D$ only to node labels containing CN
  - For CN $\sqsupseteq D$, add $\neg D$ only to node labels containing $\neg$CN
  - Can expand definitions lazily
    - Only add definitions after other local (propositional) expansion
    - Only add definitions one step at a time
Absorption II

Transform GCIs into primitive definitions, e.g.

- \( CN \cap C \subseteq D \rightarrow CN \subseteq D \cup \neg C \)
- \( CN \cup C \supseteq D \rightarrow CN \supseteq D \cap \neg C \)

Absorb into existing primitive definitions, e.g.

- \( CN \subseteq A, CN \subseteq D \cup \neg C \rightarrow CN \subseteq A \cap (D \cup \neg C) \)
- \( CN \supseteq A, CN \supseteq D \cap \neg C \rightarrow CN \supseteq A \cup (D \cap \neg C) \)

Use lazy expansion technique with primitive definitions

- Disjunctions only added to “relevant” node labels

Performance improvements often too large to measure

- At least **four orders of magnitude** with GALEN KB

Algorithmic Optimisations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list

- Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking

- Caching
  - Cache partial models
  - Cache satisfiability status (of labels)

- Heuristic ordering of propositional and modal expansion
  - Min/maximise constrainedness (e.g., MOMS)
  - Maximise backtracking (e.g., oldest first)
Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved
  - Jump back to relevant branch points without exploring alternative branches
- Effect is to prune away part of the search space
- Performance improvements with GALEN KB again too large to measure

---

**Backjumping**

E.g., if $\exists R. \neg A \sqcap \forall R. (A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$

![Diagram](attachment://dependency-directed-backtracking-diagram.png)
Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem

- Cache (partial) models of concepts
  - Use to detect “obvious” non-subsumption
  - $C \nsubseteq D$ if $C \cap \neg D$ is satisfiable
  - $C \cap \neg D$ satisfiable if models of $C$ and $\neg D$ can be merged
  - If not, continue with standard subsumption test
  - Can use same technique in sub-problems

Summary

- Naive implementation results in effective non-termination
- Problem is caused by non-deterministic expansion (search)
  - GCIs lead to huge search space
- Solution (partial) is
  - Careful choice of logic/algorithm
  - Avoid encodings
  - Highly optimised implementation
- Most important optimisations are
  - Absorption
  - Dependency directed backtracking (backjumping)
  - Caching
- Performance improvements can be very large
  - E.g., more than four orders of magnitude
- The official DL homepage: http://dl.kr.org/
- The DL mailing list: dl@dl.kr.org
- Patrick Lambrix’s very useful DL site (including lots of interesting links):
  http://www.ida.liu.se/labs/iislab/people/patla/DL/index.html
- The annual DL workshop:
  Proceedings on-line available at:
  http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/
- The OIL homepage: http://www.ontoknowledge.org/oil/
- More about i-com: http://www.cs.man.ac.uk/~franconi/
- More about FaCT: http://www.cs.man.ac.uk/~horrocks/