Description Logic Reasoning

COMP62342
Sean Bechhofer
sean.bechhofer@manchester.ac.uk

Inference

- Ontologies provide
  - Vocabulary that describes a domain
  - Assumptions about the ways in which that vocabulary should be interpreted

- What can we then infer from that information?
  - In particular, what inferences can be drawn from the assumptions that we’ve expressed?
Recall that our semantics was defined in terms of Interpretations:
- Domain of discourse $\Delta$
- Function I mapping:
  - Individuals names $x, y, z$ to elements of $\Delta$
  - Class names $A, B, C$ to subsets of $\Delta$
  - Property names $R, S,$ to sets of pairs of elements of $\Delta$

Interpretations then extended to cover concept expressions:
- $(A \cap B)^I = A^I \cap B^I$

Interpretation is a model of an axiom $A$ iff the interpretation of the axiom holds:
- $I \models A \subseteq B$ iff $A^I \subseteq B^I$

Interpretation is a model of an ontology $O$ iff it is a model of all the axioms in $O$

Note use of logical ("German syntax") here rather than Manchester Syntax.
Inference

- What can we infer from an Ontology $O$?
  - And what do we mean by infer?

- The semantics describe the conditions under which an interpretation is a model of an Ontology $O$.
- It can be the case that due to the constraints that $O$ places on the interpretations, there are consequences that also hold in all the interpretations.
- Recall, an Ontology doesn’t define a single model, it is a set of constraints that define a set of possible models
  - No constraints (empty Ontology) means any model is possible
  - More constraints means fewer models
  - Too many constraints may mean no possible model (inconsistent Ontology)

Basic Inference Problems

- Subsumption
  - $C \subseteq_0 D$ iff $C^I \subseteq D^I$ in all models $I$ of $O$

- Equivalence
  - $C \equiv_0 D$ iff $C^I = D^I$ in all models $I$ of $O$

- Satisfiability
  - $C \not\subseteq_0 \bot$ iff $C^I$ non empty in some model $I$ of $O$

- Instantiation
  - $i \in_0 C$ iff $i \in C^I$ in all models $I$ of $O$

- Consistency
  - $O$ consistent iff there is at least one model $I$ of $O$

- Coherency
  - $O$ coherent iff all names classes are satisfiable

- Problems reducible to satisfiability:
  - e.g., $C \subseteq_0 D$ iff $(C \land \neg D)$ not satisfiable w.r.t. $O$
Example Inferences

- \( O = \{ B \subseteq A, C \subseteq B \} \)
  - \( C \subseteq O \ A \)

- \( O = \{ C \subseteq A \cap B \} \)
  - \( C \subseteq O \ A \)

- \( O = \{ \} \)
  - \( A \subseteq O \ A \cup B \)

- \( O = \{ C \equiv \exists R.A, B \subseteq A \} \)
  - \( \exists R.A \subseteq O \ C \)

- \( O = \{ B \subseteq A, x:B \} \)
  - \( x \subseteq O \ B \)

- \( O = \{ C \subseteq A \cap B \} \)
  - \( C \not\subseteq O \bot \)

- \( O = \{ C \subseteq A, C \subseteq \neg A \} \)
  - \( C \not\subseteq O \bot \)
  - \( O \) incoherent

- \( O = \{ C \subseteq A, C \subseteq \neg A, x:C \} \)
  - \( O \) inconsistent

Consistency and Unsatisfiability

- Note the difference between class satisfiability and ontology consistency

- A class \( C \) is unsatisfiable if there are no models such that its interpretation is non-empty
- An Ontology \( O \) is inconsistent if there are no models of \( O \)
- A consistent Ontology \( O \) may contain unsatisfiable classes.

- \( O = \{ C \subseteq A \cap B, D \subseteq C \cap \neg B \} \)
  - \( D \) unsatisfiable, but models exists thus \( O \) is consistent...

- \( O = \{ C \subseteq \forall R.\neg A, x:C, y:A, <x,y>:R \} \)
  - Inconsistent Ontology
Why are these useful?

- **Subsumption:** check knowledge is correct
  - Build classification hierarchies of primitive (named) classes
- **Equivalence:** check knowledge is minimally redundant
- **Satisfiability:** check knowledge is meaningful
- **Instantiation:** check if individual instance of class C.
  - Supporting query.

Structural Approaches

- Early implementations used *structural* approaches
- E.g. to check subsumption
  1. Normalise expressions
  2. Compare the structure of the expressions to see if there is “overlap”.
- This is effective, but hard to get complete results, particularly in the face of complex axioms (or GCIs as they are sometimes known).

- An alternative is to use an approach based on the underlying semantics, e.g. *tableaux*. 
Tableaux Algorithms: Basics

- Tableaux algorithms used to test satisfiability
- Try to build tree-like model $I$ of input class $C$
- Work on classes in negation normal form
  - Rewrite and push in negation using de Morgan’s laws
  - E.g. $\neg \exists R.C$ to $\forall R.\neg C$
- Break down $C$ syntactically, inferring constraints on elements of $I$
- Decomposition uses tableau rules corresponding to constructors in the logic (e.g. $\land$, $\exists$)
  - Some rules are nondeterministic, e.g. they involve some choice
  - In practice, this means search.

- Try and build a “completion tree” by applying rules
- Stop when a clash occurs or when no more rules are applicable.
- Blocking (cycle check) used to guarantee termination
- Returns “$C$ is consistent” iff $C$ is consistent
  - Tree model property
Tableaux Algorithms: Details

- Work on tree $T$ representing model $I$ of class $C$
  - Nodes $x,y$ represent elements of domain $\Delta$
  - Nodes labelled with $L(x)$, sub-expressions of $C$
  - Edges represent role-successorships between elements of $\Delta$
- $T$ initialised with single root node labelled $\{C\}$
- Tableau rules repeatedly applied to node labels.
  - Extend labels of a node or extend/modify the tree structure
  - Rules can be blocked, e.g. if a predecessor node has superset label
  - Nondeterministic rules mean we may need to search for possible extensions
- $T$ contains a Clash if there is an obvious contradiction in some node label
  - E.g. $\{A, \neg A\} \subseteq L(x)$ for some class $A$ and node $x$

- $T$ fully expanded if no rules are applicable
- $C$ satisfiable iff fully expanded clash-free tree $T$ found
  - There is then a correspondence between $T$ and a model of $C$ (see later remarks regarding completeness)
- Thus the tableaux algorithm helps us by finding a witness for the consistency of $C$
  - There is some model.
ALC

- Propositional constructors
  - \( \cap, \cup, \neg \)
- Role Quantifiers
  - \( \exists, \forall \)
- Top and Bottom
  - \( T, \bot \)

- Concept names, \( T, \bot \) are Concepts. Role names are Roles.
- For \( C, D \) concepts and \( R \) a role, the following are Concepts:
  - \( \neg C \)
  - \( C \cap D \)
  - \( C \cup D \)
  - \( \exists R.C \)
  - \( \forall R.C \)

ALC and OWL

<table>
<thead>
<tr>
<th>ALC</th>
<th>OWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cap )</td>
<td>and</td>
</tr>
<tr>
<td>( \cup )</td>
<td>or</td>
</tr>
<tr>
<td>( \neg )</td>
<td>not</td>
</tr>
<tr>
<td>( \exists )</td>
<td>some</td>
</tr>
<tr>
<td>( \forall )</td>
<td>only</td>
</tr>
<tr>
<td>( T )</td>
<td>thing</td>
</tr>
<tr>
<td>( \bot )</td>
<td>nothing</td>
</tr>
</tbody>
</table>
Tableaux Rules for ALC

- $\times \times \{C_1 \cap C_2, \ldots\} \rightarrow \cap \times \times \{C_1 \cap C_2, C_3, \ldots\}$

- $\times \times \{C_1 \cup C_2, \ldots\} \rightarrow \cup \times \times \{C_1 \cup C_2, C_3, \ldots\}$
  - For $C \in \{C_1, C_2\}$

- $\times \times \exists R.C, \ldots \rightarrow \exists R.C, \ldots$
  - $y \downarrow \{C\}$

- $\times \{\forall R.C, \ldots\} \rightarrow \forall R.C, \ldots$
  - $y \downarrow \{C, \ldots\}$

Algorithm Examples

- Test the satisfiability of
  - $\exists R.A \cap \forall R.B$
  - $\exists R.A \cap \forall R.\neg A$
  - $\exists R.A \cap \forall S.\neg A$
  - $\exists R.(A \cup \exists R.B) \cap \forall R.\neg A \cap \forall R.(\forall R.\neg B)$
Is it right?

• How do we know whether our algorithm is doing the “right thing”?  
  – And what is the “right thing”?

• Soundness and Completeness help us characterise this.  
  – Soundness: we get correct answers  
  – Completeness: we get all the answers

• For our tableaux algorithm  
  – Soundness: if the algorithm says that C is satisfiable, then it is (according to the semantics)  
  – Completeness: if C is satisfiable (according to the semantics) then the algorithm will tell us this

Termination

• Given a concept expression C, our algorithm will terminate

• Informal argument:  
  – Rules (other than \(\rightarrow\)) are never applied twice on the same label  
  – The \(\rightarrow\) rule is never applied to node N more than \(n\) times where \(n\) is the number of direct successors of N.  
  – Each rule application on a label C adds labels D such that D is a strict sub-expression of C
Soundness

- If, given a concept description $C$, the algorithm terminates with a clash-free completion tree, then $C$ is satisfiable

- Informal argument
  - Given the clash-free completion tree, we can produce an interpretation where $C$ is non-empty

Completeness

- For completeness, we need to show that given a satisfiable concept expression $C$, if we start the tableaux with $C$, then we will arrive at a fully expanded clash-free tree

- Informal argument
  - As $C$ is satisfiable, we know there is an interpretation $I$ where $C^I$ is non-empty.
  - We can use this interpretation to guide the construction of the tableaux – in particular guiding choices.
Satisfiability w.r.t Axioms

- Our basic algorithm just operates on class expressions and doesn’t consider any axioms
  
- For each axiom \( C \sqsubseteq D \) in \( O \), add \( \neg C \sqsupset D \) to every node label
    - Can rewrite Ontology in terms of \( \sqsubseteq \)
  
- Potentially very expensive!
    - Adding a disjunction to every node in the graph

Unfolding

- Unfolding allows us to deal with particular forms of Ontology.

- Consider a Ontology \( O \) that only contains definitions
  - E.g. \( C \equiv D \) or \( C \sqsubseteq D \), where \( C \) is a concept name.
  
- For any concept \( A \) occurring in \( D \), we say \( A \) directly uses \( D \) and define uses as the transitive closure of directly uses.
  
- \( O \) contains a cycle if there is an atomic concept that uses itself.
  - \( \{A \sqsubseteq B, B \sqsubseteq C, C \equiv D\} \)
  - \( \{A \sqsubseteq B, B \equiv \exists R.C, C \sqsubseteq A\} \)

- If \( O \) is acyclic, we can expand and unfold the Ontology.
Unfolding

- To test satisfiability of concept description C w.r.t. acyclic Ontology O

  1. For any axiom \( B \equiv A \)
     - Replace all occurrences of B in C with A.

  2. For any axiom \( B \sqsupseteq A \)
     - Replace all occurrences of B in C with \( B' \sqcap A \), where \( B' \) is a new concept name.

  3. Then proceed with tableaux as normal on the new description.

Unfolding Examples

- Test satisfiability of \( \neg (\neg A \sqcap \exists R.C) \)

  1. w.r.t. the axioms \( \{A \sqsubseteq C \sqcap \exists R.C\} \)
  2. w.r.t. the axioms \( \{A \equiv \neg C\} \)
Tableau Rule for Transitive Roles

- We can also consider ALC plus transitive roles
  - i.e. allowing assertions about the transitivity of a role (rather than allowing us to talk about the transitive closure
- This then requires an additional rule for transitive role \( R \)

\[
\begin{array}{c}
x & \{ \forall R.C, \ldots \} & x \{ \forall R.C, \ldots \} \\
\uparrow R & \quad & \quad \downarrow R \\
y & \{ \ldots \} & \{ \forall R.C, \ldots \} \\
\end{array}
\]

- No longer naturally terminating (e.g. if \( C = \exists R.T \))
- We need a blocking strategy
  - Simple blocking is enough for ALC + transitive roles
  - Do not expand the node label if the ancestor has superset label
  - Need more for more expressive logics.

Algorithm Examples

- Test the satisfiability of

\[
\exists S.C \land \forall S.(\neg C \cup \neg D) \land \exists R.C \land \forall R(\exists R.C)
\]

Where \( R \) is a transitive role
More Expressive DLS

- Basic technique can be extended to deal with
  - Role inclusion axioms
  - Number restrictions
  - Inverse Roles
  - Concrete domains
  - Aboxes

- Extend expansion rules and use more sophisticated blocking strategy.
- Forest instead of tree for Individual Facts
  - Root nodes correspond to individuals in Ontology.

Scalability

- Reasoning with DL languages is hard
  - Ontologies on the web may grow large
  - Particularly with Instance data.

- Space usage
  - Storage required for tableaux datastructures
  - Rarely serious problem in practice
  - But problems with inverse roles and cyclical Ontologies

- Time usage
  - Non-deterministic rules lead to search
  - Serious problem in practice
  - Mitigated by
    - Careful choice of algorithm
    - Highly optimised implementations
Choice of Algorithm

- Transitive roles rather than transitive closure
  - Deterministic expansion of $\exists R. C$ even when $R$ in $R^+$
  - Relatively simple blocking conditions
- Direct algorithm/implementation instead of encodings
  - GCI axioms can be used to encode additional operators/axioms
  - E.g. domain and range
    - $(\text{domain } R) \subseteq \exists R. T \subseteq C$
  - But even simple domain encoding yields terrible performance with large numbers of roles.

Trade Offs: The Design Triangle

Expressivity (Representational Adequacy)

Usability (Weak Cognitive Adequacy vs. Cognitive Complexity)

Computability (vs. Computational and Implementational Complexity)
Cognitive Adequacy

• Strong Cognitive Adequacy
  – A KR is SCA if it is “a (psychologically valid) cognitive model of [a human’s] knowledge” (Strube, 1992)
    • If strong adequacy is claimed, the system is supposed to function like a human expert, at least in a circumscribed domain. In short, strongly adequate systems employ the very same principles of cognitive functioning as human experts do

• Weak Cognitive Adequacy
  – A KR is WCA if it is “ergonomic and user-friendly”
    • Note, however that the system may differ considerably from the experts (whose knowledge it attempts to represent) and from its users (if those are difference from the expert group). Still, the system tries to give users a comfortable feel, which may be achieved through symbols or words familiar to the user.

Gerhard Strube, The Role of Cognitive Science in Knowledge Engineering, 1992

Tradeoffs

• Syntax
  – How do we write things down?

• Expressivity
  – Ability of the language to distinguish between different concrete situations
  – If suitable to our needs, a formalism (or KR) is representationally adequate

• Computationall Complexity
  – Reasoning
  – How hard is it to work with?
    • Theoretical Complexity
    • Implementational Complexity
  – How hard is it to produce a production quality implementation

• Cognitive Complexity
  – Focus on Weak Cognitive Adequacy i.e., Usability
  – How hard is it to understand or comprehend?
  – How much effort does it take to express something?

• A good KR (or KR formalism) achieves a good balance of all of these for most of its uses, most of the time

General desiderata:
• Clarity of specification
• Expressivity
• Usability
• Computability
A Reasoning Perspective

- What expressivity do you need?
- What are your core service?
- What are the key services?
- Are you interactive or not?
- What's the scale you need to deal with?
  - And other performance characteristics
- What do you know about implementation?

- May neglect
  - Many surface syntax issues
  - Non logical aspects of the language
  - Cognitive complexity

Services

- Core
  - Satisfiability
  - Consistency Checking
- Key
  - Entailment
  - Classification (atomic subsumptions)
    - Atomic class satisfiability
  - Instantiation
  - Query/ASK

- Querying
  - Basic logical inference services insufficient
  - DB style query languages
  - Supporting applications
- Explanation
  - Why do concepts subsume?
  - Supporting ontology design process
- Non-Standard Inferences
  - Least Common Subsumer
  - Matching
  - Supporting ontology design process
Extra Logical Services

• It’s not just about logic!
  – We also need services that are not directed related to the underlying formal semantics of the representation

• Annotation Services
  – Associating information with concepts.
  – Facilitating the use of an ontology within an application
  – “Conceptual Coatrack”

• Lexical Services
  – Associating words, terms or symbols with concepts
  – Facilitating understanding and use in applications
  – Rendering definitions

Summary

• Tableaux Reasoning provides implementations for basic inference problems
  – Satisfiability
  – Subsumption
  – Classification

• Tableaux rules applied to try and build a treelike model of a concept (and thus demonstrate satisfiability)

• Further Reading: