Week 4

COMP62342
Sean Bechhofer, Uli Sattler
sean.bechhofer@manchester.ac.uk,
uli.sattler@manchester.ac.uk
Today

- Some clarifications around last week’s coursework
- More on reasoning:
  - extension of the tableau algorithm & discussion of blocking
  - traversal or “how to compute the inferred class hierarchy”
  - OWL profiles
- Snap-On: an ontology-based application
- The OWL API: a Java API and reference implementation
  - creating,
  - manipulating and
  - serialising OWL Ontologies and
  - interacting with OWL reasoners
- **Lab:**
  - OWL API for coursework
  - Ontology Development
Some clarifications around last week’s coursework
Ontologies, inference, entailments, models

- **OWL** is based on a *Description Logic*
  - we can use DL syntax
  - e.g., \( C \sqsubseteq D \) for \( C \text{ SubClassOf} D \)

- An OWL ontology \( O \) is a **document**:
  - therefore, it cannot **do** anything - as it isn’t a piece of software!
  - in particular, an ontology cannot **infer** anything
    (a reasoner may infer something!)

- An OWL ontology \( O \) is a **web document**:
  - with ‘import’ statements, annotations, …
  - corresponds to a **set of logical OWL axioms**
  - the OWL API (today) helps you to
    - parse an ontology
    - access its axioms
  - a **reasoner** is only interested in this set of axioms
    - **not** in annotation axioms
    - see https://www.w3.org/TR/owl2-primer/#Document_Information_and_Annotations
    - https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Annotations
Ontologies, inference, entailments, models (2)

- We have defined what it means for O to entail an axiom C SubClassOf D
  - written $O \models C \text{ SubClassOf } D$
  - or $O \models C \sqsubseteq D$
- based on the notion of a model $I$ of O
  - i.e., an interpretation $I$ that satisfies all axioms in O
- don’t confuse ‘model’ with ‘ontology’
  - one ontology can have many models
  - the more axioms in O the fewer models O has

- A DL reasoner can be used to
  - check entailments of an OWL ontology O and
  - compute the inferred class hierarchy of O
    - this is also known as classifying O
  - e.g., by using a tableau algorithm
Learn terms & meaning & relations!
More on Reasoning

Why?

To deepen our understanding of OWL

To understand better what reasoners do
Recall Week 2: OWL 2 Semantics: Entailments

Let O be an ontology, α an axiom, and A, B classes, b an individual name:

- O is **consistent** if there exists some model I of O
  - i.e., there is an interpretation that satisfies all axioms in O
  - i.e., O isn’t self contradictory
- O **entails** α (written O ⊨ α) if α is satisfied in all models of O
  - i.e., α is a consequence of the axioms in O
- A is **satisfiable** w.r.t. O if O ∉ A SubClassOf Nothing
  - i.e., there is a model I of O with A^I ≠ \{\}
- b is an **instance of** A w.r.t. O (written O ⊨ b:A) if b^I ⊆ A^I in every model I of O

**Theorem:**

1. O is consistent iff O ∉ Thing SubClassOf Nothing
2. A is satisfiable w.r.t. O iff O ∪ \{n:A\} is consistent (where n doesn’t occur in O)
3. b is an instance of A in O iff O ∪ \{b:not(A)\} is not consistent
4. O entails A SubClassOf B iff O ∪ \{n:A and not(B)\} is inconsistent
Let $O$ be an ontology, $\alpha$ an axiom, and $A, B$ classes, $b$ an individual name:

- $O$ is **consistent** if there exists some model $I$ of $O$
  - i.e., there is an interpretation that satisfies all axioms in $O$
  - i.e., $O$ isn’t self contradictory
- $O$ **entails** $\alpha$ (written $O \models \alpha$) if $\alpha$ is satisfied in all models of $O$
  - i.e., $\alpha$ is a consequence of the axioms in $O$
- $A$ is **satisfiable** w.r.t. $O$ if $O \not\models A$ SubClassOf Nothing
  - i.e., there is a model $I$ of $O$ with $A^I \neq \{}$
- $b$ is an **instance of** $A$ w.r.t. $O$ if $b^I \subseteq A^I$ in every model $I$ of $O$

- $O$ is **coherent** if every class name that occurs in $O$ is satisfiable w.r.t $O$
- **Classifying $O$** is a reasoning service consisting of
  1. testing whether $O$ is consistent; if yes, then
  2. checking, for each pair $A,B$ of class names in $O$ plus Thing, Nothing $O \not\models A$ SubClassOf $B$
  3. checking, for each individual name $b$ and class name $A$ in $O$, whether $O \not\models b:A$

  …and returning the result in a suitable form: $O$’s **inferred class hierarchy**
Week 3: how to test satisfiability …

Last week, you saw a **tableau algorithm** that

- takes a class expression \( C \) and decides **satisfiability** of \( C \)
- i.e., it
  - answers ‘yes’ if \( C \) is satisfiable
    - ‘no’ if \( C \) is not satisfiable
  - sound, complete, and terminating

- we saw this for the **\( ALC \)**
  - **\( ALC \)** is a Description Logic that forms logical basis of OWL
    - only and, or, not, some, only
  - works by trying to generate an interpretation with an instance of \( C \)
    - by breaking down class expressions (in NNF!)
    - generating new P-successors for some-values from restrictions
      (\( \exists P. C \) restrictions in DL)

- we can handle an ontology that is a set of **acyclic SubClassOf axioms**
  - via **unfolding** (check Week 3 slides!)
Week 3: tableau rules

\[ x \cdot \{ C_1 \sqcap C_2, \ldots \} \rightarrow \sqcap \quad x \cdot \{ C_1 \sqcap C_2, C_1, C_2, \ldots \} \]

\[ x \cdot \{ C_1 \sqcup C_2, \ldots \} \rightarrow \sqcup \quad x \cdot \{ C_1 \sqcup C_2, C, \ldots \} \quad \text{For } C \in \{ C_1, C_2 \} \]

\[ x \cdot \{ \exists R.C, \ldots \} \rightarrow \exists \quad x \cdot \{ \exists R.C, \ldots \} \quad y \cdot \{ C \} \]

\[ x \cdot \{ \forall R.C, \ldots \} \quad \text{R} \quad y \cdot \{ C, \ldots \} \quad \text{R} \]

Algorithm Examples

- Test the satisfiability of
  \[ 9 R.A \sqcup 8 R.B 9 R.A \sqcup 8 R.\neg A 9 R.A \sqcup 8 R.\neg A \sqcup 8 R.A \sqcup 8 R.\neg B \]
Mini-exercise

• Apply the tableau algorithm to test whether A is satisfiable w.r.t.

\[
\{ \text{A SubClassOf B and (P some C),} \\
\text{B SubClassOf C and (P only (not C or D))} \}
\]
This week: GCIs and tableau algorithm

- When writing an OWL ontology in Protégé,
  - axioms are of the form $A \text{ SubClassOf } B$ with $A$ a class name
  - (or $A \text{ EquivalentTo } B$ with $A$ a class name)
  - last week’s tableau handles these via unfolding:
    - works only for acyclic ontologies
    - e.g., not for $A \text{ SubClass } (P \text{ some } A)$

- but OWL allows for general class inclusions (GCIs),
  - axioms of the form $A \text{ SubClassOf } B$ with $A$ a class expression
  - e.g., $(\text{eats some Thing}) \text{ SubClassOf Animal}$
  - e.g., $(\text{like some Dance}) \text{ SubClassOf (like some Music)}$
  - this requires another rule:

\[
x \bullet \{\ldots\} \quad \rightarrow_{\text{GCI}} \quad x \bullet \{\neg C \sqcup D,\ldots\}
\]

for each $C \sqsubseteq D \in O$
E.g., test whether

A is satisfiable w.r.t.

\{A \text{ SubClassOf } (P \text{ some } A)\}

or \{A \sqsubseteq \exists P.A\}

This rule easily causes \textit{non-termination}

if we forget to \textit{block}
Blocking

- Blocking ensures termination
  - even on cyclic ontologies
  - even with GCIs
- If x’s node label is contained in the label of a predecessor y, we say “x is blocked by y”
- E.g., test whether A is satisfiable w.r.t. \{A \text{ SubClassOf} (P \text{ some } A)\}
  - here, n2 is blocked by n1
Blocking

- When blocking occurs, we can build a **cyclic** model from a complete & clash-free completion tree
  - hence soundness is preserved!

\[
\begin{array}{c}
\text{x} \bullet \{\exists R.C, \ldots \} \\
\rightarrow_{\exists} \\
R \\
y \bullet \{C\}
\end{array}
\]

only if x’s node label isn’t contained in the node label of a predecessor of x

\[
\begin{array}{c}
n1 \bullet \{A, \neg A \sqcup \exists P.A, \exists P.A\} \\
P \\
n2 \bullet \{A, \neg A \sqcup \exists P.A, \exists P.A\} \\
P
\end{array}
\]
Tableau algorithm with blocking

Our ALC tableau algorithm with blocking is

- **sound**: if the algorithm stops and says “input ontology is consistent” then it is.
- **complete**: if the input ontology is consistent, then the algorithm stop and says so.
- **terminating**: regardless of the size/kind of input ontology, the algorithm stops and says
  - either “input ontology is consistent”
  - or “input ontology is **not** consistent”

- …i.e., a decision procedure for ALC ontologies
  - even in the presence of cyclic axioms!
Tableau algorithm & complexity

Our ALC tableau algorithm has a few sources of complexity:

- the breadth/out-degree of the tree constructed
- the depth of/path length in tree constructed
- non-determinism due to \( \sqcup \) rule from
  - disjunctions in \( O \), e.g., \( A \text{ SubClassOf} B \)
  - \( \text{SubClassOf} \) axioms in \( O \)
  - \( \text{EquivalentTo} \) axioms in \( O \)

- \( \text{ok/linear for acyclic } O \)
- \( \text{bad/exponential for general } O \): we can construct \( O \) of size \( n \) where each model has a path of length \( 2^n \)

- hopefully not too bad

- 1 disjunction per axiom in \( O \) for each node in tree

- 2 disjunctions per axiom in \( O \) for each node in tree

- 37,778,931,862,957,161,709,568

- 3 nodes with 25 \( \text{SubClassOf} \) axioms \( \rightarrow \) how many choices?
Tableau algorithm & complexity

- Without further details: deciding ALC satisfiability
  - only of class expressions is PSpace-Complete
  - of class expressions w.r.t. ontology is ExpTime-complete
  - …much higher than intractable/SAT

- Implementation of ALC or OWL tableau algorithm requires optimisation
  - there has been a lot of work in the last ~25 years on this
  - you see the fruits in Fact++, Pellet, Hermit, Elk, … reasoners available in Protégé
  - some of them from SAT optimisations, see COMP60332

- Next, I will discuss 1 optimisation: enhanced traversal
Naive Classification

- **Remember:** Classifying $O$ is a reasoning service consisting of
  1. testing whether $O$ is consistent; if yes, then
    
    **Test:**
    is Thing satisfiable w.r.t. $O$?

  2. checking, for each pair $A, B$ of class names in $O$ plus Thing, Nothing whether $O \models A \text{ SubClassOf } B$
    
    **Test:**
    is $A \cap \neg B$ unsatisfiable w.r.t. $O$?

  3. checking, for each individual name $b$ and class name $A$ in $O$, whether $O \notmodels b:A$
    
    **Test:**
    is $O \cup \{b: \neg A\}$ is inconsistent?

…and returning the result in a suitable form: $O$’s *inferred class hierarchy*
Naive Classification

- **Remember:** Classifying \( O \) is a reasoning service consisting of

1. testing whether \( O \) is consistent; if yes, then
   
   Test:
   - is Thing satisfiable w.r.t. \( O \)?

   1 test

2. checking, for each pair \( A, B \) of class names in \( O \) plus Thing, Nothing whether \( O \models A \text{ SubClassOf } B \)
   
   Test:
   - is \( A \sqcap \neg B \) unsatisfiable w.r.t. \( O \)?

   \( n^2 \) tests for \( O \) with \( n \) class names

3. checking, for each individual name \( b \) and class name \( A \) in \( O \), whether \( O \models b:A \)
   
   Test:
   - is \( O \cup \{b: \neg A\} \) is inconsistent?

   \( nm \) tests for \( O \) with \( n \) class names, \( m \) individuals

...and returning the result in a suitable form: \( O \)'s **inferred class hierarchy**
Enhanced Traversal

- **Naive Classification of O** requires $1 + n^2 + nm$ expensive satisfiability/consistency tests
- …can we do better?
  ➡ Enhanced Traversal
  - idea: build inferred class hierarchy top-down and bottom-up, “trickling in” each class name in turn

- Assume you have, so far, constructed this hierarchy for O

- Now you “trickle” Oak: check whether
  - $O \not\models Oak \subseteq \text{Plant}$?
    yes - continue with Plant’s child
  - $O \not\models Oak \subseteq \text{Animal}$?
    no - ignore Animal’s children!
  - $O \not\models Oak \subseteq \text{Tree}$?
    yes - done!
  ➡ 2 entailment tests saved!
Enhanced Traversal

- **Naive Classification of O** requires $1 + n^2 + nm$ expensive satisfiability/consistency tests
- ...can we do better?
  ➡ Enhanced Traversal
  - idea: build inferred class hierarchy top-down and bottom-up, “trickling in” each class name in turn

- Potentially avoids many of the $n^2$ satisfiability/consistency tests
  - very important in practice
  - different variants have been developed

- Just one of many optimisations!
OWL Profiles

Restrictions of OWL to tame complexity
OWL Profiles

- Despite all optimisations, classification of an ontology may still take too long if it is
  - big and/or
    - 300,000 axioms or more
  - rich
    - ALC plus inverse properties, atleast, atmost, sub-property chains,…

- For OWL 2 [*], profiles have been designed
  - syntactic fragments of OWL obtained by restricting constructors available

- Each profile is
  - maximal, i.e., we know that if we allow more constructors, then computational complexity of reasoning would increase
  - motivated by a use case

[*] the one we talk about here/you use in Protégé
OWL Profiles

OWL 2 has 3 profiles, roughly defined as follows:

- **OWL 2 EL:**
  - only ‘and’, ‘some’, SubProperty, transitive, SubPropertyChain
  - it’s a *Horn* logic
    - no reasoning by case required,
    - no disjunction, not even hidden
  - designed for big class hierarchies

- **OWL 2 QL:**
  - only restricted ‘some’, restricted ‘and’, inverseOf, SubProperty
  - designed for querying data in a database through a class-level ontology

- **OWL 2 RL:**
  - no ‘some’ on RHS of SubClassOf, …
  - designed to be implemented via a classic rule engine

- For details, see OWL 2 specification!
Some Key Complexity Classes

- FO
- Predicate Logic
- OWL 2
- Propositional Logic
- All Problems
- Semi-Decidable
- Decidable
- NP
- P
- OWL 2 EL
The design triangle

Expressivity
(Remresentational Adequacy)

Usability
(Weak Cognitive Adequacy vs. Cognitive Complexity)

Computability
(vs. Computational and Implementational Complexity)
OWL reasoning

- is unusual:
  - standard reasoning involves solving many reasoning problems/satisfiability tests
- is decidable:
  - for standard reasoning problems, we have decision procedures
  - i.e., a calculus that is sound, complete, and terminating
- can be complex
  - but we know the complexity for many different DLs/OWL variants/profiles
  - and implementations require many good optimisations!

- goes beyond what we have discussed here
  - entailment explanation
  - query answering
  - module extraction
  - ...

Summary
Today

✓ Some clarifications from last week’s coursework
✓ More on reasoning:
  ✓ extension of the tableau algorithm & discussion of blocking
  ✓ traversal or “how to compute the inferred class hierarchy”
✓ OWL profiles
  • Snap-On: an ontology-based application
  • The OWL API: a Java API and reference implementation
    • creating,
    • manipulating and
    • serialising OWL Ontologies and
    • interacting with OWL reasoners
✓ Lab:
  • OWL API for coursework
  • Ontology Development