Hoolet: An OWL Reasoner with Support for Rules

Sean Bechhofer, Ian Horrocks

University of Manchester

http://owl.man.ac.uk/hoolet

Reasoning with OWL

- OWL DL has a “standard” first-order style semantics
- This allows us to use known results from Description Logic research to build reasoners for OWL
  – FaCT, RACER, Pellet
- However, the expressiveness of “full” OWL DL causes some problems
  – Currently no know effective algorithms in the presence of cardinality, inverses and enumerations
  – Reasoners such as FaCT and RACER “pretend” to handle one-of.
- Can we use alternative reasoning engines?

OWL and First Order Reasoning

- An alternative approach is to translate OWL DL into equivalent FOL axioms and then use a FO prover to provide inference
- Disadvantages
  - In general this compromises decidability, although a FO reasoner may be able to apply a complete strategy.
  - DL reasoners have been specifically optimised to handle DL style reasoning tasks. FO reasoners may require extra tuning to handle the tasks created.
- Advantages
  - Can handle all of OWL DL
  - Can be extended to deal with language extensions such as SWRL.

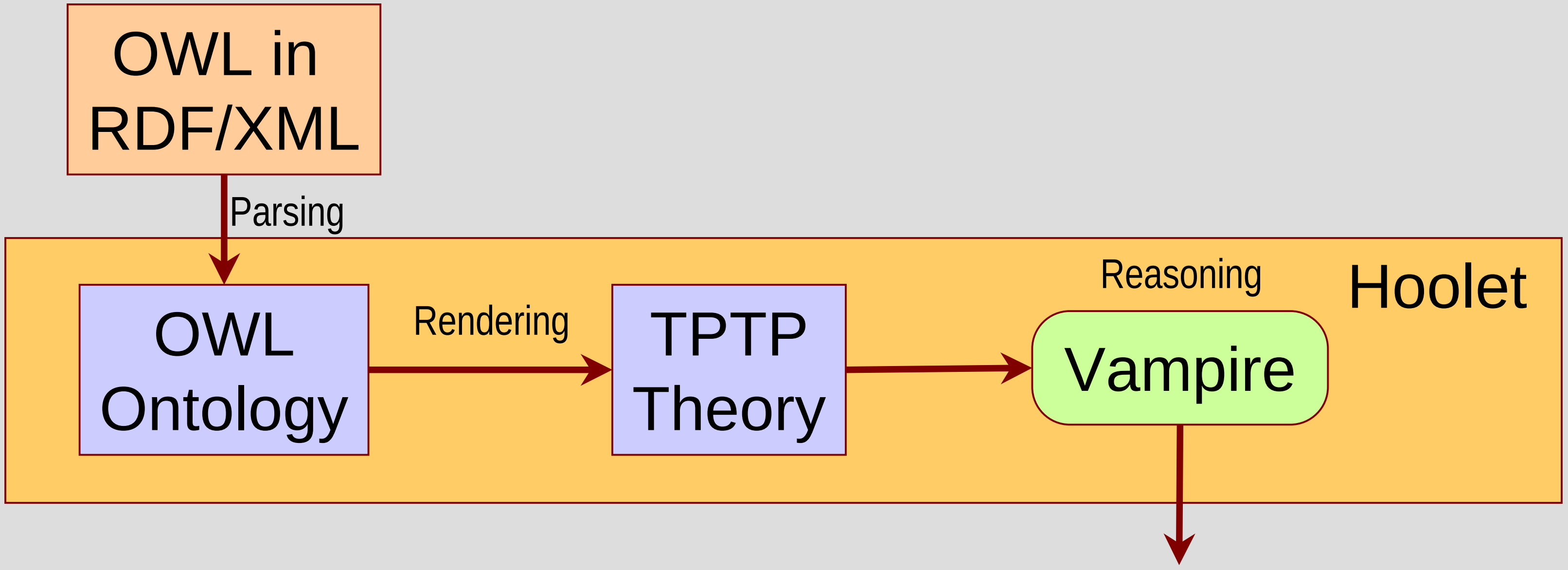
Hoolet

- A (prototype) OWL reasoner using a First Order prover.
- OWL ontology translated to equivalent axioms using the standard TPTP format.
- Axioms then passed to Vampire for satisfiability testing.
- Queries are translated to conjectures which are added to the theory.
- Hoolet may not be a very **effective** reasoner
  - This naive approach is not likely to scale well.
- However, it does provide a useful tool for use on small illustrative examples.
  - And may form part of an effective reasoning infrastructure

Example Translations

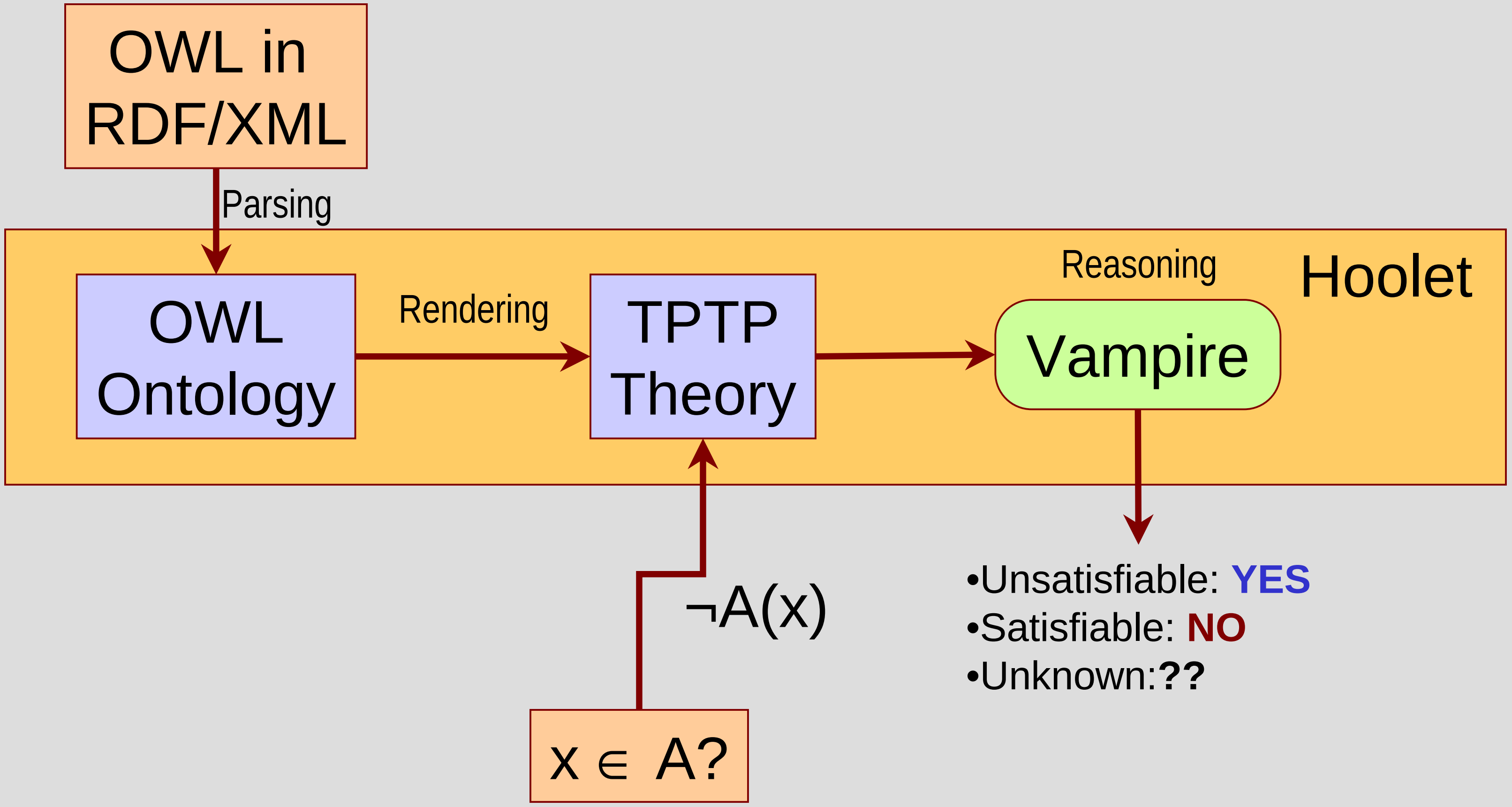
Class(B complete A)	8 x.A(x) $ B(x)
SubClassOf(intersectionOf (A B) unionOf(C D))	8 x.(A(x) u B(x)) ! (C(x) t D(x))
Class(B partial restriction(p someValuesFrom A))	8 x.B(x) ! (9 y.A(y) u p(x,y))
Class(A complete one-of(a b c))	8 x.A(x) $ (x=a t x=b t x=c)

Satisfiability Testing



- Unsatisfiable: **NO**
- Satisfiable: **YES**
- Unknown: **??**

Query



- Unsatisfiable: **YES**
- Satisfiable: **NO**
- Unknown:**??**

Rules

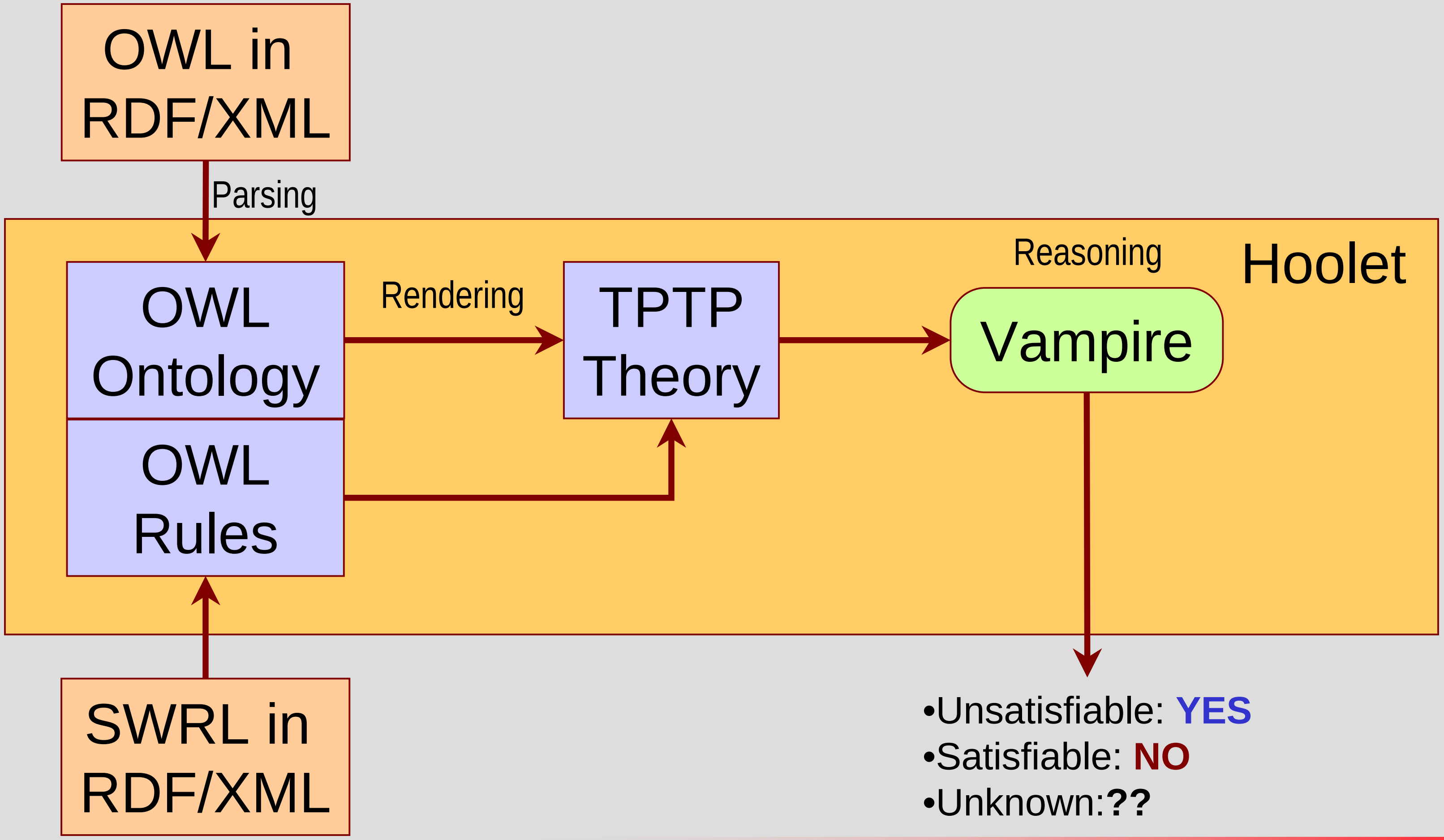
- It is easy to extend Hoolet to handle SWRL rules.
- Each rule is simply translated to an axiom according to the semantics of the rules, with free variables universally quantified.

  hasParent(?x,?y), hasSibling(?y,?z),male(?z) $\rightarrow$ hasUncle(?x,?z)

  translates to:

  $\forall$x,y,z.hasParent(x,y)$\wedge$hasSibling(y,z) $\wedge$male(z) $\rightarrow$ hasUncle(x,z)
- Rules are then added to the theory.

Adding Rules



- Unsatisfiable: **YES**
- Satisfiable: **NO**
- Unknown:**??**

Hoolet Application

- Hoolet supplies a simple GUI for loading ontologies and rules
  - Uses **WonderWeb OWL API** for parsing and representation.
  - (Ab)uses **Vampire** prover for reasoning.
- Ontologies should be represented using OWL in RDF/XML
- Rules are represented using a (possibly idiosyncratic) RDF schema.
  - Restrictions on rule atoms: only classes allowed.
- Simple Queries:
  - satisfiability
  - subsumption
  - retrieval.
- Prototype from **http://owl.man.ac.uk/hoolet**