

Kruthi Bhoopalam and Volker Haarslev Concordia University, Montreal, Quebec, Canada

Introduction

Fire is a DL-based rule engine for OWL ontologies extended with SWRL-like rules.

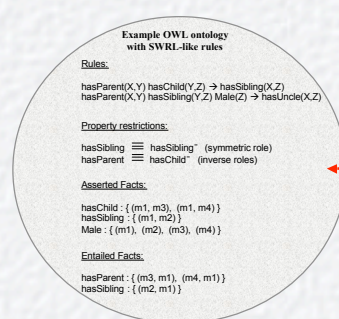
The Fire system,

- is implemented in Java.
- intends to extend RACER's DL reasoning for rules.
- supports a *SWRL-like* rule language, treating rules as *true rules* and not as implications.

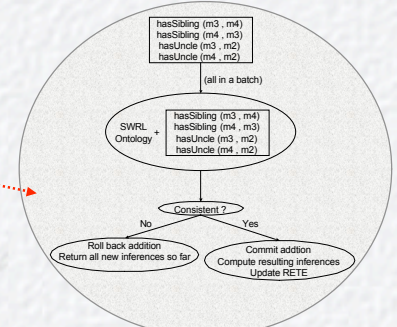
The current implementation uses

- nRQL (new Racer Query Language) queries and services of RACER externally using JRacer Java API.
- Protégé SWRL Factory Java API.

Fire System Illustration with Example



The green arrow shows the architecture of the Fire system. The pattern matcher component of Fire processes the OWL ontology and SWRL-like rules that are read correspondingly from RACER and the Protégé SWRL Factory API; the rule application component processes the new instantiations of triggered rules returned from the pattern matcher and applies them to the ontology. Thus rules get fired. Details of the system components are shown in corresponding circles pointed to by red arrows.



The SWRL-like Rule Language

Syntax

The SWRL-like rule syntax follows the SWRL rule syntax¹.

Further restrictions on the syntax

These restrictions are further imposed on the syntax for ease of implementation.

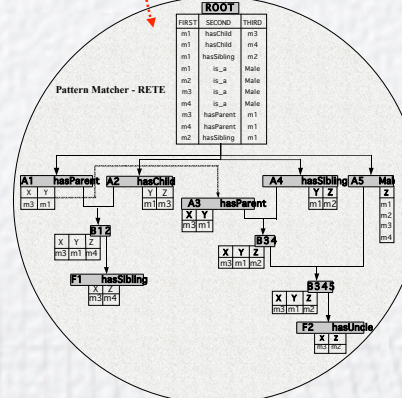
- Rules must be atomic (single atom in rule consequent)
- Rules must be 'strictly Horn' (predicates of rule atoms may only be name of property or class)
- SWRL built-ins, sameAs, differentFrom and OWL DatatypeProperty predicates are not allowed in rule atoms
- Negated concept/role atoms are not allowed in rules

Semantics

- Rules are *true rules* (i.e., if body holds then head must hold) and not implications.
- The instantiations obtained for triggered rules are considered only if they are consistent with all the triggered rules.
- Rules are not allowed to create new individuals in the KB. The reasoning support follows the *active domain semantics*, i.e., the variables in rules are bound only to explicitly named individuals in the knowledge base (KB) and not to the arbitrary individuals encountered during the proof.

¹ <http://www.daml.org/2003/11/swrl/>

The circle on the right shows a detailed view of the pattern matcher component of Fire. This is implemented by the RETE algorithm. The algorithm constructs a network of nodes based on patterns in the rules. Asserted and entailed facts in the KB are filtered into this network. The facts reaching the final nodes (here F1 and F2) are the resulting instantiations of the triggered rules.



The circle above shows the detailed working of the rule application component of Fire. New instantiations are identified from the pattern matcher's response and are asserted to the ontology together as a *batch*. If the resulting KB is still consistent; assertions made earlier in this cycle are committed as inferences, the RETE is updated and the pattern matcher begins another cycle to find instantiations for triggered rules. If the resulting KB is inconsistent; the assertions made earlier in this cycle are rolled back and the system exits returning the new inferences obtained so far. When there are no new inferences in RETE's response, the system exits returning the new inferences obtained so far.

Experimental Results

The RETE algorithm performed more than 20 times faster than the implemented naive pattern matching algorithm. This table gives the results of our comparison experiments. It considers the known number of facts before the algorithm is run and the number of new inferences fired from the algorithm.

Rules	Initial no. of facts	No. of newly fired inferences	Time with naive (sec)	Time with RETE (sec)
80	584	991	6.875	21.693
1149	1301	2486	3.811	115.488
1775	2127	5000	21.514	388.261
2280	2804	8461	26.011	524.118
2726	3229	10154	36.469	762.172

Future Directions and Open Issues

Future Directions

- Extending support for OWL DatatypeProperty, sameAs and differentFrom predicates in rule atoms.
- Optimize construction of RETE pattern network by exploiting common predicates among atoms of different rules.
- Use RACER's publish-subscribe mechanism to efficiently learn about changes made to any concept.
- Extending support for rules with SWRL semantics.
- Providing Fire as a plug-in for Protégé's SWRL Tab.
- Integration of the RETE algorithm implementation into RACER for reasoning with nRQL rules.

Open Issues

- To support reasoning for SWRL rules, it is interesting to explore an approach similar to the CARIN algorithm.
- Derive all possible *completions* of the KB using a DL reasoner.
 - Fire rules for every clash-free completion derived.
 - A non-contradicting result of any clash-free completion with fired rules determines an answer.
- Expressiveness supported – what fragment of SWRL can be practically supported?...still unclear.