Abstract

Memory Efficient Stream Reasoning on Resource-Limited Devices

by Dylan Armstrong

Supervisor: Declan O'Sullivan Assistant Supervisor: Wei Tai

Master in Computer Science (MCS)

2014

In sensor rich systems standard approaches usually attempt to perform reasoning on a centralised machine, sending data from sensor devices through the web or by other wired or wireless technologies. However, in some conditions this may not be feasible, or may be restrictive. In these cases reasoning can be performed on the sensor device nodes themselves. For this reason, the reasoning process must be memory efficient.

Previous work has seen the adaption of a static RETE based reasoner (COROR) into a stream based reasoner (SCOROR). A streaming environment is commonly the sort of environment that memory constrained sensor devices are used in. This research focuses on finding an alternative reasoning algorithm for SCOROR that could offer improved memory consumption. A comparison of the RETE, TREAT and LEAPS reasoning algorithms is offered, with LEAPS proving the most promising for improved memory consumption.

An in-depth design and implementation of the LEAPS reasoner is given, along with the steps needed to modify the algorithm in order to support stream reasoning.

An evaluation of this reasoner found that the reasoning times for both RETE and LEAPS reasoners were quite similar. The RETE reasoner outperformed the LEAPS reasoner when there was a combination of a large ontology size with high expressivity. The suggested reasons for this are the use of wild-card predicates in rules and the resetting of saved iteration states due to the removal of temporal triples. Memory consumption for the LEAPS reasoner was shown to be significantly reduced in comparison to the RETE reasoner.

This research shows that the LEAPS algorithm can be implemented and modified to perform in a streaming environment. Evaluation suggests that it is indeed the case that a more efficient option in terms of memory consumption over the RETE implementation exists in the form of this implementation.