Abstract Syntax and Semantics: Slots & Constraints

Hassan Aït-Kaci
Harold Boley
Michael Kifer
Peter Patel-Schneider
Chris Welty

W3C RIF WG Breakout, F2F4, Athens, GA, 5 Nov 2006

Motivation for Slots and Constraints

Slots:

- Integration with RDF (properties) and Java (fields)

Constraints:

- Integration with Description Logics (e.g., OWL1.1 DL)
- Interfacing Built-ins (e.g., Functions & Operators)
- Interfacing Java Types

Herbrand plus Slotted Terms

- Currently: Herbrand terms (positional arguments): **f(a1, …, an) – viewed as shorthand for f{1->a1, …, n->an}**
- Equational Constraint abstraction (dot notation): **f.1=a1, …, f.n=an** (argument positions as keys)

- Charter: Add slotted terms (keyed arguments): (user keys k1, …, kn, not necessarily distinct): **f{k1->a1, …, kn->an}**
- Equational Constraint abstraction (dot notation): **f.k1=a1, …, f.kn=an**

Generalized Syntax: From Rulebases to Slots

Rulebase

annotation (e.g., in RDF)
- name mapping

clause

Rule

annotation

quantifer *head* *body* *constraint*

Forall Atom Atom | And Formula

Atom*

HerbrandAtom | SlottedAtom

HerbrandAtom
Relation (HerbrandTerm)*
Constant |
Variable |
HerbrandExpression
Function (HerbrandTerm)*

SlottedAtom
Relation {SlottedTerm}*
Key->Constant |
Key->Variable |
Key->SlottedExpression
Function {SlottedTerm}*

Generalized Syntax: Clauses

- Currently: Unconstrained Horn Clauses
  ∀ H :- B1, …, Bm
- Generalization: Constrained Horn Clauses (variables are shared across Bi’s and optional C):
  ∀ H :- B1, …, Bm [& C]
- In Core: Constraint C is conjunction of equations with ground dot-notation terms on left-hand side
- In dialects: Constraint C can be any formula (abstract ‘oracle’ point of view: don’t care how constraint solver can solve C)

Semantic Hierarchy

RIF Core: Based on current model theory,
with suitable extensions

RIF Standard Dialects:

- If a dialect has a **model theory**, then this is normative
  - Any proof theory and operational semantics must respect the model theory
- If a dialect does not have a model theory but has an explicit **proof theory**, then this is normative
  - Any operational semantics must respect the proof theory
- If a dialect does not have a proof theory but has an **operational semantics**, then this is normative (e.g., expressed as a pseudo-coded algorithm)