Scalability and Query-Orientation in Hybrid Knowledge Bases

Terrance Swift
CENTRIA – Universidade Nova de Lisboa

July 9, 2011

Preface

- If there are N different formalisms for a KB
- There are up to 2^N different ways to combine these knowledge bases
- Therefore I'll mostly focus on certain aspects of hybrid KBs: combining WFS-based rule systems with Description Logics (DLs) and how to make them suitable for practical use.

Outline

- Introduction: What is a hybrid knowledge base and why/when/how do we use one?
- The description logics $\mathcal{ALC}$ and $\mathcal{EL}^+$
- The WFS-based rule based systems XSB/CDF, Flora-2, Silk
  - These systems extend rules to have simple aspects of DL reasoning
- The basics of MKNF (a way to combine rules and DLs)
- Combining XSB/CDF with $\mathcal{ALC}$ via MKNF
- Tightly combining XSB/CDF with $\mathcal{EL}^+$ via MKNF through **SLG($\mathcal{O}$)**
- Adding evidential probabilities to an MKNF KB
- Some questions about hybrid KBs.

What is a Hybrid KB?

Hybrid KBs can mean different things:

- Semantically Hybrid. Combining information from different sources using name spaces, paraconsitency.
- Computationally Hybrid: Stratified KBs
  - ASP calling DBs – DLV
  - Prolog calling ASP – XASP
  - DLs compiled into ASP – early versions of CDF, KAON-2
- Semantically and Computationally Hybrid
  - Rules calling first-order logic calling rules calling first-order logic
  - Need to address closed-world vs. open-world reasoning, issues of decidability.
  - Can be addressed through Minimal Knoweldge with Negation as Failure (MKNF) as well as through Multi-Context systems and other approaches.

Why use a Hybrid KB?

- Sociological Reasons
  - There is a communitiy studying various description logics and using OWL-DL
  - There is a community studying rules under various semantics
- Belief in Inherent strengths or weaknesses of various KR formalisms
- As a way to understand the strengths and weaknesses of various KR formalisms

But what are these strengths and weaknesses? When do we need to combine KR formalisms?

Why use a Hybrid KB?

Logic-based rules have their advantages

- *Decidability* Both formalisms do/can restrict programs to be decidable, though perhaps with a loss of expressivlty and convenience (e.g. lists).
- *Deductive Power*: ASP systems can draw powerful inferences from a knowledge base.
- *Scalability* WFS based systems offer good scalability (they generally behave linearly in practice).
- *Inconsistency Checking* Inconsistencies may be detected in both formalisms (though ASP offers stronger mechanisms)
- Formalisms for non-monotonic and explicit negation
- Fixed-point inference, e.g. transitive closure.

Why use a Hybrid KB?

First-order logic has its strengths

- *Decidability* Restricted subsets of first-order logic are decidable. We'll be sloppy and call these subsets DLs here.
- *Deductive Power*: DLs based on $\mathcal{ALC}$ and its extensions can draw powerful infrecnes from knowledge bases.
- *Scalability* Low-complexity DLs such as $\mathcal{EL}^+$ and DL-Lite can be scalable.
- *Inconsistency Checking* Logical inconsistencies may be checked.
- DLs offer open-world negation
- Knowledge engineers may specify knowledge rather than program it.
- Ontology editors such as Protege provide a good IDE for KE use

Description Logics: $\mathcal{ALC}$ (cf. [BCM+03])

- Axioms of the form
  - $CE_1 \sqsubseteq CE_2$
  - $CE_1 \equiv CE_2$

  where CE_1, CE_2 are class expressions.
- A *class expression*, C, is either an atomic class name in $\mathcal{L}_D^{\mathcal{A}_C}$ or is inductively formed by one of the following constructions in which a is an atomic class name, C_1 and C_2 class expressions, R a relation, and n a natural number.

$$C \leftarrow a|\top|\neg C_1|C_1 \sqcap C_2|(\exists R.C_1)$$
$$+$$
$$C \leftarrow \bot\ |C_1 \sqcup C_2|(\forall R., C_1)$$

Description Logics: $\mathcal{ALC}$

Often given in special syntax:

$$Grandmother \equiv (Person \sqcap Female) \sqcap (\exists hasChild\ (Person \sqcap (\exists hasChild\ Person)))$$

Equivalent to 2-variable formulas in FOL with unary predicates and binary relations.

$$\forall X[Grandmother(X) \equiv (Person(X) \wedge Female(X)) \wedge \exists Y[hasChild(X, Y) \wedge Person(Y) \wedge \exists \mathbf{X}[hasChildPerson(Y, \mathbf{X})]]]$$

- $\mathcal{ALC}$ can be extended with other features, such as counting $\exists > 3X$, role inheritance, role inverses, etc. Reasoners for various formulations of OWL can be written using extensions of $\mathcal{ALC}$.
- $\mathcal{ALC}$ is PSPACE-complete; extensions can be det-EXPTIME or worse
- DLs distinguish between a T-box (axioms about classes and roles) and an A-box (assertions about individuals and their relations to other individuals or classes).

Description Logics: $\mathcal{EL}^+$ [BBL05]

C, D are general concepts, $r(_i)$ are primitive roles.

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\exists$-restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
GC Inheritance	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
R Inheritance	$r_1 \circ \cdots \circ r_k \sqsubseteq r$	$r_1^{\mathcal{I}} \circ \cdots \circ r_k^{\mathcal{I}} \sqsubseteq r^{\mathcal{I}}$
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$r(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}})$

- Unlike $\mathcal{ALC}$ no $\neg$ (hence no reasoning by cases); but does have R Inheritance.
- Can express role transitivity $r \circ r \sqsubseteq r$; disjointness of classes $C \sqcap D \sqsubseteq \bot$, etc.
- Subsumption can be computed in time quadratic to the number of primitive classes [BBL05].

Prolog as a KB formalism?

Prolog systems offer lots of nice features

- Computation of WFS. As discussed below, this can be useful for inconsistency detection and for stating preferences.
- Call subsumption. Reduces (but does not obviate) need for optimizing compiler to re-order literals, refactor recursive forms.
- Answer subsumption – implementational basis for multi-valued logics, reasoning under uncertainty in the PITA package (cf. talk in main conference). Can be used for optimality problems.
- Incremental tabling – allows tables to be maintained in the presence of updates.
- Constraint Libraries
- Large set of options for indexing and adaptive indexing
- Probability Packages
- IDE for programmers

However, no single system offers all these features.

Prolog as a KB formalism?

Prolog systems are lacking

- Optimizing Compilers (though Ciao comes closest)
- No GUI interface for KEs.
- No built-in inheritance (which KEs usually want)

Rule-Based Systems: Flora-2 [YKWZ11][1]

- Provides rules for programming about both a classes and objects they contain. Allows monotonic (DL-like) and non-monotonic (OO-like) inheritance
- Schema notation enforces types and cardinality constraints for relations among individuals
- Uses Hilog for higher-order reasoning, and integrates HiLog with a module system
- Primitive data types include those of Prolog a W3C compliant iri data type; XML dates, times, durations, and others.
- Can allow equality reasoning (currently being reimplemented)
- Allows backtrackable updates and transactions. Updates may be propagated through argumentation theories via incremental tabling
- Manages tabling for users
- Allows direct calls to Prolog and is implemented using (every single friggin' feature of) XSB

[1] Due to legal reasons, newer versions of Flora-2 have not yet been put onto sourceforge – stay tuned.

Rule-Based Systems: Flora-2

Schema:

paper[authors => person, title => string].
journal_p :: paper[in_vol => volume].
conf_p :: paper[at_conf => conf_proc].
journal_vol[of => journal, volume => integer, number => integer, year => integer].
journal[name => string, publisher => string, editors => person].
conf_proc[of_conf => conf_series, year => integer, editors => person].
conf_series[name => string].
publisher[name => string].
person[name => string, affil(integer) => institution].
institution[name => string, address => string].

Objects:

o_j1 : journal_p[title -> 'Records, Relations, Sets, Entities, and Things',
authors -> {*o_mes*}, in_vol -> *o_i11*].
o_di : conf_p[title -> 'DIAM II and Levels of Abstraction',
authors -> {*o_mes*, *o_eba*}, at_conf -> *o_v76*].
o_i11 : journal_vol[of -> *o_is*, number -> 1, volume -> 1, year -> 1975].
o_is : journal[name -> 'Information Systems', editors -> {*o_mj*}].
o_v76 : conf_proc[of -> vldb, year -> 1976, editors -> {*o_pcl*, *o_ejn*}].
o_vldb : conf_series[name -> 'Very Large Databases'].
o_mes : person[name -> 'Michael E. Senko'].
o_mj : person[name -> 'Matthias Jarke', affil(1976) -> *o_rwt*].
o_rwt : institution[name -> 'RWTH_Aachen'].

Rule-Based Systems: Flora-2

Uses WFS extended with explicit negation and defeasable reasoning

- WFS is used to determine priorities for non-monotonic inheritance
- Defeasible reasoning is based on specifiable notions of *opposition* and *overriding* among rules.
- Defeasible reasoning is parameterized by various argumentation theories [WGK+09] that can express notions of refutation, rebuttal, and disqualification of rules
- Flora-2 rules may be labelled, and a defeasible rule labelled *r* has the literal `not defeated(r)` added to its body.

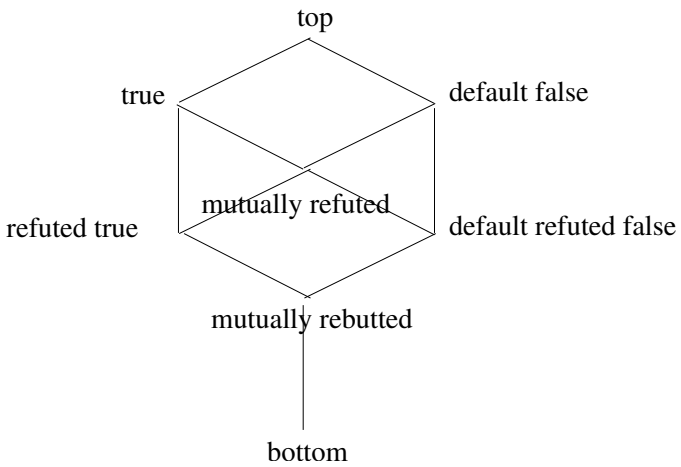
Rule-Based Systems: Flora-2

```
:- table defeated/1.
defeated(A):- defeated_by(A,_B).        defeated(A):- defeats(A,_B).

defeated_by(A,B):- refutes(B,A),B.      defeats(A,B):- refutes(A,B),B.
defeated_by(A,B):- rebuts(B,A),B.       defeats(A,B):- rebuts(A,B),B.

refutes(A,B):- conflicts(A,B), overrides(A,B).
rebuts(A,B):- conflicts(A,B).

conflicts(A,B):- opposes(A,B),A.        conflicts(A,B):- opposes(B,A),A.
```



Rule-Based Systems: Silk

- Currently under development by Vulcan, Inc.
- Currently implemented using JFlora-2, Java and Interprolog
- Parsing and some I/O is done in Java to make use of packages that implement W3C and other standards.
- Extends Flora-2 with First-order-like "Omni Rules"
- Includes a sophisticated Eclipse-based IDE with Silk-specific editors and a justification system.

Rule-Based Systems: CDF

- Open-source XSB package for ontology management. Developed by XSB, Inc and heavily used in their products and services. Allows Type-0 and Type-1 ontologies [2]
- Type-0 ontologies allow class and role inheritance, conjunction, existential and universal quantification but no disjunction or negation.

$$C \leftarrow a|\top|C_1 \sqcap C_2|(\exists R.C_1)|(\forall R.C_1)$$

- No $\neg$; no $\bot$. No reasoning by cases and no contradiction.

[2] Type-0 ontologies are joint work with David S. Warren.

Rule-Based Systems: CDF

Example of a Type-0 ontology. Note that all classes are named.

$Man \sqsubseteq Person \sqcap Male$

```
isa(cid(man),cid(person))
isa(cid(man),cid(male))
```

$Husband \sqsubseteq Man \sqcap \exists Spouse.Person$

```
isa(cid(husband),cid(man))
hasAttr(cid(husband),rid(spouse),cid(person))
```

$adam : Husband$

```
isa(oid(adam),cid(husband))
```

Rule-Based Systems: CDF

- CDF supports namespaces, product classes (to encode non-binary relations), database access, and a Swing front-end through XJ. This allows KE programming to some extent.
- CDF predicates can be defined via extensional facts or intensional rules, supporting queries to DBMSs, GUI models, etc. The semantics of intensional rules is outside of CDF.
- XSB's alternate argument, multi-argument and star indexing are employed to provide efficient access of facts.
- Tabling with negation is used to navigate the *isa* hierarchy efficiently and to return the most specific answers when monotonic inheritance is used.
- Type-0 ontologies typically can contain T-boxes with 10,000-100,000 classes, and A-boxes about 10,000,000 or more individuals.

So Type-0 ontologies essentially allow KEs to program statements about classes within an inheritance hierarchy and about properties of these classes.

Rule-Based Systems: CDF

- Type-1 ontologies provide full $\mathcal{ALCQ}$ deduction. Other CDF system features are preserved.
- Queries have a form such as `allModelsEntails(Term,ClassExpr)` to determine whether *Term* $\sqsubseteq$ *ClassExpr* is provable by the ontology.
- DL deduction in Type 1 ontologies was first implemented using XASP [Swi04]. For reasons of scalability, this was abandoned in favor of a traditional tableau prover which is now in use.

The basics of MKNF

- MKNF allows a DL to be combined with rules under the stable model or well-founded semantics [MR07, KAH08].
- Properties and relations of individuals may be computed using a (potentially recursive) combination of T-box axioms, A-box assertions and rules.
- Literals in rules may be atoms, or be prepended with the *not* or K operators. K operators might be thought of as "default true".
- Formal presentation of MKNF requires a committment to notation: I'll just present the basic ideas here.

The basics of MKNF: Example [MR07]

$$NonMarried \equiv \neg Married$$
$$\neg Married \sqsubseteq HighRisk$$
$$\exists Spouse.\top \sqsubseteq Married.$$

NonMarried and *Married* are equivalent concepts; anyone who not *Married* is *HighRisk*; anyone with a *Spouse* is *Married*.

$$\mathbf{K}\ nonMarried(X) \leftarrow person(X), \mathbf{not}\ married(X).$$
$$surcharged(X) \leftarrow person(X), \mathbf{K}\ highRisk(X)..$$

A *person* is *nonMarried* if that person is not known to be *Married*;
A *HighRisk person* is *Surcharged*.

The Basics of MKNF

Suppose we add $person(john)$ to the KB.

- by r1, we get $nonMarried(john)$;
- by a1) we get $\neg Married(john)$, by a2) we get $HighRisk(john)$;
- by r2) we get $surcharged(john)$

Rules and Axioms combine to give us this conclusion.

$a1)$ $NonMarried \equiv \neg Married$

$a2)$ $\neg Married \sqsubseteq HighRisk$

$a3)$ $\exists Spouse.\top \sqsubseteq Married$

$r1)$ $\mathbf{K}\ nonMarried(X) \leftarrow person(X), \mathbf{not}\ married(X).$

$r2)$ $surcharged(X) \leftarrow person(X), \mathbf{K}\ highRisk(X).$

The Basics of MKNF

Intuitively, for definite programs we construct an iterated fixed point:

$$
\begin{array}{ll}
D_0 = \mathit{Tableau}(\mathcal{O}) & R_0 = \mathit{WFS}(\mathcal{P}) \\
D_1 = \mathit{Tableau}(\mathcal{O} \cup R_0) & R_1 = \mathit{WFS}(\mathcal{P} \cup D_0) \\
\ldots & \ldots \\
D_n = \mathit{Tableau}(\mathcal{O} \cup R_{n-1}) & R_n = \mathit{WFS}(\mathcal{P} \cup D_{n-1})
\end{array}
$$

- **K** means provable based on $\mathcal{P}$ and DL-deduction from the previous iteration

Combining Tabling with $\mathcal{ALC}$ via MKNF [3]

- For scalability, we want a goal-oriented version of MKNF
- This was implemented using XSB/CDF: rules are goal-oriented under WFS; DL-reasoning is goal-oriented to the extent that the tableau prover constrains its search
- Since the tableau prover is written in Prolog, rules can call the prover; since CDF information can be based on intensional rules, the prover can call rules.
- Rules must be *DL-safe*: calls to DL-atoms must first be ground by rule atoms. For instance, `person(X)` grounds calls to **K** `married(X)` and **K** `highRisk(X)`
- We describe research prototype implementation called *CDF Rules*.

[3] Joint work with Sophia Gomes and Jose Alferes [GAS11].

Combining Tabling with $\mathcal{ALC}$ via MKNF

- Consider first definite programs plus DLs.
- When we call `surcharged(john)` the system first computes a fixed point for all properties and relations of all individuals
  - encountered during rule evaluation
  - encountered during tableau construction.

  I.e., a fixed point is construacted for a (dynamically changing) set of queries.
- This fixed point is determined using tabling

Combining Tabling with $\mathcal{ALC}$ via MKNF

To determine class properties in programs without negation in rules

```
/* A is a unary term: e.g. p(indiv) for some individual */
known(A):- computeFixedPoint(A), /* if necessary */
           known_1(A).
```

- `computeFixedPoint(p(indiv))` makes a query to the rule system and ontology about properties of `indiv` and related individuals.
- Specifically, calls `known(Atom,Iter)` and `allModelsEntails(Atom,Iter)` for `Iter` $= 0, \ldots, N$
- Fixed point introspecting the tables at the end of each iteration.
- `known_1(A)` then just checks the table for `A` in the final iteration

Combining Tabling with $\mathcal{ALC}$ via MKNF

How the rules call the ontology

```
:- table known/2, allModelsEntails/2.
known(A,Iter):-
   (    call(A),
        /* Check last iteration of DL */
      ; Iter > 0,PrevIter is Iter - 1,
        allModelsEntails(A,PrevIter)   ).
```

Combining Tabling with $\mathcal{ALC}$ via MKNF

How the DL prover calls the rules. `hasAttr = hasAttr_ext + hasAttr_int`

```
hasAttr_int(oid(Obj1,NS),rid(Role,NS1),oid(Obj2,NS2)):-
    ground(Obj1), ground(Obj2), ground(Role),!,
    /* Call = Role(Obj1,Obj2) */
    Call =.. [Role,Obj1,Obj2],
    /* Check last iteration, if any, of rules */
    last_known(Call).

/* Find all possible rules for Obj if called with role argument uninstantiated */
hasAttr_int(oid(Obj1,NS),rid(Role,NS1),oid(Obj2,NS2)):-
    ground(Obj1), ground(Obj2), var(Role),
    definedRole(Call,Role,Obj1,Obj2),
    last_known(Call).
```

Combining Tabling with $\mathcal{ALC}$ via MKNF

NegativeLiterals $= N_0$

$$D_0 = Tableau(\mathcal{O}) \qquad R_0 = WFS(\mathcal{P}/N_0)$$
$$\ldots \qquad \ldots$$
$$D_n = Tableau(\mathcal{O} \cup R_{n-1}) \qquad R_n = WFS(\mathcal{P}/N_0 \cup D_{n-1})$$

NegativeLiterals $= N_1$

$$D_0 = Tableau(\mathcal{O}) \qquad R_0 = WFS(\mathcal{P}/N_1)$$
$$\ldots \qquad \ldots$$
$$D_n = Tableau(\mathcal{O} \cup R_{n-1}) \qquad R_n = WFS(\mathcal{P}/N_1 \cup D_{n-1})$$

:

NegativeLiterals $= N_n$

$$D_0 = Tableau(\mathcal{O}) \qquad R_0 = WFS(\mathcal{P}/N_n)$$
$$\ldots \qquad \ldots$$
$$D_n = Tableau(\mathcal{O} \cup R_{n-1}) \qquad R_n = WFS(\mathcal{P}/N_n \cup D_{n-1})$$

Combining Tabling with $\mathcal{ALC}$ via MKNF

For programs with negation

- Rules treat DL-negation as explicit negation, and ensure that explicit negation implies default negation via coherency.
- Now we must have an iterated fixed point
  - Inner fixed point determines true and explicitly false literals as with definite programs, but default negation uses the previous outer iteration, if any
  - At each outer iteration, we change the set of atoms that default negation considers "proven"
- An alternating fixed point is used as in [van89]. However because of the goal-orientation, this can be manageable
- Of course, (non shared) program atoms can be computed as usual via SLG.

Combining Tabling with $\mathcal{ALC}$ via MKNF

```
dlnot(A):-
        computeFixedPoint(A),
        get_object_iter(A,OutIter),
        dlnot(A,OutIter).

/* In 1st iteration, ensure that TU = everything */
dlnot(_A,0):- !.
/* In subsequent iterations,
   check previous outer iteration */
dlnot(A,OutIter):-
          PrevIter is OutIter - 1,
          get_final_inner_iter(A,PrevIter,Inner),
          tnot(known(A,PrevIter,Inner)).
```

Combining Tabling with $\mathcal{ALC}$ via MKNF

To Summarize

- Rules may call the DL-prover; the DL-prover may call rules
- Goal-directedness is provided by WFS, the Tableau Prover, and by trying to minimize the number of individuals for which statements must be proven.
- WFS is provided by an alternating fixed point. This means that DL-derivations are "non-monotonic" in the sense that they may change from iteration to iteration.
- XSB's tabling strategy, SLG, uses delay and simplification to compute WFS, rather than the alternating fixed point. For technical reasons, the use of SLG for MKNF can lead to unsupported conclusions. These conclusions can in principle be detected, but such detection can be complicated.

Combining Tabling with $\mathcal{EL}^+$ via MKNF [4]

- XSB does not in general use an alternating – or iterated – fixed point to compute WFS. Rather it uses SLG resolution: tabling with delay and simplification.
  - Essentially, literals whose top-down evaluation may have a loop through negation are delayed, and may later be simplified away as a proof or witness of failure becomes available.
- **SLG($\mathcal{O}$)** extends SLG to handle calls to external deduction over a theory $\mathcal{O}$ treating $\mathcal{O}$ as an oracle.
- Greatly simplifying things, **SLG($\mathcal{O}$)** adds an ORACLE RESOLUTION operation to SLG. Given a goal G to the oracle, the oracle responds with a rule G :- $L_1, \ldots, L_n$ where $\mathcal{O} \cup \{L_1, \ldots, L_n\} \models G$.
- Given a suitable ORACLE RESOLUTION operator for a theory, **SLG($\mathcal{O}$)** is shown to be sound and complete for MKNF, and to have various nice properties.

[4] Joint work with M. Knorr and J. Alferes [AKS09].

Combining Tabling with $\mathcal{EL}^+$ via MKNF

- **SLG($\mathcal{O}$)** needs neither alternation nor an explicit iterated fixed point – its just tabling with a special operation or two. Its thus more "closely" integrated with XSB and Prolog.
- On the other hand, **SLG($\mathcal{O}$)** puts greater demand on the prover's callback function than our previous method.
- As a first step, we extend WFS with $\mathcal{EL}^+$ through **SLG($\mathcal{O}$)** [KA10].
- A similar result was very recently obtained by M. Knorr and J. Alferes to use **SLG($\mathcal{O}$)** to combine WFS with DL-lite$_R$.

Description Logics: $\mathcal{EL}^+$ [BBL05]

The syntax of $\mathcal{EL}^+$ and its semantics. C, D are general concepts, $r(_i)$ are primitive roles.

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\exists$-restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
GC Inheritance	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
R Inheritance	$r_1 \circ \cdots \circ r_k \sqsubseteq r$	$r_1^{\mathcal{I}} \circ \cdots \circ r_k^{\mathcal{I}} \sqsubseteq r^{\mathcal{I}}$
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$r(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}})$

Combining Tabling with $\mathcal{EL}^+$ via MKNF

- Rules are defined for the various sequents used to prove subsumption in DL.
- A query to $\mathcal{EL}^+$ of the form $C(a)$ or $R(a, b)$ causes the $\mathcal{EL}^+$-prover to try to construct a proof. If shared atoms are encountered, the prover returns a rule that needs to be proved.
- For instance, suppose the prover is trying to prove $D(a)$ and there is an axiom

$$\exists R.C \sqsubseteq D$$

such that R or C were shared. Then if it has no other proof of $D(a)$ the $\mathcal{EL}^+$ prover may return the rule:

$$D(a) \leftarrow R(a, Y), C(a)$$

for further evaluation by the rule system.

- Basically, you can think of the Oracle as "magically" adding program rules on demand.

Adding Evidential Probability to MKNF [5]

- Evidential Probability (EP) [KT01] is a method that allows reasoning about statistical statements that have been added to a theory. Statements have the form:

$$\%\vec{x}(\tau(\vec{x}), \rho(\vec{x}), [l, u]).$$

  This can be read that $\rho(\vec{x})$ implies $\tau(\vec{x})$ with a probability between l and u.
- Given a DL, such statments can be rewritten for classes as:

$$\%(\tau, \rho, [l, u]).$$

  indicating that each ρ is a τ with probability between l and u. Roles can be treated similarly, but we focus on classes here.
- Statistical statements may be scattered among classes, and may represent partial or inconsistent information.

[5] Joint Work with Gregory Wheeler [SW11]

Adding Evidential Probability to MKNF

- Its important to note that EP is a low-complexity method for meta-reasoning about statistical statments that have been added to a logic. It is *not* part of the logic itself.
- For a DL, EP uses the principles of Richness, Specificity, Precision and Strength to reason about the subjective probability that a given individual belongs to a class (or has a given role).
- Two intervals *conflict* if one is not a subinterval of another. EP provides a method to reason about inconsistent sets of intervals, even if the set is non-convex.
- In epidemiology or actuarial studies, full distributions may not be available, e.g. we don't have studies for the incidence of a disease for all genomes and environments. As a result, Bayesian reasoning can be difficult or impossible to apply in these cases, but the weaker method of EP is available.

Computing the EP for a DL Theory

Let's say that we have a red, imported racing bicycle b and want to determine for insurance purposes whether b might be stolen in the future.

1. Determine the set $\mathcal{S}$ of *potential probability statements* that apply to b, i.e. statements of the $\%(\tau, \rho, [l, u])$ such that $b \in \rho$ and $stolen \sqsubseteq \tau$.
2. Apply the *Principle of Richness* to remove from $\mathcal{S}$ each statistical statement σ that conflicts with some other statement $\sigma' \in \mathcal{S}$ that is more reliable than σ – and that does not conflict with some even more reliable statement. These unreliable statements may represent marginal rather than full distributions or less stringent studies. In this way we produce $\mathcal{S}^R$.
3. Apply the *Principle of Specificity*. Remove from $\mathcal{S}$ each $\%(\tau, \rho, [l, u])$ σ that conflicts with some other statement $\%(\tau', \rho', [l', u'])$ such that $\rho' \sqsubseteq \rho$ and $\%(\tau', \rho', [l', u'])$ is not similarly conflicted by some statement in $\mathcal{S}^R$ about a more specific ρ'' than ρ'. Call this new set $\mathcal{S}^{RS}$.
4. Apply the *Principle of Precision*. Remove from $\mathcal{S}^{RS}$ any statement $\%(\tau, \rho, [l, u])$ such that there is some other statement $\%(\tau', \rho', [l', u'])$ in $\mathcal{S}'$ such that $[l', u']$ is a subinterval of $[l, u]$. Call this set $\mathcal{S}^{RSP}$.
5. Apply the *Principle of Strength* to determine an EP-covering interval for $\mathcal{S}^{RSP}$. The actual algorithm used tries to make this interval as precise as possible and can be found in [KT01, SW11].

Adding Evidential Probability to MKNF

As a simple example: recall that b is a red, imported racing bicycle and suppose we have the following knowledge base. The Specificity relation is implied by the names, and the Richness relation is empty.

%(stolen,redMountainDomestic,0.027,0.971)
%(stolen,redTouringDomestic,0,0.062)
%(stolen,redRacingDomestic,0,0.055)
%(stolen,redTouring,0,0.045)
%(stolen,redMountain,0.027,0.097)
%(stolen,redDomestic,0.012,0.057)
%(stolen,touringImported,0,0.055)
%(stolen,racingDomestic,0,0.057)
%(stolen,mountainDomestic,0,034,0.786)
%(stolen,racing,0,0.046)
%(stolen,mountain,0.035,0.077)
%(stolen,domestic,0.021,0.050)
%(stolen,redMountainImported,0,1)
%(stolen,redTouringImported,0,0.056)
%(stolen,redRacingImported,0,0.063)
%(stolen,redRacing,0,0.045)
%(stolen,redImported,0,0.046)
%(stolen,racingImported,0,0.058)
%(stolen,mountainImported,0,0.121)
%(stolen,touringDomestic,0,0.045)
%(stolen,red,0.008,0.047)
%(stolen,touring,0.005,0.0407)
%(stolen,imported,0.004,0.049)
%(stolen,top,0.021,0.045)

Adding Evidential Probability to MKNF

First, find the set of potential probability statements – those that apply to a class to which b belongs:

%(stolen,redRacingImported,0,0.063)
%(stolen,redImported,0,0.046)
%(stolen,red,0.0084,0.047)
%(stolen,imported,0.004,0.049)
%(stolen,redRacing,0,0.045)
%(stolen,racingImported,0,0.058)
%(stolen,racing,0,0046)
%(stolen,top,0.021,0.045)

Next, applying the *Principle of Specificity* obtains

%(stolen,redRacingImported,0,0.063)
%(stolen,redImported,0,0.046)
%(stolen,racing,0,0046)
%(stolen,redRacing,0,0.045)
%(stolen,racingImported,0,0.058)

Finally, applying the Principle of Precision obtains the interval $[0, 0.0454]$. The Principle of Strength is not needed here.

Adding Evidential Probability to MKNF: Summary‘

- MKNF provides a formalism to add EP to a DL KB.
  - DL+Rule reasoning is used to obtain the initial set S of potential probability statements for a given individual, and to determine the specificity relationship.
  - DL+Rule reasoning also can be used to represent the binary Richness relation over reified the statistical statments.
  - A more procedural rule-based algorithm may be used to obtain the final interval.
  - This is implemented on top of CDF-Rules as a proof of concept.
- The entire process of EP reasoning is $\mathcal{O}(N_P^2)$ where N_P is the number potential probability statements ($\mathcal{S}$).
- In terms of complexity, EP is a reasonable extension not only to $\mathcal{EL}^+$, but also to rule systems with inheritance such as CDF, Flora-2, or Silk.
- [SW11] provides a much more complex example involving pig breeding.

Summary

- CDF Type(0) Ontologies: $C \sqsubseteq D$; $R_1 \sqsubseteq R_2$; $\sqcap$; $\exists R.C$; $\forall R.D$.
  - Relatively simple XSB library (˜6000 lines). **Suitable for research and commercial use.**
- Flora-2: $C \sqsubseteq D$; $\sqcap$; $\neg$; *not*; equality reasoning, HiLog
  - Highly sophisticated XSB library (˜50000 lines). **Suitable for research and commercial use.**
- $\mathcal{EL}^+$: $C \sqsubseteq D$; $R_1 \circ \ldots \circ R_n \sqsubseteq R$; $\sqcap$; $\exists R.C$; $\bot$.
  - Concrete oracle instantiation to combine '$\mathcal{EL}^+$ with XSB through **SLG($\mathcal{O}$)**, but not yet implemented.
  - Concrete oracle instantiation to combine DL-lite$_R$ with XSB through **SLG($\mathcal{O}$)**, but not yet implemented.
- CDF Type(1) Ontologies: $\mathcal{ALCQ}$
  - Combined with XSB through MKNF package as a research prototype.
- Evidential Probability
  - Implemented within MKNF formalism as a research prototype; suitable for any of the above packages.

(Hopefully) Provocative Questions

- There are various levels of scalability: XSB (+CDF), $MKNF_{WFS} + \mathcal{ALC}/\mathcal{EL}^+$; ASP... What level of scalability will be needed in 2 years for semantic web applications?
- Many of OWL KBs (UMLS,SNOMED, ICD) consist of inheritance + binary relations. Not much, if any, reasoning is performed beyond monotonic inheritance.
- When do we need DLs? Can we get by with an OO-logic like Flora-2, or even a Prolog-extension like CDF?
- What is it about FOL that we really need? Open-world negation – as opposed to explicit negation? ???
- Can KEs specify a KB in a rule system? They can't in Prolog, CLP, TLP, or Flora. They might in Silk. Can optimizing compilers help?
- Deep reasoning can be done by rule-based KBs – e.g. Cyc, but extending this knowledge requires specialized training. Can rule based KBs ever perform reasoning in a portable manner like OWL does?

Current and Future Work for LP-based KBs

- Handling updates [SLS11] through incremental tabling and other techniques.
- Explicit management of tables – when can/should they be abolished?
- How do you decide what to table? How should these things be tabled (call-variance, call-subsumption, incremental)
- How/when can literal reordering optimizations be done? What sort of termination analysis is best? How/when can forms of recursion be rearranged (e.g. double- to single- recursion).
  - Sophisticated manual optimizations of recursive forms was needed when translating Cyc into Silk (and then into XSB).
- How can explosive behavior analyzed or profiled, and how can this behavior be presented to a KE?
- Based on these techniques how can an IDE be designed so that KBs can create and maintain sophisticated, reusable KBs?

References I

[AKS09] J.J. Alferes, M. Knorr, and T. Swift.
Query-driven procedures for hybrid MKNF knowledge bases.
In *International Conference on the Semantic Web*, 2009.
Full version available at `http://www.cs.sunysb.edu:~tswift`.

[BBL05] F. Baader, S. Brandt, and C. Lutz.
Pushing the $\mathcal{EL}$ envelope.
In *IJCAI'05: 19th Int. Joint Conf. on Artificial Intelligence*, pages 364–369. Morgan Kaufmann, 2005.

[BCM+03] F. Baader, D. Calvenese, D. McGuiness, D. Nardi, and P. Patel-Schneider.
The Description Logic Handbook: Theory, Implementation, and Applications.
Cambridge University Press, 2003.

References II

[GAS11] S. Gomes, J.J. Alferes, and T. Swift.
Implementing query answering for hybrid MKNF knowledge bases.
In *Theory and Practice of Logic Prog.*, 2011.
To appear.

[KA10] M. Knorr and J.J. Alferes.
Querying in el+ with nonmonotonic rules.
In *European Conference on Artificial Intelligence*, pages 1079–1080, 2010.

[KAH08] M. Knorr, J.J. Alferes, and P. Hitzler.
A coherent well-founded model for hybred mknf knowledge bases.
In *European Conference on AI*, 2008.

[KT01] H. Kyburg and C. Teng.
Uncertain Inference.
Cambridge University Press, 2001.

References III

[MR07] B. Motik and R. Rosati.
A faithful integration of description logics with logic programming.
In *20th Int. Joint Conf on Artificial Intelligence (IJCAI)*, pages 477–482, Hyderabad, India, January 6–12 2007. AAAI Press.

[SLS11] M. Slota, J. Leite, and T. Swift.
Splitting and updating hybrid knowledge bases.
Theory and Practice of Logic Prog., 11(4-5):800–819, 2011.

[SW11] T. Swift and G. Wheeler.
Extending description logics to support statistical information.
Available at `http://www.cs.sunysb.edu/~tswift`, 2011.

[Swi04] T. Swift.
Deduction in ontologies via answer set programming.
In *Intl. Conf. on Logic Prog. and Non-Monotonic Reasoning*, pages 275–289, 2004.

References IV

[van89] A. van Gelder.
The alternating fixpoint of logic programs with negation.
In *Proc. of 8th PODS*, pages 1–10. ACM, 1989.

[WGK+09] H. Wan, B. Grosof, M. Kifer, P. Fodor, and S. Liang.
Logic programming with defaults and argumentation theories.
In *Intl. Conf. on Logic Prog.*, pages 432–448, 2009.

[YKWZ11] G. Yang, M. Kifer, H. Wan, and C. Zhao.
FLORA-2: User's Manual Version 0.97, 2011.