OWL + ASP = ColdDeadFish

(Work in Progress)

Terrance Swift

SUNY Stony Brook

Overview

- Context of work
  - Overview of Ontologies;
  - Why Ontologies are important for LP;
  - ColdDeadFish (Swift, Warren);
- An ASP-based theorem-prover for ColdDeadFish.

"If Logic Programmers invented sushi, they'd market it as 'cold dead fish'."
– Rupert Hopkins, President, XSB Inc.

Ontologies

- Model *taxonomies* of classes (e.g. a set of classes ordered by the subset relation)

```
subclass('Medical Equipment',top).
  subclass'Wound Care Products','Medical Equipment').
    subclass('Sutures and Related Products',
                                 'Wound Care Products').
      subclass('Sutures','Sutures and Related Products').
```

- Model objects that are members of a given class.

```
  memberof('Suture 547466',Absorbable Sutures')
  memberof('Suture U245H','Sutures and Related Products')
```

- Such taxonomies underly proposals for web-based commerce:
  - UNSPSC
  - NAICS
  - ISO 3166
  - HSUSA
  - Reed
  - SnoMed

Ontologies

- Model relations between classes (and objects)
  - Inheritable or not – different sementics
    - Relations may be inherited by subclasses or by objects as attributes;
    - Not all relations are inheritable, e.g. a match-score.
  - Existential or Universal
    - For each person, p, there exists a *father-of(p)* who is male.
    - For each person, p, any *brother-of(p)* must be male.
- Example: RosettaNet

```
existsrel(Sutures,endType,endTypeDomain).
allrel(Sutures,endType,endTypeDomain).

subclass(endTypeDomain,top).
  subclass(singleArmed,endTypeDomain).
  subclass(doubleArmed,endTypeDomain).
  subclass(unarmed,endTypeDomain).

attribute('Suture U245H',endTypeDomain,doubleArmed).
```

Ontologies

Often add a logic-based Class Definition Language (CDL)

- *Extended* classes can be defined in terms of (above) *atomic* classes
  - *A* singleArmedSuture *is a* suture *whose* endType *has value* singleArmed.
- Necessary or sufficient conditions for atomic classes may be defined.
  - *No* suture *can be both* singleArmed *and* doubleArmed.
- If a CDL is decidable, a theorem prover can
  - check satisfiability of extended classes;
  - check subsumption of one class by another.
- Examples: Federal Logistics Information System, Galen

Ontologies

Ontologies have been proposed as a major component of the Semantic Web.

- DAML+OIL is a DARPA-sponsored effort to define ways in which DoD agents communicate knowledge.
- OWL is a W3C-sponsored effort to define a language for representing knowledge by means of ontologies.
- Both have syntaxes that are specializations of RDF.
- Neither is a “standard” yet, but represents the current thinking of much of the ontology community.

Why LP-Ontologies?

- Tight coupling to XSB
  - Representation of ground Prolog terms
  - Don't want to go through a CORBA or HTML interface
- Integration with XSB tools and interfaces
  - DB-interfaces
    * Ontologies with $10^7 - 10^8$ elements
  - Visualization/Editing Tools
    * Protege insufficient for large databases,
  - Documentation Generators (xsbdoc)
- Create a platform to explore integration of different KR techniques.

ColdDeadFish

Two Levels

1. Manipulation of classes with restricted CDLs: consistency in P (Swift, Warren)
   - Used extensively in XSB, Inc (15 employees) products.
     - Extraction of structured data from unstructured textual or OCR input;
       - Generate parsers that use Preference Logic Grammars
     - Classification of parts by their attributes;
     - Agent communication.
       - E.g. Web-services Broker for DARPA Agents in Cougaar Architecure
       - Agents communicate via JAXR, JNTI and other acronyms I dont understand.
2. Manipulation of classes with expressive CDLs (Swift):
   - Theorem prover determines
     - satisfiability of class definitions
     - subsumption of class definition
   - Integration of theorem prover with rest of system is underway.
     - acyclic class definitions (Version 1).

ColdDeadFish

- Both levels will be released as Open-Source Software around Jan 1.
- About 10,000 lines of XSB code, 60-70 page manual.

Syntax of one Class Definition Language

For

- A, A': atomic class
- P, P': atomic relation; Tr, Tr': atomic transitive relation
- n: positive integer
- C, C_1, C_2 arbitrary classes
- R: arbitrary relation

$$C \leftarrow A|\neg C_1|C_1 \sqcap C_2|C_1 \sqcup C_2|all(R, C_1)|exists(R, C_1)$$
$$|atLeast(n, R, C_1)|atMost(n, R, C_1)$$

$$R \leftarrow P|Tr|R^-$$

Subclassing constraints:

$$subclass(A, A')$$
$$subclass(P, P')$$
$$subclass(Tr, Tr')$$

Class Definitions

- Definition covers $\mathcal{ALCQI}$ plus transitive and hierarchical relations, and rudimentary "rules" (e.g. *subclass(A,A')*);
- Deciding whether a formula plus constraints is inconsistent (under semantics below) is
  - Decidable ;
  - Various subsets, e.g. $\mathcal{ALC}$, are PSPACE-complete [Smolka-Schmidt-Strauss, 91]; $\mathcal{ALCQI}$ is PSPACE-hard (for binary encoding of integers).
- Covers most features needed for consistency or subsumption checking of OWL class definitions.

Semantics of Class Definition Language

For

- a set $\mathcal{A}$ of atomic classes;
- a set $\mathcal{P}$ of atomic relations;
- a set $\mathcal{T}$ of atomic transitive relations

an interpretation function $< \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} >$ is a function from (non-atomic) classes and relations to a set $\Delta^{\mathcal{I}}$ such that

(1)

- $A \in \mathcal{A}, A^{\mathcal{I}}$ is subset of $\Delta^{\mathcal{I}}$
- $P \in \mathcal{P}, P^{\mathcal{I}}$ is a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
- $Tr \in \mathcal{T}, Tr^{\mathcal{I}}$ is a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ that constitutes a transitive relation.
- if $subclass(A, A')$ then $A^{\mathcal{I}} \subseteq A'^{\mathcal{I}}$
- if $subclass(P, P')$ then $P^{\mathcal{I}} \subseteq P'^{\mathcal{I}}$
- if $subclass(Tr, Tr')$ then $Tr^{\mathcal{I}} \subseteq Tr'^{\mathcal{I}}$

Semantics

(2)

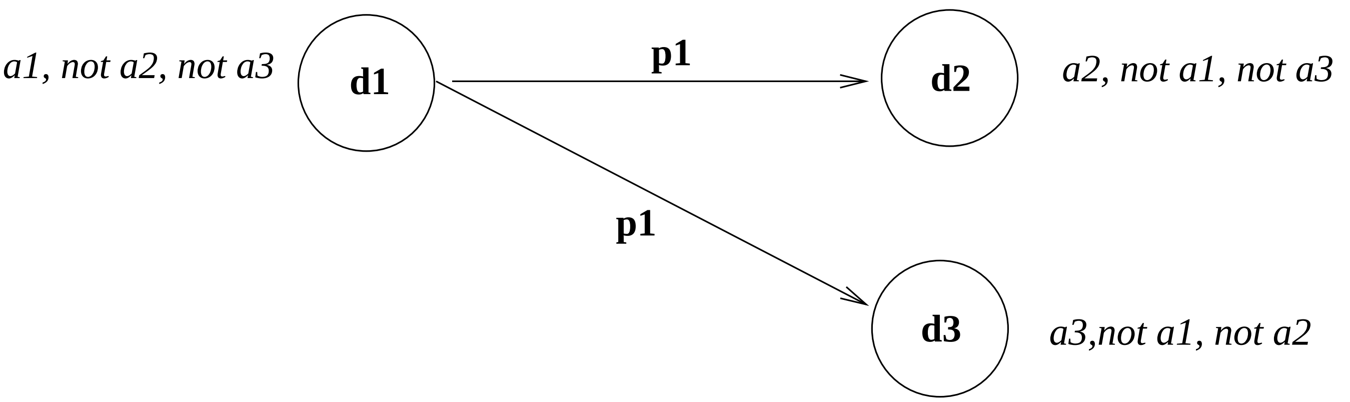
$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} - C^{\mathcal{I}} \\
(exists(R, C))^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\
(all(R, C))^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \forall y. \langle x, y \rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\} \\
(atLeast(n, R, C))^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \sharp\{y | \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\} \\
(atMost(n, R, C))^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} | \sharp\{y | \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\} \\
(R^{-})^{\mathcal{I}} &= \{\langle o', o \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} | \langle o, o' \rangle \in R^{\mathcal{I}}\}
\end{aligned}
$$

- $\sharp$ is used to denote the cardinality of sets.
- $\top =_{def} (C \sqcup \neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}}$
- $\bot =_{def} (C \sqcap \neg C)^{\mathcal{I}} = \emptyset$
- A class expression C is *satisfiable* if there exists an interpretation $\mathcal{I}$ such $C^{\mathcal{I}} \neq \emptyset$.
- A class expression C_1 *subsumes* a class C_2 if in every interpretation $\mathcal{I}$, $C_2^{\mathcal{I}} \subseteq C_1^{\mathcal{I}}$.

Class Definitions

(a1,exists(p1,a2),exists(p1,a3))

$$\Delta^{\mathcal{I}} = \{d1, d2, d3\}$$
$$a1^{\mathcal{I}} = \{d1\}$$
$$a2^{\mathcal{I}} = \{d2\}$$
$$a3^{\mathcal{I}} = \{d3\}$$
$$p1^{\mathcal{I}} = \{\langle d1, d2\rangle, \langle d1, d3\rangle\}$$



$$exists(p1, a2)^{\mathcal{I}} = \{d1\}$$
$$exists(p1, a3)^{\mathcal{I}} = \{d1\}$$
$$(a1, exists(p1, a2), exists(p1, a3))^{\mathcal{I}} = \{d1\}$$

Class Definitions

- Theorem proving method stems from Hollunder, Nutt, Schmidt-Strauss [90]
  - extended to qualified cardinality expressions, transitive and hierarchical relations, etc.
  - implemented in ASP
- Main idea of search is to traverse through Class Definition and
  - When encountering *exists(R,C)* and *at_least(n,R,C)*
    - Add new domain elements (non-deterministically);
    - Adjust interpretation of primitive relations (non-deterministically);
  - When encountering primitive classes, adjust interpretations (non-deterministically);
  - When encountering *not, ',', ';'*, manipulate the interpretations;
  - Check that interpretations constructed satisfy *all(R,C), at_most(n,R,C)*.

Class Definitions

1. Transform the Class Definition to a logic program that "abduces" an interpretation under a set of constraints
   - *memberof(D,A)* states that $D \in A^{\mathcal{I}}$, for domain element D and primitive class A.
   - *rel*(D_1, P, D_2) states that $< D_1, D_2 > \in P^{\mathcal{I}}$, for domain element D_1 and D_2; and primitive relation P.
2. Abduce an interpretation so that the constraints are satisfied via a 2-valued model.

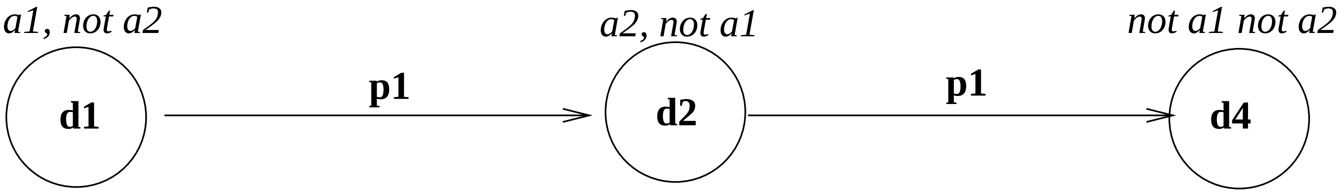
(a1,exists(p1,a2),exists(p1,a3))

```
satisfiable:-
  memberof*(d1,a1),
  rel*(d1,p1,d2), memberof*(d2,a2),
  rel*(d1,p1,d3), memberof*(d3,a3).
```

Class Definitions: Cardinality Constraints

(a1 ; not(a1)),exists(p1,a2),exists(p1,exists(p1,not(a1))),
atMost(1,p1,top)

$$\Delta^{\mathcal{I}} = \{d1, d2, d3, d4\}$$
$$a1^{\mathcal{I}} = \{d1\}$$
$$a2^{\mathcal{I}} = \{d2\}$$
$$r^{\mathcal{I}} = \{\langle d1, d2\rangle, \langle d2, d4\rangle$$



$$(a1; not(a1))^{\mathcal{I}} = \{d1, d2, d3, d4\}$$
$$exists(p1, a2)^{\mathcal{I}} = \{d1\}$$
$$exists(p1, not(a1))^{\mathcal{I}} = \{d1, d2\}$$
$$exists(p1, exists(p1, not(a1)))^{\mathcal{I}} = \{d1\}$$
$$atMost(1, p1, top)^{\mathcal{I}} = \{d1, d2, d3, d4\}$$

Conceptual Compilation

(a1 ; not(a1)),exists(p1,a2),exists(p1,exists(p1,not(a1))),
atMost(1,p1,top)

```
satisfiable:- sat1,sat2,sat3,sat4.

sat1:-
  guard1*,
  not guard1a*,
  memberof*(d1,a1)).

sat1:-
   guard1a*,
   not guard1*,
  -memberof*(d1,a1).

sat2:-
  rel*(d1,p1,d2),
  memberof*(d2,a2),

sat3:-
  guard3*,
  not guard3a*,
  rel*(d1,p1,d2),
  rel*(d2,p1,d4),
  -memberof*(d4,a1).

sat3:-
  guard3a*,
  not guard3*,
   rel*(d1,p1,d3),
   rel*(d3,p1,d4),
  -memberof*(d4,a1).

sat4:- {rel(d1,p1,d3),rel(d1,p1,d4)} 1.
```

Conceptual Representation

"Guards" mimic a top-down evaluation by

- Enforcing mutual exclusion of disjuncts
- Ensuring that certain literals are abduced together: `rel(d1,p1,d2), rel(d2,p1,d4)` are both abduced based on `guard1` being true;
- Allow abduction only if relevant to a derivation.
  - E.g. `{ guard_n } 1 :- sat_m.`
  - E.g. `guard_n v not_guard_n :- sat_m.`

Stable Model Generation

Compute `satisfiable`

```
satisfiable:-
  sat1,                                %(a1 ; not(a1))
  sat3.                                %exists(p1,exists(p1,not(a1))

rel(d1,p1,d2).                         %exists(p1,a2)
memberof(d2,a2).

{rel(d1,p1,d2),rel(d1,p1,d3)} 1.       %at_most(1,p1,top)

sat1:-                            sat1:-
  guard1*,not guard1a*.             guard1a*,not guard1*.

memberof(d1,a1):- guard1*,not(guard1a).
-memberof(d1,a1):- guard1a*,not(guard1).

sat3:-                            sat3:-
  guard3*,not guard3a*.             guard3a*,not guard3*.

rel(d1,p1,d3):- guard3a*,not(guard3).
rel(d3,p1,d4):- guard3a*,not(guard3).
-memberof(d4,a1):- guard3*,not(guard3a).

rel(d2,p1,d4):- guard3*,not(guard3a).
-memberof(d4,a1):- guard3*,not(guard3a).
```

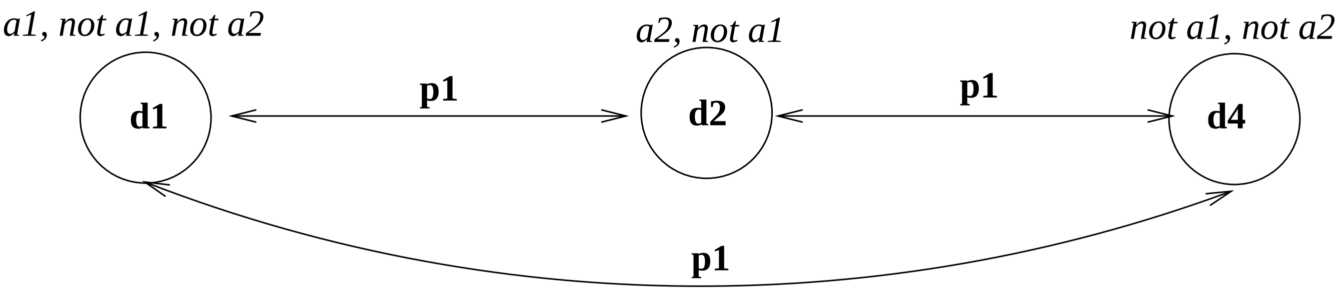
Stable Model Generation

- XSB
  - Constructs Class Definitions from an ontology (under development);
  - Compiles Class Definitions to stable model;
  - Grounds the stable model (using tabling)
    * Current targets: SModels or DLV.
- ASP used to
  - abduce;
  - check consistency.
- Benchmark programs determine consistency of some class expressions of size 10^4 in a couple of seconds
  - Many optimizations but core TP is < 1000 lines of code
- Appears suitable for large-scale ontologies;
- Comparisons with tableaux-based theorem provers, Fact, DLP not yet available.

Class Definitions: Relational Properties

What happens if r is transitive (and symmetric)?

(a1 ; not(a1)),exists(p1,a2),exists(p1,exists(p1,not(a1)),
atMost(1,p1,top))



- A consistent interpretation cannot be constructed: $d1$ must belong to $a1 \cap not(a1)$, i.e. to $\emptyset$.

Class Definitions: Relational Properties

- Relational properties cause additional rule generation
  - E.g. transitive

    ```
    rel(d1,p1,d3):-
      rel(d1,p1,d2),
      rel(d2,p1,d3).
    ```

    Ensures that when `rel(d1,p1,d2)` and `rel(d2,p1,d3)` are abduced then `rel(d1,p1,d3)` is also derived.
  - Same approach for symmetric relations, hierarchical relations, subclass rules, etc.
- Relational properties must be taken into account when generating ASP program.

Discussion

- Search and consistency checks of ASP systems are efficient; sufficient control to emulate the top-down features of a tableaux-style theorem prover.
- Advantage of ASP systems in propagating inferences: e.g. maintaining transitive, inverse, and hierarchical relations as new relational elements are added.
- XSB has advantages as a grounder (at least for the author :-)
- Can ASP add to the semantics of ontologies?
  - Query relaxation via annotation logic
  - Is it useful to extend e.g. at_least(n,R,C) with weight constraints?
  - Under what conditions can necessity and sufficiency axioms (perhaps more general than shown here) in a CDL be seen as ASP rules?
  - Is there a more direct encoding into LP/ASP?