Combining Logic Programs with Object-Oriented Representation
Outline

• Limitations of frames & description logics
• Early hybrid KR&R systems
  – Krypton
• Recent trends in hybrid KR&R systems
  – Description graphs
  – F-Logic
Limitation of Frames & Description Logics

- Polyadic predicates
  - Relations with arity greater than two
    - A is between B and C
- Limited relational expressivity
  - DLs can model only domains where objects are connected in a tree like manner
    - "an uncle of a person is a brother of that person’s father"
- Disjunction
  - Many concepts have disjunctively defined definitions
    - A person is legally employable if they are of employable age, and they are either a legal immigrant or US citizen
- Negation and relative complements
  - Many facts are inherently negative
    - Under no circumstances is it illegal to drive on the shoulder of a road
- Concepts defined using conditionals
  - An animal is dangerous to humans if it attacks them when near by
- Equivalence to two other concepts
  - A dinner wine is proper if it is red whenever the main dish is red meat or is tomato based or else if it is white wine

A more complete list is available in "Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services" by Doyle and Patil, AIJ, 1991
Krypton

- Knowledge representation has two components
  - Terminological component (Tbox)
  - Assertional component (ABox)
- Functional specification of a Knowledge Base
  - Instead of focusing on ``What structures should the system maintain for a user?'' shift the focus to ``What exactly should the system do for a user?''
- Hybrid reasoning using theory resolution
  - An extension to resolution to combine theorem proving with terminological reasoning
Knowledge Representation in Krypton

Doctor ≡ [And Professional
[ Exists 1 MedicalDegree]
....]

Person ≡ Human

Exists x, y Person (x) \land child (y, x) \land Doctor (y)
Functional Specification of a Knowledge Base

- Two classes of operations:
  - Tell and Ask
- Tell Operation
  - Tell for a Tbox takes a symbol and associates it with a Tbox term
  - Tell for an ABox takes a sentence and asserts that to be true
- Ask Operation
  - Ask for a Tbox can ask whether one term subsumes the other or whether one is disjoint with another
  - Ask for an ABox takes a sentence and asks if it is true

Modern APIs to knowledge bases can have over 100 operations
Hybrid Reasoning

- Theory resolution is a set of complete procedures for incorporating theories into resolution theorem proving
  - Reduce lengths of proof
  - Reduce size of the search space
  - Beneficial division of labor
Hybrid Reasoning

• Example to show theory resolution
  – Prove that if Chris has no sons and daughters, then Chris has no children
  – Knowledge stated using structured descriptions
    • Boys are persons whose sex is male
    • Girls are persons whose sex is female
    • No sons are persons all of whose children are girls
    • No daughters are persons all of whose sons are boys
    • Every person has a sex
    • Male and female are disjoint
Example of Theory Resolution

1. \( Boy(x) \cup Person(x) \)
2. \([Boy(x) \land Sex(x, y)] \cup Male(y) \)
3. \( Girl(x) \cup Person(x) \)
4. \([Girl(x) \land Sex(x, y)] \cup Female(y) \)
5. \([NoSon(x) \land Child(x, y)] \cup Girl(y) \)
6. \([NoDaughter(x) \land Child(x, y)] \cup Boy(y) \)
7. \( Person(x) \cup Sex(x, sk1(x)) \)
8. \( Male(x) \equiv \neg Female(x) \)

hypothesis

9. \( NoSon(Chris) \)

hypothesis

10. \( NoDaughter(Chris) \)

negated conclusion

11. \( Child(Chris, sk2) \)

resolve 11 and 5, simplify by 9

12. \( Girl(sk2) \)

resolve 12 and 3

13. \( Person(sk2) \)

resolve 7 and 13

14. \( Sex(sk2, sk1(sk2)) \)

resolve 11 and 6, simplify by 10

15. \( Boy(sk2) \)

resolve 15 and 2

17. \( Sex(sk2, x) \cup Male(x) \)

hypothesis

9. \( NoSon(Chris) \)

hypothesis

10. \( NoDaughter(Chris) \)

negated conclusion

11. \( Child(Chris, sk2) \)

resolve 11 and 9

12. \( Girl(sk2) \)

resolve 11 and 10, simplify by 12

13. \( \Box \)

resolve 14 and 20, simplify by 19

21. \( \Box \)

Figure 1: Resolution Proof for Childless Problem

Figure 2: KRYPTON Proof for Childless Problem
Recent Trends in combining formalisms

- Start from a description logic system and add features that address the known limitations and yet retain soundness, completeness, and efficiency
  - Description graphs [Motik et. al., AIJ 2009]
    - Extend a DL system to include graphs that describes parts connected in arbitrary ways
    - Represent conditional knowledge by adding rules
- Start from a logic programming system and add features of description logics and yet do not sacrifice the properties of the base formalism
  - F-Logic / SILK
    - We will consider this in more detail
F-Logic / SILK

- F-logic was originally developed as a database logic to formalize object-oriented systems including methods and inheritance
  - [Kifer, Lausen, Wu, JACM 2005]

- SILK is a recent knowledge representation language based on F-Logic that supports F-logic and several new features such as Lloyd Topor transformation, named skolem functions, equality, and prioritized defaults
  - See http://silk.semwebcentral.org/
A Simple Example of F-logic

Object description:

John[\textit{name} -> 'John Doe', \textit{phones} -> \{6313214567, 6313214566\},
\textit{children} -> \{Bob, Mary\}]

Mary[\textit{name} -> 'Mary Doe', \textit{phones} -> \{2121234567, 2121237645\},
\textit{children} -> \{Anne, Alice\}]

Structure can be nested:

Sally[\textit{spouse} -> John[\textit{address} -> '123 Main St.'] ]

Adapted from a tutorial on Programming with Logic and Frames by M. Kifer
Defining Classes and Individuals

ISA hierarchy:

John : Person - *class membership*
Mary : Person
alice : Student

Student :: Person - *subclass relationship*

Student : EntityType
Person : EntityType

Class & instance at the same time

Adapted from a tutorial on Programming with Logic and Frames by M. Kifer
Methods and Queries

**Methods:** like attributes, but take arguments

\[
?P[ageAsOf(?Year) \rightarrow ?Age] : -
?P:Person, ?P[born \rightarrow ?B], ?Age is ?Year-?B.
\]

- Attributes can be viewed as methods with no arguments

**Query:**

_**John’s children who were born when he was 30+ years old:**_

\[
? - John[born \rightarrow ?Y, children \rightarrow ?C],
\]
\[
\]

or

\[
? - John[ageAsOf(?Y) \rightarrow 30, children \rightarrow ?C] ,
\]
\[
\]

Adapted from a tutorial on Programming with Logic and Frames by M. Kifer
Domain and Range Restrictions

**Type signatures:** Define the types for method arguments and for their results

Person[
  born => integer,
  ageAsOf(integer) => integer,
  name => string,
  address => string,
  children => person].

Signatures can be queried:

?- Person[name => ?Type].

Answer: ?Type = string

?- Person[?Attr => string].

Answer: ?Attr = name

  ?Attr = address

Adapted from a tutorial on Programming with Logic and Frames by M. Kifer
Syntax

• Object ids:
  • Terms like in Prolog, but constants, functions can be capitalized – John, abc, f(john,34), Car(red,20000)
  • Below, O, C, M, Tobj, ... denote usual first order terms
• IsA hierarchy (isa-atoms):
  • O:C -- object O is a member of class C
  • C::S -- C is a subclass of S
• Structure (object-atoms):
  • O [Method -> Value] -- invocation of method
• Type (signature-atoms):
  • Class [Method => Class] -- a method signature
• Combinations of the above:
  • V, A, negation, quantifiers

Adapted from a tutorial on Programming with Logic and Frames by M. Kifer
Inheritance

- Inheritance of *structure* vs. inheritance of *behavior*
  - *Structural inheritance* = inheritance of the signature of a method
  - *Behavioral inheritance* = inheritance of the definition of a method

- Attributes/methods can be inheritable and non-inheritable (think of *static* vs. *instance* methods in Java)
  - *Non-inheritable method*: statements about the method in a class, \( c \), imply nothing for the subclasses of \( c \)
  - *Inheritable method*: its type and definition are inherited from a class, \( c \), to
    - the subclasses of \( c \) as inheritable methods
    - the objects of \( c \) as non-inheritable methods
Examples of Inheritance

Inheritable methods are inherited as
- *inheritable* to subclasses
- *non-inheritable* to members

Adapted from a tutorial on Programming with Logic and Frames by M. Kifer
Inheritance in the presence of Rules

- Inheritance is hard to define properly in the presence of rules.

Inherited
defeated??
derived

\[ a \left[ m \rightarrow v \right] \]
\[ b \left[ m \rightarrow v \right] \]
\[ c \left[ m \rightarrow w \right] \]
\[ c[m \rightarrow w] : b[m \rightarrow v] \]

Credulous vs Skeptical accounts exist

Adapted from a tutorial on Programming with Logic and Frames by M. Kifer
HiLog: Higher Order Logic

- Allows certain forms of logically clean meta-programming
  - Very useful for schema browsing
- Syntactically appears to be higher-order, but semantically is first-order and tractable
- Has sound and complete proof theory
Examples of Hilog

Variables over predicates and function symbols:
\[ p(?X,?Y) : - ?X(a,?Z), ?Y(?Z(b)). \]

Variables over atomic formulas (reification):
\[ call(?X) : - ?X. \]

Defining transitive closure:
\[ closure(?P)(?X,?Y) :- ?P(?X,?Y) ; \\
    closure(?P)(?X,?Y) :- ?P(?X,?Z) \text{ and } closure(?P)(?Z,?Y) \]
Reasoning Algorithm

- Based on a procedure called SLG resolution that uses resolution and a form of non-monotonic reasoning called well founded semantics (to be covered in the next lecture)

Consider the following set of clauses:

| i.    | a :: b                      |
| ii.   | p(a)                       |
| iii.  | c [m @ b ⇒ (v, w)]         |
| iv.   | r [attr → a]               |
| v.    | r [attr → f(S)] ∨ ¬p(X) ∨ ¬O [M @ X ⇒ S] |
| vi.   | ¬p(f(Z))                   |

We can refute the above set using the following sequence of derivation steps, where θ denotes the unifier used in the corresponding step:

| vii.  | r [attr → f(S)] ∨ ¬O [M @ a ⇒ S]  | by resolving (ii) and (v); θ = {X\a} |
| viii. | c [m @ a ⇒ (v, w)]              | by input restriction from (i) and (iii) |
| ix.   | r [attr → f(v)]                 | by resolving (viii) with (vii); θ = {O\c, S\v, M\m} |
| x.    | a ≡ f(v)                        | by the rule of scalarity, using (iv) and (ix) |
| xi.   | p(f(v))                         | by paramodulation, using (ii) and (x) |
| xii.  | □                               | by resolving (vi) with (xi); θ = {Z\v} |
F-Logic Implementations

- **FLORA (SUNY Stonybrook)** – by M. Kifer and students
- (U. Melbourne – M. Lawley) – early 90’s; first Prolog-based implementation
- **FLORID** (U. Freiburg – Lausen et al.) – late 90’s; the only C++ based implementation
- **FLIP** (U. Freiburg – Ludaescher) – late 90’s; first XSB based implementation. Inspired the **FLORA** effort
- **TFL** (Tech. U. Valencia – Carsi) – late 90’s; first attempt at F-logic + Transaction Logic
- **SILRI** (Karlsruhe – Decker et al.) – late 90’s; Java based
- **TRIPLE** (Stanford – Decker et al.) – early 2000’s; Java
- **OntoBroker** (Ontoprise.de) – 2000; commercial
Summary

• Frames and description logics have attractive computational properties but there are several classes of knowledge they cannot represent
  – Extending their representation power and still retaining good computational properties is an active area of KR&R research
• SILK/F-Logic is a state-of-the-art representation language that provides one possible attractive design for the combination
Recommended Reading

• Optional