
9b.

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

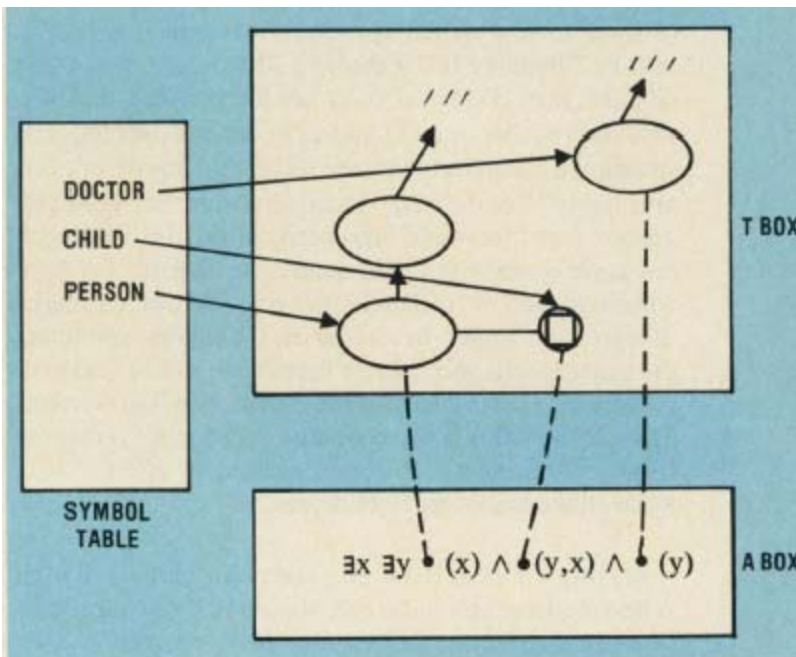
- Polyadic predicates
 - Relations with arity greater than two
 - A is between B and C
- Limited relational expressivity
 - DLs can model only domains where objects are connected in a tree like manner
 - “an uncle of a person is a brother of that person’s father”
- Disjunction
 - Many concepts have disjunctively 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 \equiv [And Professional
[Exists 1 MedicalDegree]
.....]

Person \equiv Human

Exists x, y Person $(x) \wedge$ child $(y, x) \wedge$ 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

T Box:

Tell: $KB \times \text{Symbol} \times \text{Term} \rightarrow KB$

(By symbol, I mean term.)

*Ask*₁: $KB \times \text{Term} \times \text{Term} \rightarrow \{\text{yes, no}\}$

(Does term₁ subsume term₂?)

*Ask*₂: $KB \times \text{Term} \times \text{Term} \rightarrow \{\text{yes, no}\}$

(Is term₁ disjoint from term₂?)

A Box:

Tell: $KB \times \text{Sentence} \rightarrow KB$

(Sentence is true.)

Ask: $KB \times \text{Sentence} \rightarrow \{\text{yes, no, unknown}\}$

(Is sentence 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.	$Boy(x) \supset Person(x)$
	2.	$[Boy(x) \wedge Sex(x, y)] \supset Male(y)$
	3.	$Girl(x) \supset Person(x)$
	4.	$[Girl(x) \wedge Sex(x, y)] \supset Female(y)$
	5.	$[NoSon(x) \wedge Child(x, y)] \supset Girl(y)$
	6.	$[NoDaughter(x) \wedge Child(x, y)] \supset Boy(y)$
	7.	$Person(x) \supset Sex(x, sk1(x))$
	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.	$Sex(sk2, x) \supset Male(x)$
resolve 14 and 17	18.	$Male(sk1(sk2))$
resolve 8 and 18	19.	$\neg Female(sk1(sk2))$
resolve 12 and 4	20.	$Sex(sk2, x) \supset Female(x)$
resolve 14 and 20, simplify by 19	21.	\square

Figure 1: Resolution Proof for *Childless* Problem

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.	\square

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 Id

Attribute

Object description:

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

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

Attribute

Structure can be nested:

Sally[*spouse* -> John[*address* -> '123 Main St.']]

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

Methods and Queries

Methods: like attributes, but take arguments

$?P[\textit{ageAsOf}(?Year) \rightarrow ?Age] :-$

$?P:\textit{Person}, ?P[\textit{born} \rightarrow ?B], ?Age \textit{ is } ?Year - ?B.$

- Attributes can be viewed as methods with no arguments

Query :

John's children who were born when he was 30+ years old:

$?- \textit{John}[\textit{born} \rightarrow ?Y, \textit{children} \rightarrow ?C],$
 $?C[\textit{born} \rightarrow ?B], ?B > ?Y + 30.$

or

$?- \textit{John}[\textit{ageAsOf}(?Y) \rightarrow 30, \textit{children} \rightarrow ?C],$
 $?C[\textit{born} \rightarrow ?B], ?B > ?Y.$

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

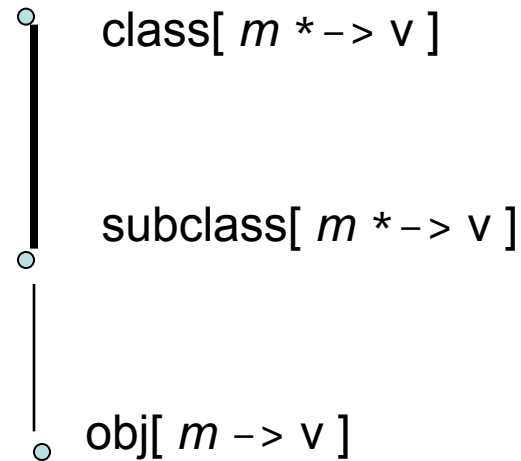
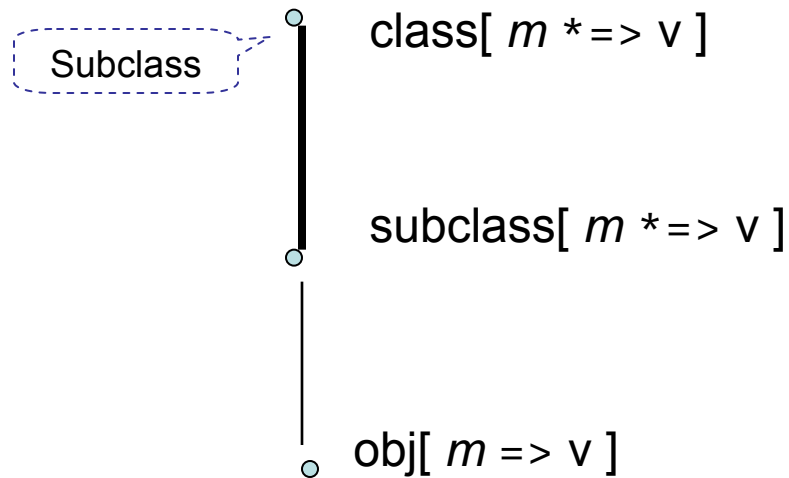
Syntax

- 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:
 - \forall, \wedge , negation, quantifiers

Inheritance

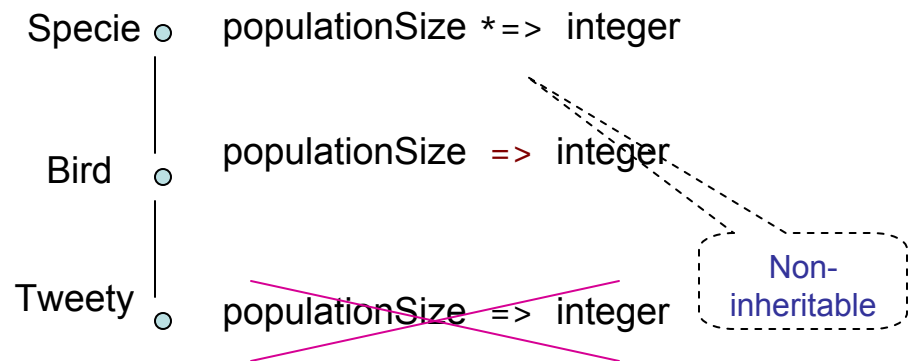
- Inheritance of *structure* vs. inheritance of *behavior*
 - **Structural inheritance** = inheritance of the signature of a method
 - **Behavioral inheritance** = inheritance of the definition of a method
- Attributes/methods can be inheritable and non-inheritable (think of *static* vs. *instance* methods 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



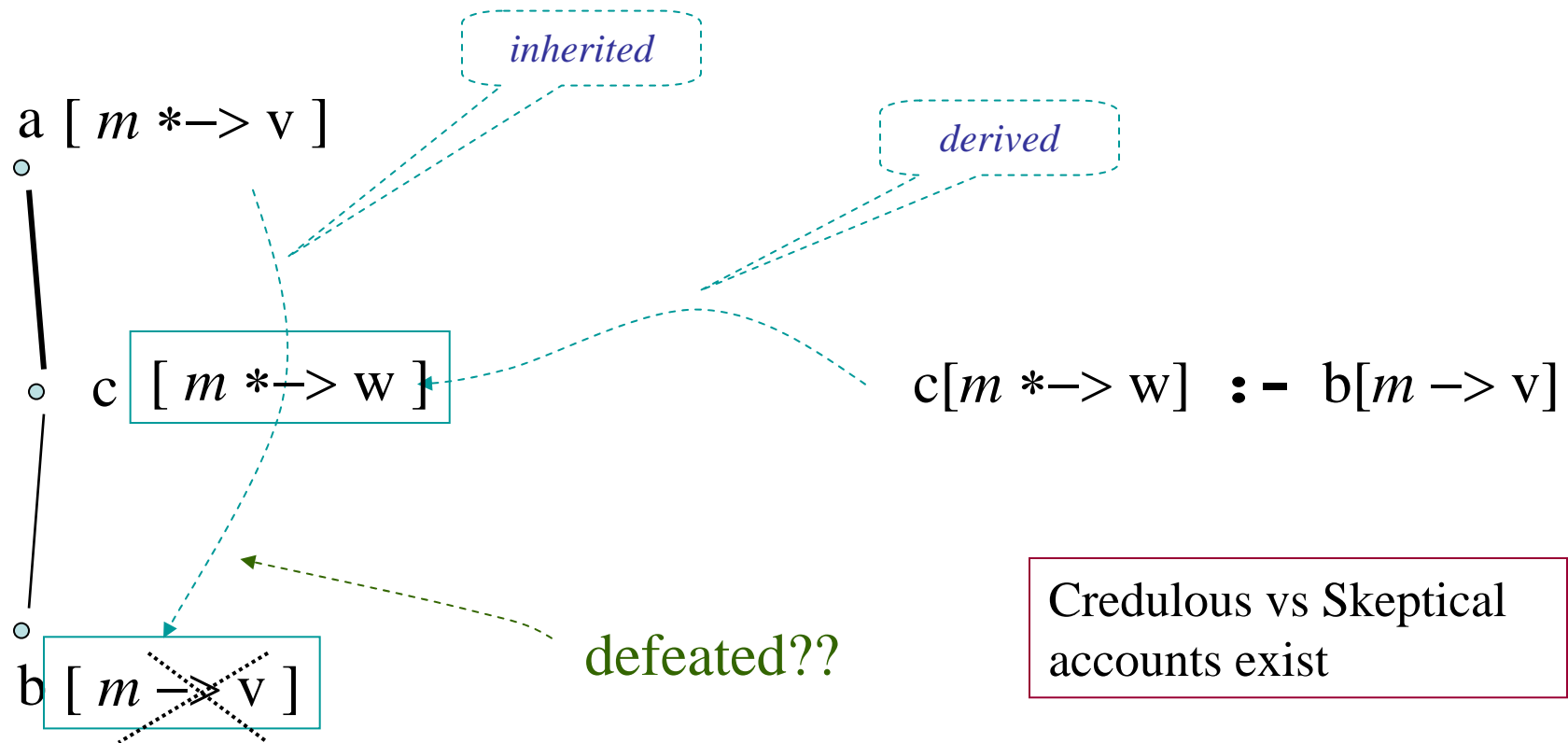
Inheritable methods are inherited as

- *inheritable* to subclasses
- *non-inheritable* to members



Inheritance in the presence of Rules

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



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
- [Chen,Kifer,Warren, HiLog: A Foundation for Higher-Order Logic Programming, J. of Logic Programming, 1993]

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. | $a :: b$ | iv. | $r [attr \rightarrow a]$ |
| ii. | $p(a)$ | v. | $r [attr \rightarrow f(S)] \vee \neg p(X) \vee \neg O [M @ X \Rightarrow S]$ |
| iii. | $c [m @ b \Rightarrow (v, w)]$ | vi. | $\neg p(f(Z))$ |

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

- | | | |
|-------|---|--|
| vii. | $r [attr \rightarrow f(S)] \vee \neg O [M @ a \Rightarrow S]$ | by resolving (ii) and (v); $\theta = \{X \setminus a\}$ |
| viii. | $c [m @ a \Rightarrow (v, w)]$ | by input restriction from (i) and (iii) |
| ix. | $r [attr \rightarrow f(v)]$ | by resolving (viii) with (vii); $\theta = \{O \setminus c, S \setminus v, M \setminus m\}$ |
| x. | $a \doteq f(v)$ | by the rule of scalarity, using (iv) and (ix) |
| xi. | $p(f(v))$ | by paramodulation, using (ii) and (x) |
| xii. | \square | by resolving (vi) with (xi); $\theta = \{Z \setminus v\}$ |
-

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 $\mathbb{F}\mathbb{L}\mathbb{O}\mathbb{R}\mathbb{A}$ 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
 - Can OWL and logic programming live together happily ever after by Boris Motik, Ian Horrocks, Riccardo Rosati and Ulrike Sattler. In International Semantic Web Conference, 2006
-