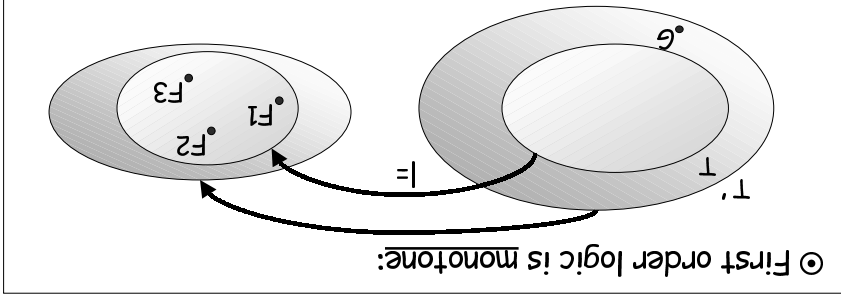


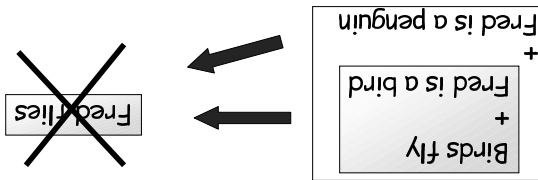
Logic Programming

Automated Reasoning in practice

Motivation: monotonicity



© But: people seldom reason monotonically!



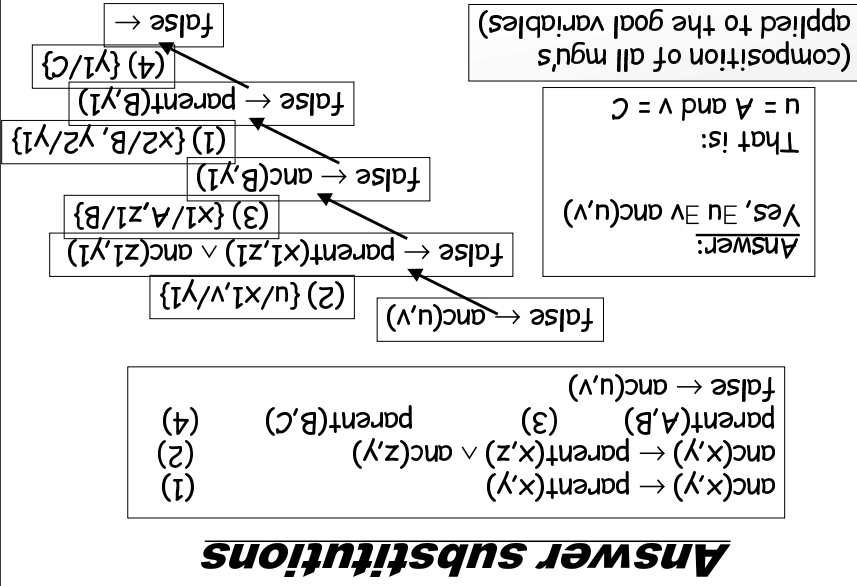
2

Default reasoning:

- © Is a form of reasoning that we'd like to use permanently
- ↳ otherwise the rules are far too complex!
- © Is usually supported by hierarchy, inheritance and exceptions in OOP
- ↳ also an early AI-formalism
- © CANNOT be expressed in FOL
- ↳ independent on the knowledge representation !!

- © CAN be expressed in some FOL extensions
- ↳ non-monotone logics
- ↳ the simplest one of those is ...

3



Answer substitutions

The link to programming

Logic Programming

- Resolution-based automated reasoning:
 - restricted to Horn clauses
 - restricted to backwards linear resolution
- BUT: with 3 important new extensions:
 - return Answer Substitutions
 - Least-model semantics instead of standard FOL model semantics
 - Extending Horn clause logic with Negation as Finite Failure

Practical programming?

⊙ Example arithmetic:

| |
|-----------------------------------|
| double_plus_1(x,y) → y is 2*x + 1 |
| false → double_plus_1(3,z) |
| false → double_plus_1(2,5) |

⊙ Example lists:

| |
|--|
| append([], list, list) → |
| append([x list1, list2, [x list3]) → |
| append(list1, list2, list3) |

| |
|-----------------------------------|
| false → append([1,2], [3,4,5], z) |
| false → append([1,2], y, [1,2,3]) |
| false → append(x, y, [1,2]) |

| |
|----------|
| Yes: z=7 |
| Yes |

| |
|------------------------|
| Yes: z = [1,2,3,4,5] |
| Yes: y = [3] |
| Yes: x = [], y = [1,2] |
| x = [1], y = [2] |
| ... |

Logic PROGRAMMING

⊙ By computing answer substitutions Logic Programming serves as a basis for a number of "general purpose" programming languages.

→ Prolog, Mercury, XSB, ...

with an efficiency comparable with C!

→ and even faster for some programs.

8

And computes ALL answers

(1) anc(x,y) → parent(x,y)
 (2) anc(x,y) → parent(x,z) ∧ anc(z,y)
 (3) parent(A,B)
 (4) parent(B,C)

Diagram illustrating the computation of all answers for the query anc(x,y):

```

  graph TD
    Q["(1) {u/x1,v/y1}"] --> R1["false ← anc(u,v)"]
    Q --> R2["false ← parent(x1,y1)"]
    R2 --> R3["(3) {x1/A,y1/B}"]
    R2 --> R4["false ←"]
    R3 --> R5["(4) {x1/B,y1/C}"]
    R3 --> R6["false ←"]
    R5 --> R7["The third answer: u = B and v = C"]
    R5 --> R8["Another answer: u = A and v = B"]
  
```

7

Least model semantics

Specify in a more compact way



Least model semantics

⊙ Example: a database:

| | |
|-------------------|--------------------|
| celebrity(Crabe) | celebrity(Jambers) |
| celebrity(Lisa) | celebrity(Tielman) |
| celebrity(Samson) | |

⊙ Is Deschreye a celebrity ??

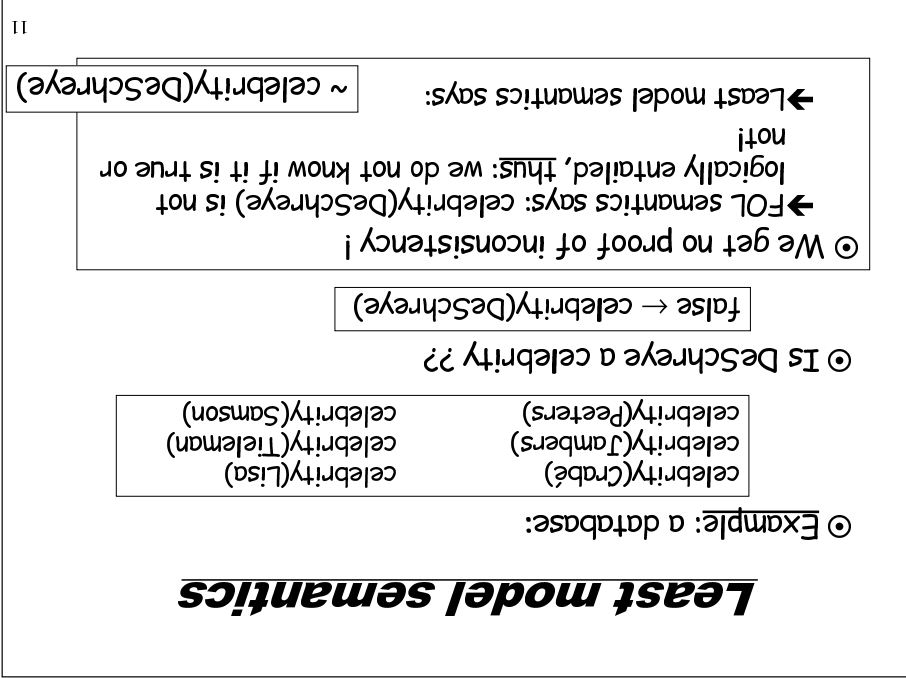
false ← celebrity(Deschreye)

⊙ We get no proof of inconsistency !

→ FOL semantics says: celebrity(Deschreye) is not logically entailed, thus: we do not know if it is true or not!

→ Least model semantics says:

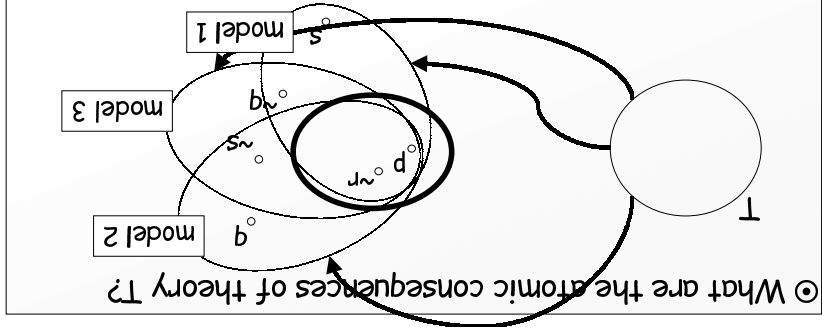
~ celebrity(Deschreye)



11

Formally: the idea

⊙ What are the atomic consequences of theory T?



→ In FOL:

Consequences are in the intersection: p en ~r.
We do not know anything about the truth of q and s.

→ In LP:

Consequences are in the intersection: p en ~r.
Other predicates are NOT true: ~q and ~s.

12

⊙ Also: some concepts can be completely axiomatized in LP but not in FOL. ← Ex.: the natural numbers!

⊙ Knowledge is differently represented in these 2 formalisms.

⊙ In FOL: {smart(Kelly)} implies neither strong(Kelly) nor ~strong(Kelly)
 ⊙ In LP: {smart(Kelly)} implies ~strong(Kelly)
 ⊙ In particular: LP is a non-monotone logic !!
 ← In {smart(Kelly), strong(Kelly)} is ~strong(Kelly) no longer entailed.

How relevant is the change of semantics?

⊙ In other words: Logic Programming supports formulating definitions of the concepts. ← Not just state what is true about these concepts (=FOL)!

→ The Closed World Assumption! ◆ (= everything not entailed by the theory is false)

⊙ Logic programming provides a compact way to express 'complete knowledge' on some subject.
 ⊙ If you do not say that something is true, than it is false.

The "closed" assumption

⊙ The logic program:

$$\forall x \text{celebrity}(x) \leftrightarrow (x = \text{Crabé}) \vee (x = \text{Jambers}) \vee (x = \text{Peeters}) \vee (x = \text{Lisa}) \vee (x = \text{Tielman}) \vee (x = \text{Samson})$$

or also to:
 is equivalent to the infinite FOL theory:

| | | | | |
|--------------------|--------------------|---------------------|----------------------|-------------------|
| celebrity(Crabé) | celebrity(Jambers) | celebrity(Lisa) | celebrity(Tielman) | celebrity(Samson) |
| celebrity(Peeters) | celebrity(Dalil) | celebrity(Janssens) | celebrity(Deschreye) | celebrity(Cobain) |
| ... | ... | ... | ... | ... |

Relation with FOL

⊙ This is meaningful only with the least model semantics (where everything that is not proven to be 'true', is considered to be 'false')

If all attempts to prove B using linear LP-resolution, fail in a finite time, conclude $\text{not}(B)$.

⊙ $\text{not}(B)$ means:

⊙ Not the meaning of standard negation

Meaning of negation as finite failure

Here: Introduce negations in bodies!

- something different from FOL!
- (but: with the least model semantics we will get
- both give complete predicate logic!
- ◆ allow negation before the body atoms
- ◆ allow disjunctions in the heads

← equivalent:

⊙ How?

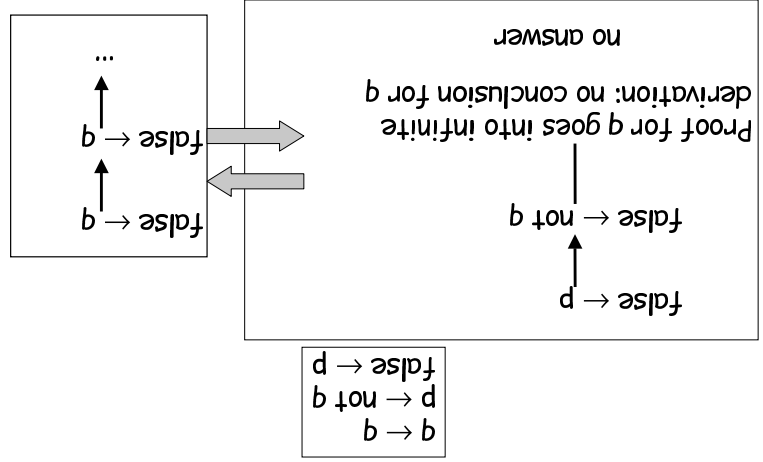
← extending the representation power of Logic Programming beyond the Horn clauses logic

⊙ The basic idea:

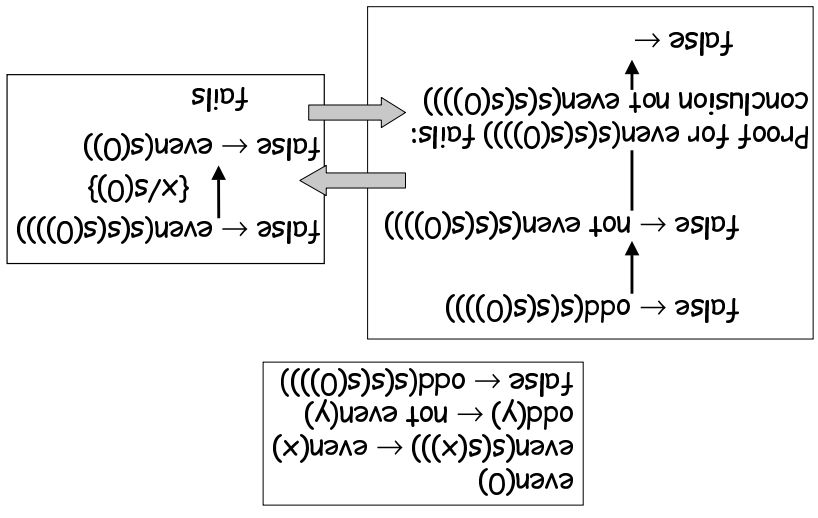
Negation as finite failure

Negation as finite failure

⊙ But $\sim q$ is true according to the least model semantics!

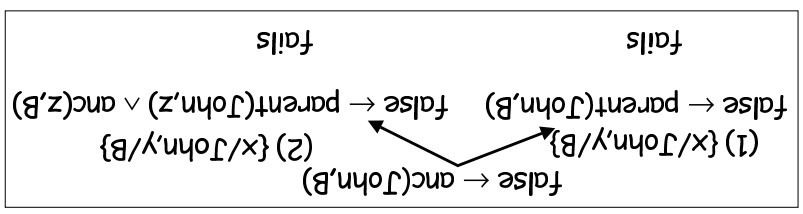


And another example



Another example

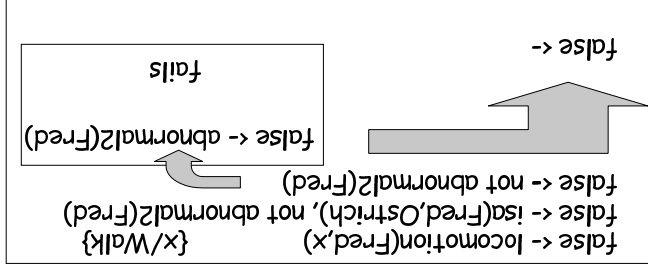
⊙ Conclusion: not anc(John,B)



⊙ Try to prove that "anc(John,B)" holds!

- (1) anc(x,y) -> parent(x,y)
- (2) anc(x,y) -> parent(x,z) v anc(z,y)
- (3) parent(A,B)
- (4) parent(B,C)

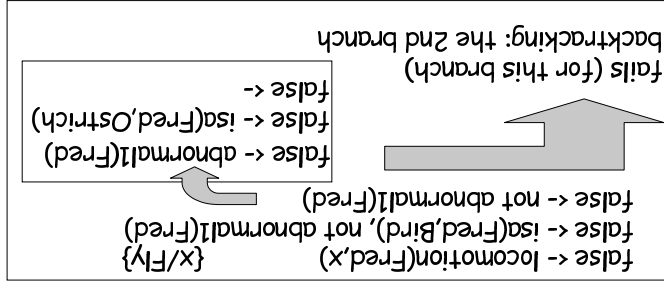
The ancestor example



Also known: isa(Fred,Ostrich), Prove that: $\exists x$ locomotion(Fred,x)

$locomotion(x,Fly) \rightarrow isa(x,Bird), not\ abnormal1(x)$
 $locomotion(x,Walk) \rightarrow isa(x,Ostrich), not\ abnormal2(x)$
 $isa(x,Bird) \rightarrow isa(x,Ostrich)$
 $abnormal1(x) \rightarrow isa(x,Ostrich)$
 $isa(Fred,Bird)$

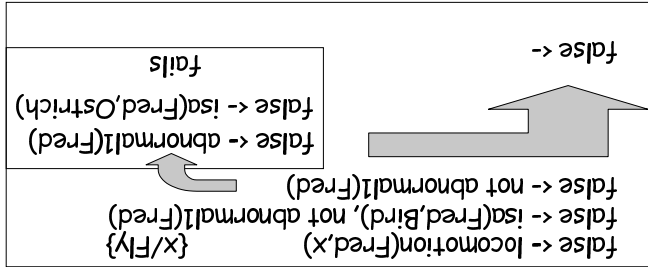
Default reasoning (3):



Also known: isa(Fred,Ostrich), Prove that: $\exists x$ locomotion(Fred,x)

$locomotion(x,Fly) \rightarrow isa(x,Bird), not\ abnormal1(x)$
 $locomotion(x,Walk) \rightarrow isa(x,Ostrich), not\ abnormal2(x)$
 $isa(x,Bird) \rightarrow isa(x,Ostrich)$
 $abnormal1(x) \rightarrow isa(x,Ostrich)$
 $isa(Fred,Bird)$

Default reasoning in LP (2):



Also known: isa(Fred,Bird), Prove that: $\exists x$ locomotion(Fred,x)

$locomotion(x,Fly) \rightarrow isa(x,Bird), not\ abnormal1(x)$
 $locomotion(x,Walk) \rightarrow isa(x,Ostrich), not\ abnormal2(x)$
 $isa(x,Bird) \rightarrow isa(x,Ostrich)$
 $abnormal1(x) \rightarrow isa(x,Ostrich)$

Default reasoning in LP (1):

⊙ A number of languages: CHIP, Eclipse, Sicsus, etc.

⊙ Advantages of Constraint solving for *efficient* problem solving

⊙ Advantages of Logic for knowledge representation

⊙ Integrate constraint processing techniques (consistency, forward checking, looking ahead, ...) with Logic Programming.

Constraint Logic Programming

⊙ **Combine!** ← Open Logic Programming

- ◆ LP-definitions for the part for which you have a complete knowledge,
- ◆ FOL formulae for the rest.

⊙ Logic Programming is very useful if you have a COMPLETE knowledge over your predicates

⊙ FOL is very useful if your knowledge is INCOMPLETE

Beyond FOL and Logic Programming

⊙ A specific programming language based on LP. Uses a depth-first strategy to search linear resolution proofs.

→ incomplete

- ◆ can get stuck in infinite branches

⊙ Has a bunch of built-in predicates (sometimes without logical meaning) for:

→ Numerical computations, input-output, changing the search strategy, meta-programming, etc.

⊙ More recent LP languages: Goedel, Mercury, Hal, ..

Prolog