Data mining, machine learning, and uncertainty reasoning

林偉川

Homework

- Y=C’B’A+C’BA’+CB’A+CBA’
  ➔B’A+BA’
- Y=ABCD+A’BCD+ABCD’+ABC’D
  ➔BCD+ABC+ABD
- Y=A’B’CD+A’BC’D’+A’BCD’+A’BCD+ABC’D+ABCD+ABCD’+AB’C’D
  ➔A’CD+A’BD’+AC’D+BC

Positive and negative examples for concept learning

example	at1	at2	at3	Classification
e1	a	x	n	⊕ (positive)
e2	b	x	n	⊕
e5	a	y	m	⊕
e7	b	y	m	⊕
e8	a	x	m	⊕
e6	b	y	n	⊗
e3	a	y	n	⊗ (negative)
e4	a	z	n	⊗

AQ algorithm

- Select 1st seed: e1 (at1=a, at2=x, at3=n)
  pick 1st negative example: e6 (at1=b, at2=y, at3=n). To create the star of seed e1 (the set of maximally general description of e1), begin by creating the set of all descriptions of e1 that do not cover f1

  R1: (at1=a) R2: (at2=x)

  R1: ~ (at1=b) R2: ~(at2=y)

AQ algorithm

- Each of descriptions also covers some of negative examples. These rules are specialized so as to exclude these negative examples
- Multiplying out the current rules by the negations of negative examples and applying absorption law

  R1: at2=x Λ at3=n e1,e2

  R2: at2=(x V y) Λ at3=m e5,e7,e8
- This is the star of e1
- R2 covers 3 positive examples and R1 covers only 2 examples ➔ R2 is selected

AQ algorithm

- The next step is to select a new seed from still uncovered positive example ➔ only e2 exists
- Select next seed: e2 ➔ 2 rules are generated and covers examples are the same as R1
- Optimizes the assumed preference criterion:

  R1: at2=x Λ at3=n e1,e2

  R2: at2=(x V y) Λ at3=m e5,e7,e8

Visualization of the example

R1

R2'

				at2
⊕	⊕		⊕	x
⊗	⊗	⊕	⊕	y
	⊗			z
b	a	b	a	at_1
n		m		at_3

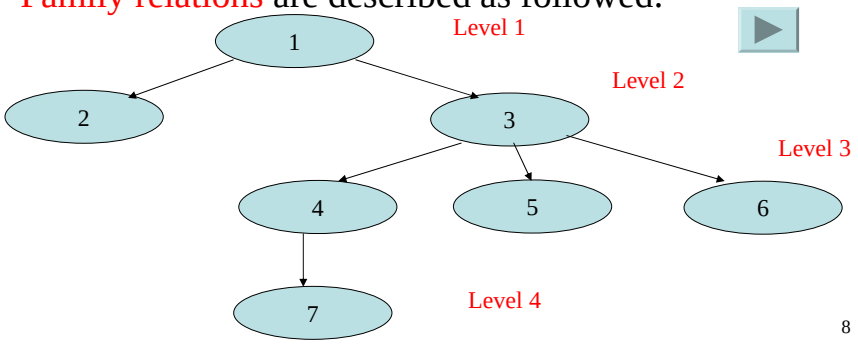
Match solutions: at2=x V 【(at2=y) Λ (at3=m)】

R1: at2=x Λ at3=n e1,e2

R2: at2=(x V y) Λ at3=m e5,e7,e8

How can predicate logic be used?

- Attribute-value languages are very useful but have their limitations
- Logical descriptions with predicate that describe relations among objects or their parts are certainly more powerful
- Family relations are described as followed:



Family relation

- Family relations are described by the single predicate parent(x,y) (first-order logic)
- Parent(1,2) means 1 is the parent of 2
- The family tree can be encoded by the following set of relationships:

  parent={(1,2)(1,3)(3,4)(3,5)(3,6)(4,7)}
- Suppose that with these relationships in the background knowledge, the system wants to learn the meaning of grandparent(x,y)

Family relation

- The teacher provides the following positive examples of the concept:

  grandparent(1,4), grandparent(1,5), grandparent(1,6), grandparent(3,7)

  ➔ these are easier to encode by the set of relationships: grandparent={(1,4)(1,5)(1,6)(3,7)}, all other family relations among the persons 1,…7 are negative examples of grandparent

Family relation

- Each possible pair of persons (X,Y) can be represented by a Boolean attribute whose truth value is determined by the truth value of the predicate relation parent(X,Y)
- Inductive Logic Programming (ILP) is currently an intensively studied branch of machine learning

Learning Horn clauses from relations

- Write a program which is capable of learning the concept of grandparent from examples of family relationships ➔ described by Horn clause
- What learning strategies would be applied?
- Assume that the concept descriptions is a set of Horn clauses in the form of:

C_1 :- L_{11}, L_{12}, … L_{1m}

C_2 :- L_{21}, L_{22}, … L_{2m} …

(comma indicates that literals are linked by conjunction)

Learning Horn clauses from relations

- The predicate C_i is the head of a clause and the literals L_{ij} form the body of the clause
- Each of the literals represents a relation that has n >= 0 arguments
- Whether divide-and-conquer or the AQ method, the learner started with a relatively general description involving a single attribute, and gradually specialized by adding more condition

Learning Horn clauses from relations

- Adding a literal to a clause body has a similar specializing effect as adding a condition to a decision rule in AQ or appending a node at the bottom of a decision tree
- A good strategy is to start with a clause consisting solely of the head and then specialize it by adding literals to its body

Learning Horn clauses from relations

- There are no other predicates in the background knowledge except for parent, only those variables are allowed in the body that also appear in the head [restriction], possible clauses to define grandparent in this language (by just 1 literal) are: (none of these clauses covers positive examples)

grandparent(X,Y) :- parent(X,Y)

grandparent(X,Y) :- parent(Y,X)

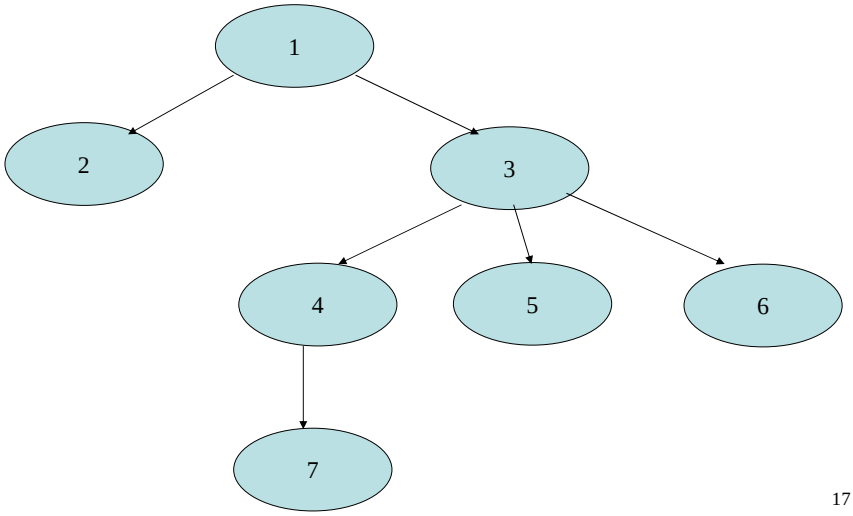
grandparent(X,Y) :- parent(X,X)

grandparent(X,Y) :- parent(Y,Y)

Learning Horn clauses from relations

- None of the above clauses cover any of the positive examples
- The restriction should be relaxed allowing for one argument that does not appear in the head of the clause
- For literals of this kind can be constructed: parent(X, Z), parent(Y, Z), parent(Z, X) and parent(Z, Y). Suppose the system select the following option:
  grandparent(X, Y) :- parent(X, Z)

Family Relation



Learning Horn clauses from relations

- Let's examine for the tripplets (X, Z, Y), satisfying this clause, the head of the clause represents a positive example and for which it represents a negative example ➔ in this family case, there are 7^3=343 possible tripplets (X, Z, Y)

  ⊕：(1,2,4) (1,2,5) (1,2,6) (1,3,4) (1,3,5) (1,3,6) (3,4,7) (3,5,7) (3,6,7) total 9 ⊕'s
  ➔(1,4) (1,5) (1,6) (3,7) 4 positive examples

Learning Horn clauses from relations

⊗： (1,2,1) (1,2,2) (1,2,3) (1,2,7) (1,3,1) (1,3,2) (1,3,3) (1,3,7) (3,4,1) (3,4,2) (3,4,3) (3,4,4) (3,4,5) (3,4,6) (3,5,1) (3,5,2) (3,5,3) (3,5,4) (3,5,5) (3,5,6) (3,6,1) (3,6,2) (3,6,3) (3,6,4) (3,6,5) (3,6,6) (4,7,1) (4,7,2) (4,7,3) (4,7,4) (4,7,5) (4,7,6) (4,7,7) total 33 ⊗'s

➔ (1,1)(1,2)(1,3)(1,7)(3,1)(3,2)(3,3)(3,4) (3,5)(3,6)(4,1)(4,2)(4,3)(4,4)(4,5)(4,6)(4,7) 17 negative examples

Assessment of learning algorithms

- If overspecialization, the learner will tend to misclassify positive testing examples more than negatives
- If overgeneralization, the learner will more frequently fail by misclassifying negative examples (classifying them as positives)
- The learner should develop a hypothesis (internal description of concept) that is consistent (does not cover any ⊗) and complete (covers all ⊕'s)

Learning from horn clause relation

- Grandparent(x,y) :- parent(x,z) is inconsistent because it also covers negative examples
- The inconsistency can be reduced by a properly chosen specialization of the clause
- Correct adding the extra literals as followed: Grandparent(x,y) :- parent(x,z), parent(z,y)
- This clause covers the following tripplets (x,y,z):
  ⊕ :- (1,3,4)(1,3,5)(1,3,6)(3,4,7)
  ⊗ :- none

Learning from horn clause relation

- All of the positives and none of the negatives are covered, the learner is satisfied with this clause and stops here
- If not add the literal parent(z,y) just another literal parent(y,z)?
- Grandparent(x,y) :- parent(x,z), parent(y,z) it does not cover any negative and positive examples ➔ a suitable criterion should be decided what literal is needed to added to a clause

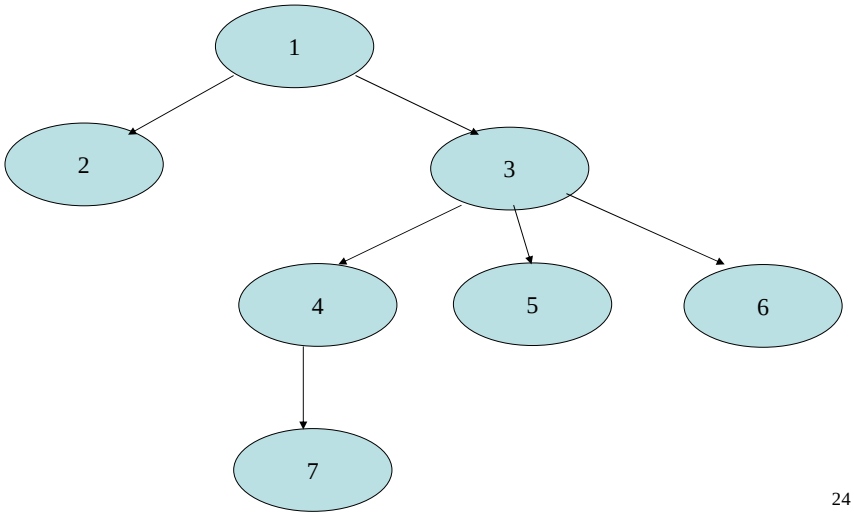
3 major theoretical properties of logical reasoning systems

(1)Soundness: To be confident that an inferred conclusion is "true"

(2)Completeness: To be confident that inference will eventually produce any true conclusion

(3)Tractability: To be confident that inference is feasible

Family Relation



How to decide the added literal

- What if ancestor? The positive examples are provided as followed:
  ancestor={(1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (3,4) (3,5) (3,6) (3,7) (4,7)}
- The search for the best literal arrives at the following description:
  ancestor(X,Y) :- parent(X,Y) this clause is consistent, but is incomplete!!!

How to decide the added literal

- It would have to find an alternative clause covering those positives that lie outside the reach of the first clause.
- The result is that at least one of the clauses should cover the example if it is to be a positive instance of the concept

```
ancestor(X,Y) :- parent(X,Y)
ancestor(X,Y) :- parent(X,Z), parent(Z,Y)
ancestor(X,Y) :-
        parent(X,Z),parent(Z,W),parent(W,Y)
```

How to decide the added literal

- It is incomplete because it just covers only 4 generations ➔A recursive description is needed
- ancestor(X,Y) :- parent(X,Y)
  ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y)
- To start with, the learner uses the predicate parent. After formulating the first clause, the learner also uses the predicate ancestor ➔ FOIL system kernel foundation

FOIL Algorithm

1. Initialize the clause by defining the head that represents the name of the concept to be learned and leave the body empty
2. While the clause covers negative examples do:
   find a "good" literal to be added to the clause body
3. Remove all examples covered by the clause
4. Add the clause to the emerging concept definition. If there are any uncovered positive examples then go to step 1.

How to find a good literal?

- Make use of information criterion similar to that employed in the induction of decision tree
- Denote by T_i^+ the number of ⊕'s covered by the conjunction $L_1, L_2, \ldots, L_{i-1}$ and denote by T_i^- the number of ⊗'s covered by the same conjunction
- The information provided by signaling a positive example among all the examples covered by this clause is:

$$I_i = -\log\left(\frac{T_i^+}{T_i^+ + T_i^-}\right)$$

How to find a good literal?

- After adding a new literal L_i this information becomes as followed:

$$I_{i+1} = -\log\left(\frac{T_{i+1}^+}{T_{i+1}^+ + T_{i+1}^-}\right)$$

- The success of literal L_i is measured by 2 factors:
  1. The remaining information I_{i+1}(the smaller, the better)
  2. The number $T_i^{++}$of ⊕ that remain covered by the clause after adding L_{i+1} (the higher, the better)
- FOIL's measure of success is defined as:

$$Gain(L_i) = T_i^{++} \times \left(I_i - I_{i+1}\right)$$

Conclusion of FOIL algorithm

- It carries out the AQ algorithm (progressive coverage), trying to cover the positive space by a set of clauses
- In the inner loop, a clause is stepwise specialized by a procedure controlled by a function similar to that used in the divide-and-conquer method

Inverse resolution

- FOIL exploit 2 search operators: the generalizing add_a_clause operator and the specializing add_a_literal operator
- Providing the complementary operators delete_a_clause for specialization and delete_a_literal for generalization, we arrive at the elementary repertoire of learning in first-order logic

Inverse resolution

- The operator can be exemplified by the following 4 steps, gradually changing the clause x :- a,b into x :- d,e

  - add a clause (after a literal)

$$x :- a, b \Rightarrow \left\{ \begin{array}{l} x :- a, b \\ x :- c, d \end{array} \right\}$$

  - delete a clause (delete the first literal)

$$\left\{ \begin{array}{l} x :- a, b \\ x :- c, d \end{array} \right\} \Rightarrow x :- c, d$$

  - add a literal (after the character) $x :- c, d \Rightarrow x :- c, d, e$
  - delete a literal (delete the first)

$$x :- c, d, e \Rightarrow x :- d, e$$

Inverse resolution

- The collection operators can be extended by the following alternative inductive search operators:

  - identification

$$\left\{ \begin{array}{l} a :- b, x \\ a :- b, c, d \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} a :- b, x \\ x :- c, d \end{array} \right\}$$

  - absorption

$$\left\{ \begin{array}{l} x :- c, d \\ a :- b, c, d \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} x :- c, d \\ a :- b, x \end{array} \right\}$$

Inverse resolution

- intra-construction

$$\left\{\begin{array}{l} a :- v, b, c \\ a :- w, b, c \end{array}\right\} \Rightarrow \left\{\begin{array}{l} a :- v, u \\ a :- w, u \\ u :- b, c \end{array}\right\}$$

- inter-construction

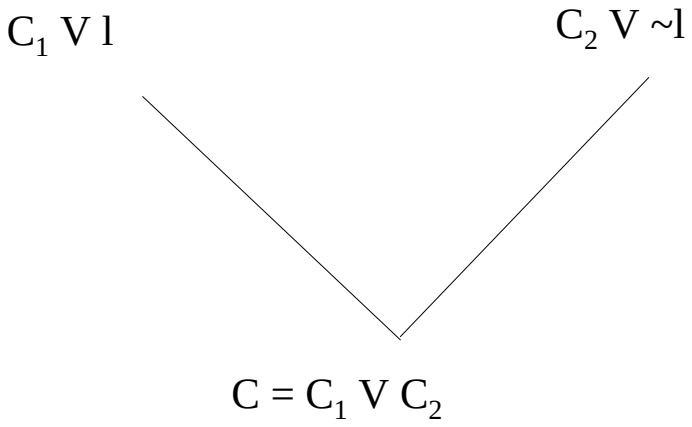
$$\left\{\begin{array}{l} a :- v, b, c \\ a :- w, b, c \end{array}\right\} \Rightarrow \left\{\begin{array}{l} a :- u, b, c \\ u :- v \\ u :- w \end{array}\right\}$$

- All of these operators can be derived from the resolution principle that is very popular in AI.

Inverse resolution

- Denote 2 disjunctions of predicate by C_1 and C_2 and denote an arbitrary predicate by l. The resolution principle is deductive and states the following rule:
  if (C_1 V l) is true and (C_2 V ~l) is true, then (C_1 V C_2) is also true
- One of them contains the literal l and the other one contains its negation ~l. The disjunction of the 2 expression, where l and ~l have been deleted is also true and called resolvent

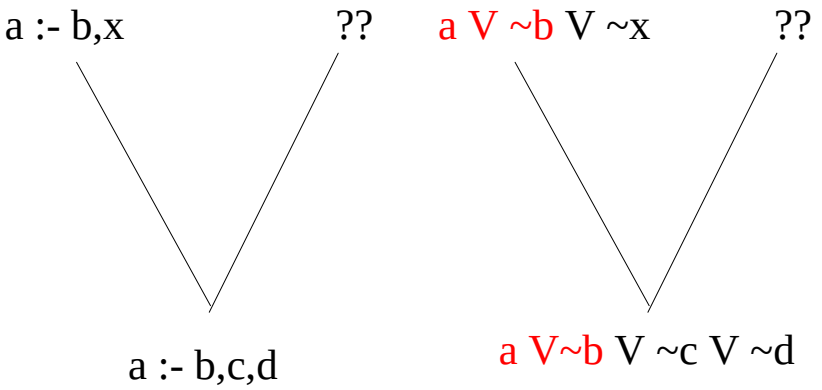
Inverse resolution



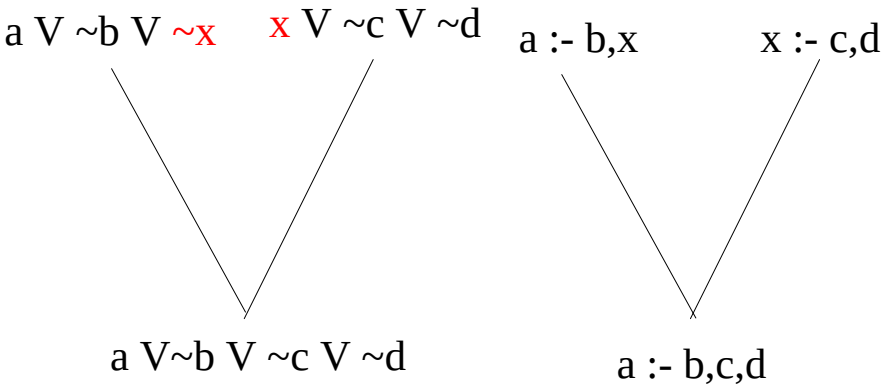
Inverse resolution

- The basic schema of resolution can be reversed
- Knowing the resolvent and one of the original strings, we construct the other original string
- Depending on whether the available clause contains the positive or the negated form of the predicate l.
- The formula A:-B can be rewritten as AV~B

Inverse resolution -- identification



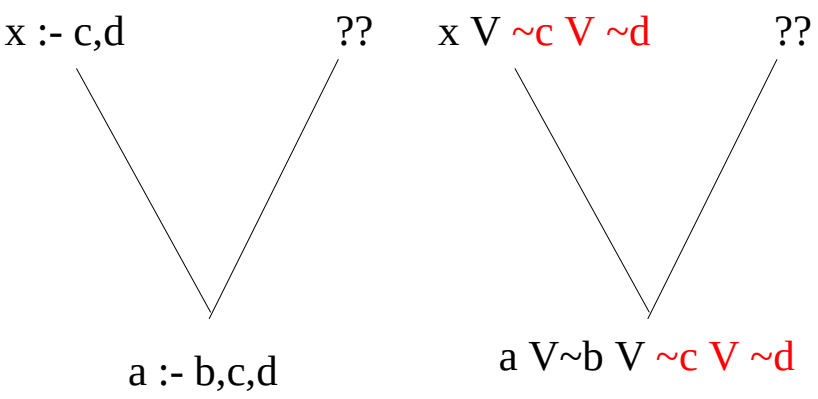
Inverse resolution -- identification



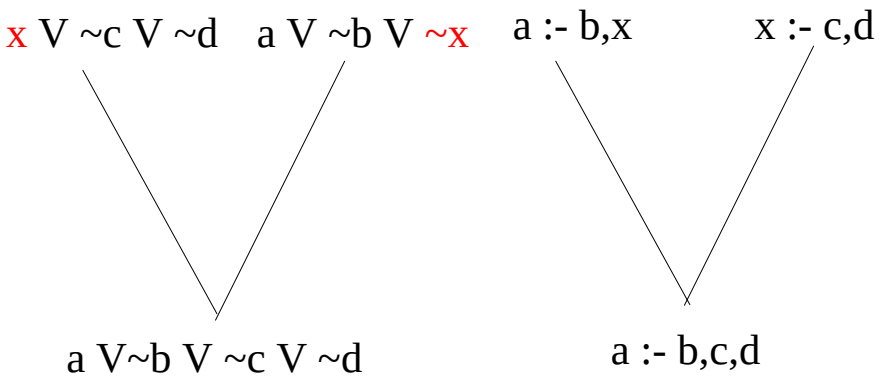
Inverse resolution

- a :- b can be rewritten as a V ~b. a :- b,x can be written as a V ~b V ~x. a :- b,c,d can be written as a V ~b V ~c V ~d.
- The resolvent contains the substring ~c V ~d that might have been inherited from the unknown clause
- The original contains the predicate ~x and x is expected to appear in the unknown clause
- Concatenating the contribution of the resolvent and the original
  ➔ x V ~c V ~d ➔ x:- c,d

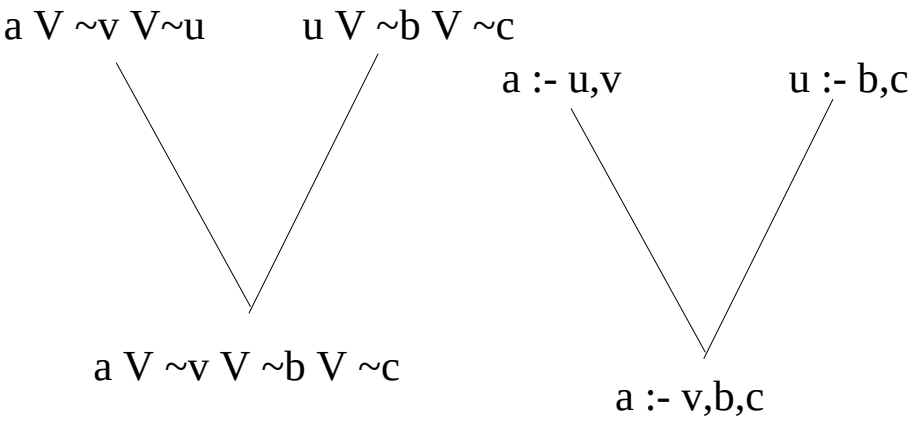
Inverse resolution -- absorption



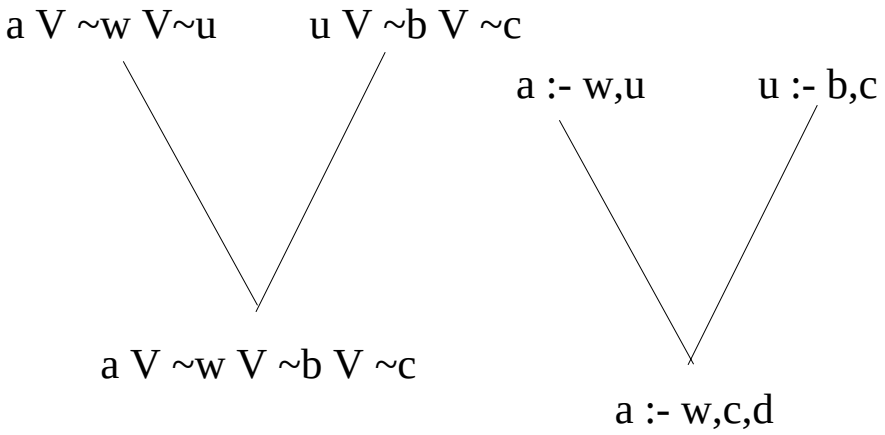
Inverse resolution -- absorption



Inverse resolution -- intra-construction



Inverse resolution -- intra-construction



Inverse resolution

- In real-world applications the task gets complicated
- The learner have to find proper substitutions of the arguments in the predicates l and ~l so that they are compatible
- The predicates p_1 = parent(john, bill) and p_2= parent(X, eve) are compatible only under the substitutions $\theta_1 = \{john / X, bill / Y\}$ in the first predicate, and $\theta_2 = \{eve / Y\}$ in the second predicate