Deductive Databases and Logic Programming (Sommer 2017)

Chapter 8: Negation

- Motivation, Differences to Logical Negation
- Syntax, Supported Models, Clark's Completion
- Stratification, Perfect Model
- Bottom-Up Computation
- Conditional Facts, Well-Founded Model
- Stable Models



After completing this chapter, you should be able to:

- explain the difference between negation in logic programming and negation in classical logic
- explain why stratification is helpful, check a given program for the stratification condition.
- compute supported, perfect, well-founded, and stable models of a given program.
- explain the well-founded semantics with conditional facts and elementary program transformations.



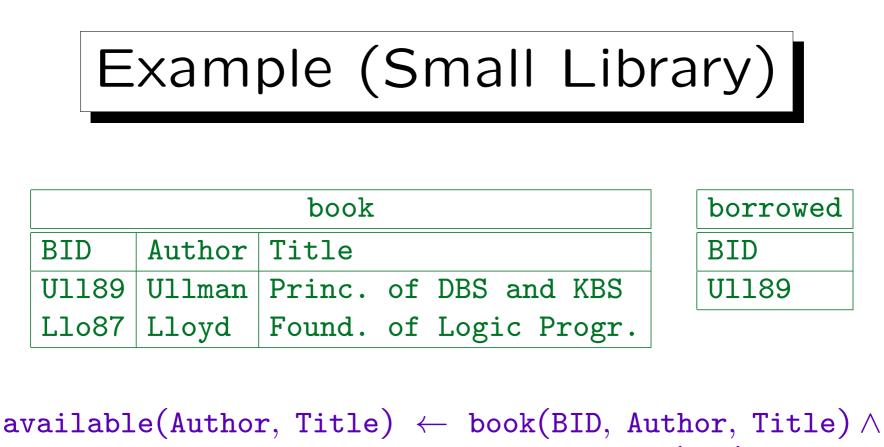


1. Motivation, Differences to classical logic)

2. Syntax, Supported Models

- 3. Stratification, Perfect Model
- 4. Conditional Facts, Well-Founded Model
- 5. Stable Models

Stefan Brass: Logic Programming and Deductive Databases



not borrowed(BID).

available

Lloyd Found. of Logic Progr.

Stefan Brass: Logic Programming and Deductive Databases

8-4



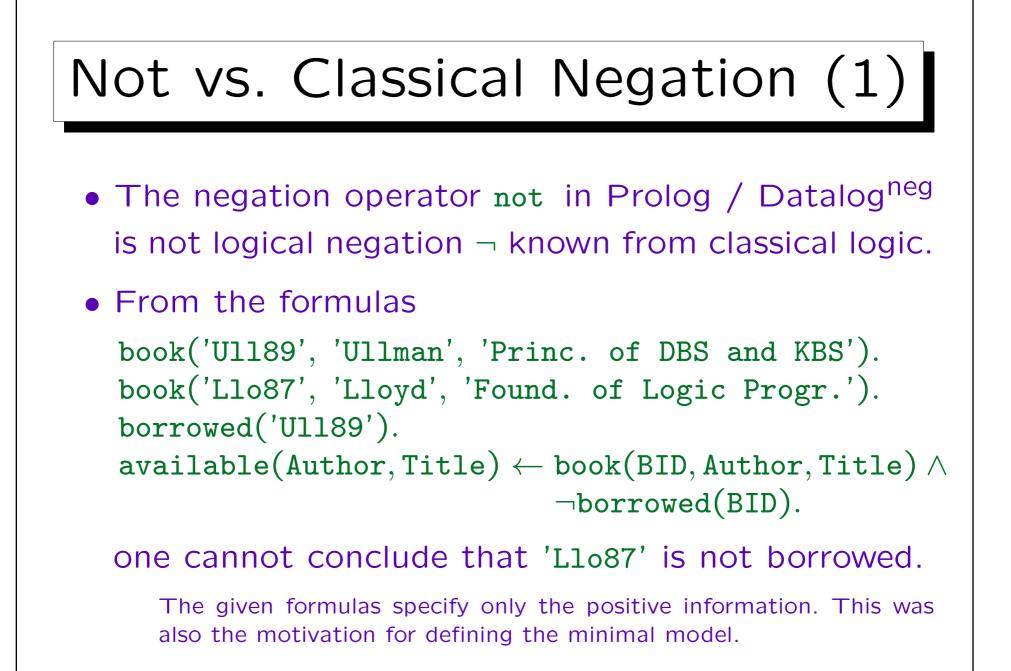
- Queries or view definitions as in the above example are possible in SQL, but cannot be expressed with definite Horn clauses (classical Datalog).
 - A good query language should be relationally complete, i.e. it should be possible to translate every relational algebra expression into that language.
 - ♦ This goal is reached for Datalog with negation (Datalog^{neg}) (even without recursion).
- Prolog has an operator not (also written \+).

Motivation (2)

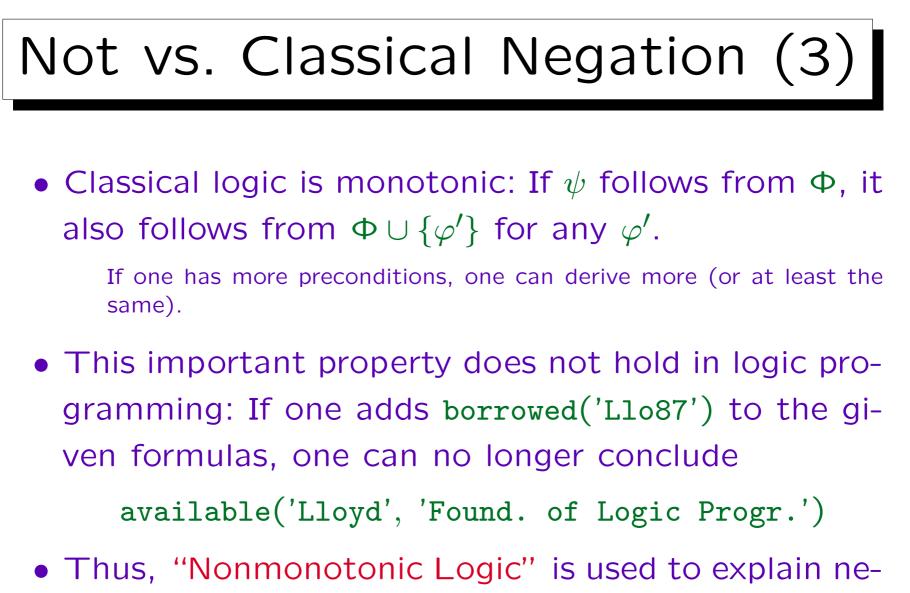
• Set difference is a natural operation. If it misses in a query language, the user will pose several queries and compute the set difference himself/herself.

If a query language computes sets, it should be closed under the usual set operations. I.e. a set operation applied to the result of two queries should be expressable as a single query. (One could also require other simple operations, such as counting.) For relations, it should be closed under relational algebra operations. This is not quite the same as relational completeness, because this closure condition holds also e.g. for recursive queries (not expressible in relational algebra).

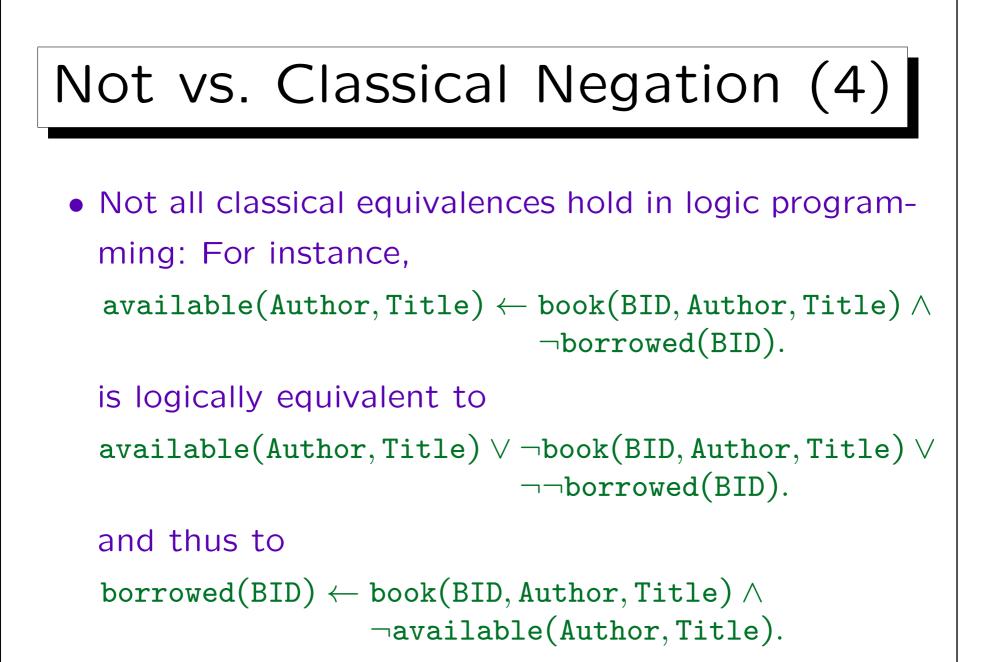
• It was defined above which facts are false in the minimal model. Up to now, knowledge about false facts cannot be used within in the program.



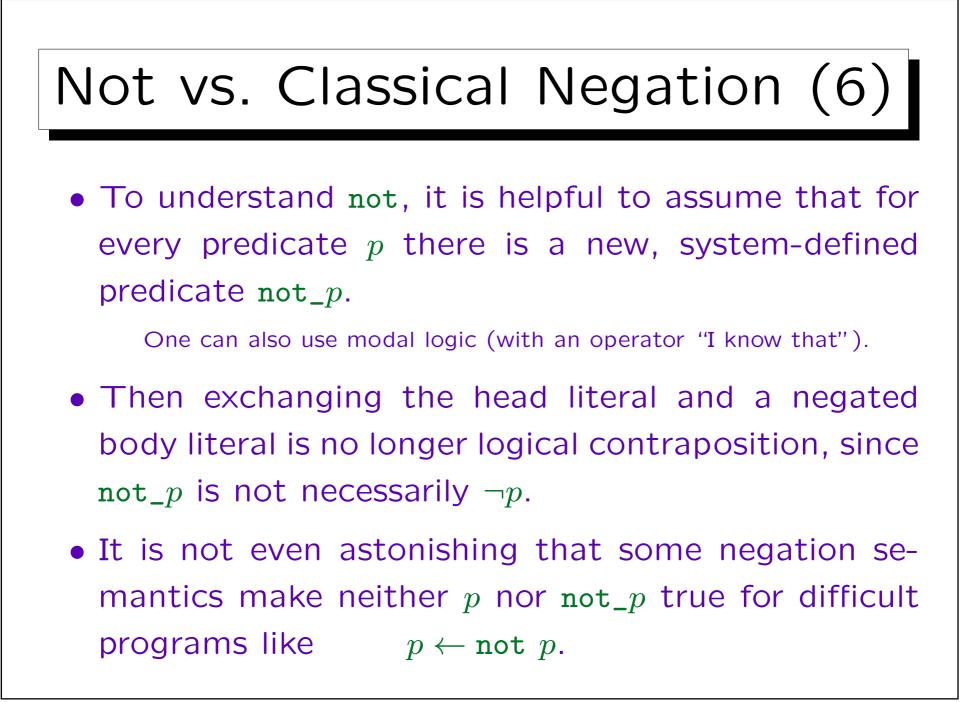
Not vs. Classical Negation (2)
 Therefore, also this is not a logical consequence: available('Lloyd', 'Found. of Logic Progr.')
 This is a difference to Horn clause Datalog: There, all answer-facts in the minimal model are logical consequences of the given program.
 Negative facts must be assumed "by default" if the corresponding positive fact is not provable.
 In order to prove not A, Prolog first tries to prove A. If this fails, not A is considered "proven": "Negation as (Finite) Failure" / "Default Negation".



gation in logic programming.



Not vs. Classical Negation (5)
 In logic programming (with not instead of ¬), the two formulas have a completely different semantics: borrowed(BID) ← book(BID, Author, Title) ∧ not available(Author, Title).
 Because no available-facts can be derived, Prolog now concludes that also 'Llo87' is borrowed.
 In logic programming, rules can be used in only one direction. The distinction between head and body is important.
The contraposition of a rule is not used.



Why not Classical Logic?

NOT is useful/necessary:

- Already the specification of finite relations (as in relational databases) is quite complicated in first order logic.
- The transitive closure cannot be defined in first order logic.

The well-known rules entail what must be true. But one cannot make sure in classical logic that every other fact is false, i.e. one cannot give an \leftrightarrow -definition of path that works for any given edge-relation.



- 1. Motivation, Differences to classical logic)
- 2. Syntax, Supported Models
- 3. Stratification, Perfect Model
- 4. Conditional Facts, Well-Founded Model
- 5. Stable Models

Stefan Brass: Logic Programming and Deductive Databases



- Now two types of body literals are allowed:
 - ♦ Postive body literals (atomic formulas, as usual): $p(t_1,...,t_n)$
 - Negative body literals (default negation of an atomic formula):

not
$$p(t_1,\ldots,t_n)$$

• The default negation operator not cannot be used in the head of a rule.

This corresponds to the above view that " not_p " is a system defined predicate. One cannot introduce rules that define this predicate.





 SLDNF-Resolution (SLD-resolution with negation as failure) is a generalization of SLD-resolution to programs with negative body literals.

Some authors think that it is more precise to say "finite failure".

• As in SLD-resolution, a tree is constructed, where the nodes are marked with goals (conjunctions of positive and negative literals).

Seen as a refutation proof, one can also view the goals as disjunction of the opposite literals.

• If the selected literal is a positive literal, child nodes are constructed as in SLD-resolution.



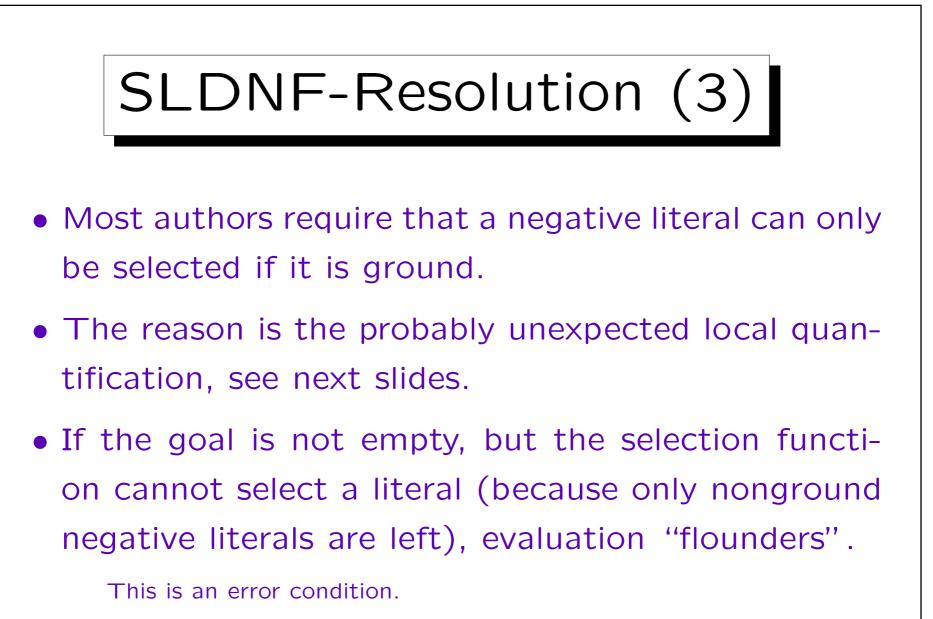


 If the selected literal is a negative literal, SLDNFresolution calls itself recursively with the corresponding positive literal as query.

I.e. a new tree is constructed, the root marked the the positive literal.

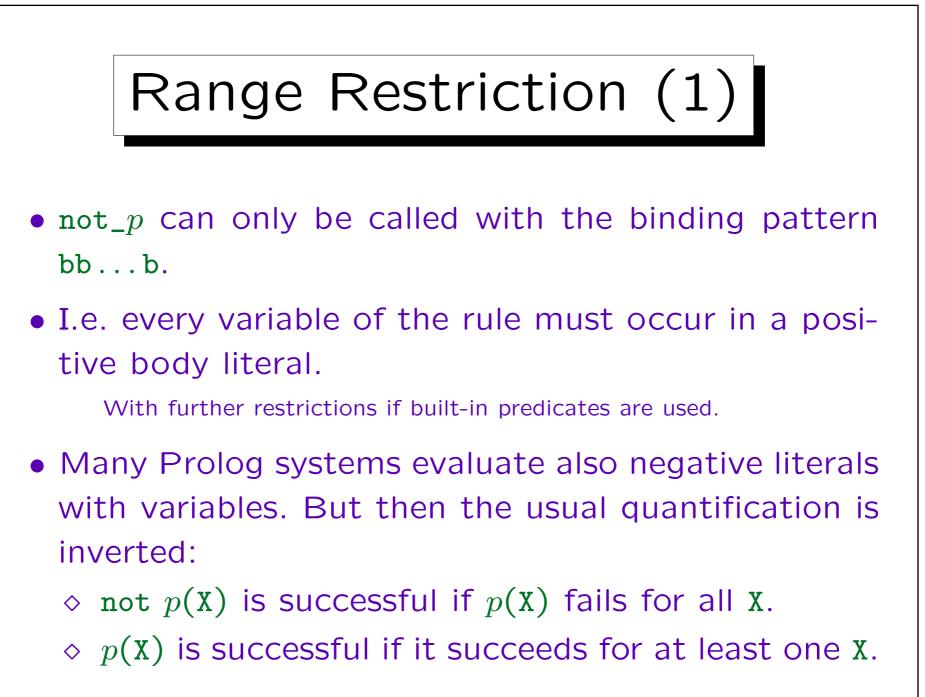
- If this tree is finite and contains no success node (empty goal), the negative literal is considered proven, and the calling node gets a single child node with the negative literal removed.
- ◊ If the tree contains a success node, the calling node is a failure node (without child nodes).

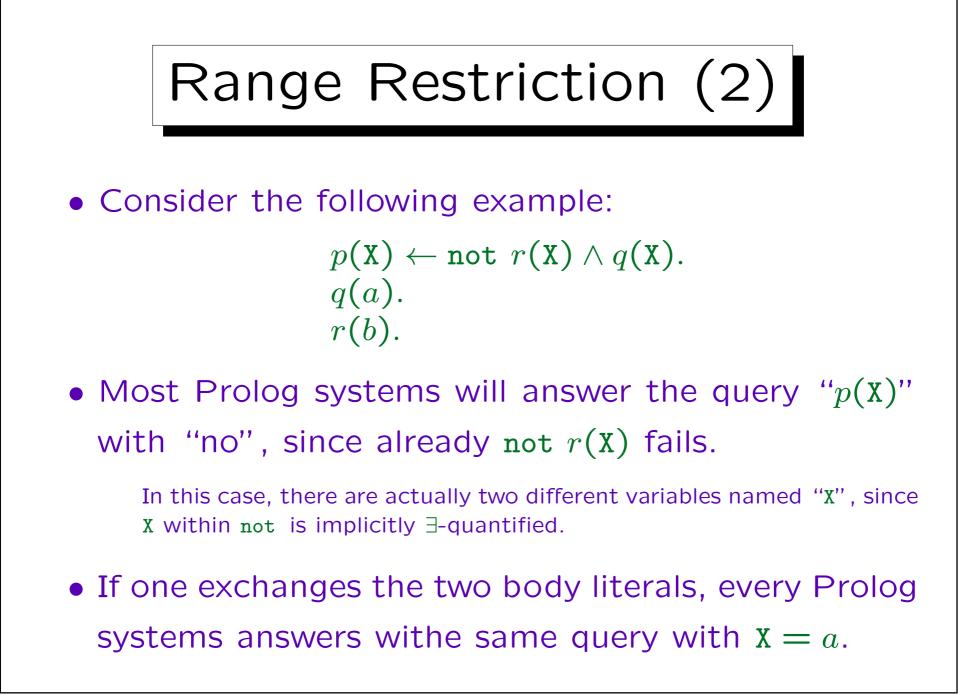




• Most Prolog systems do not obey this restriction.

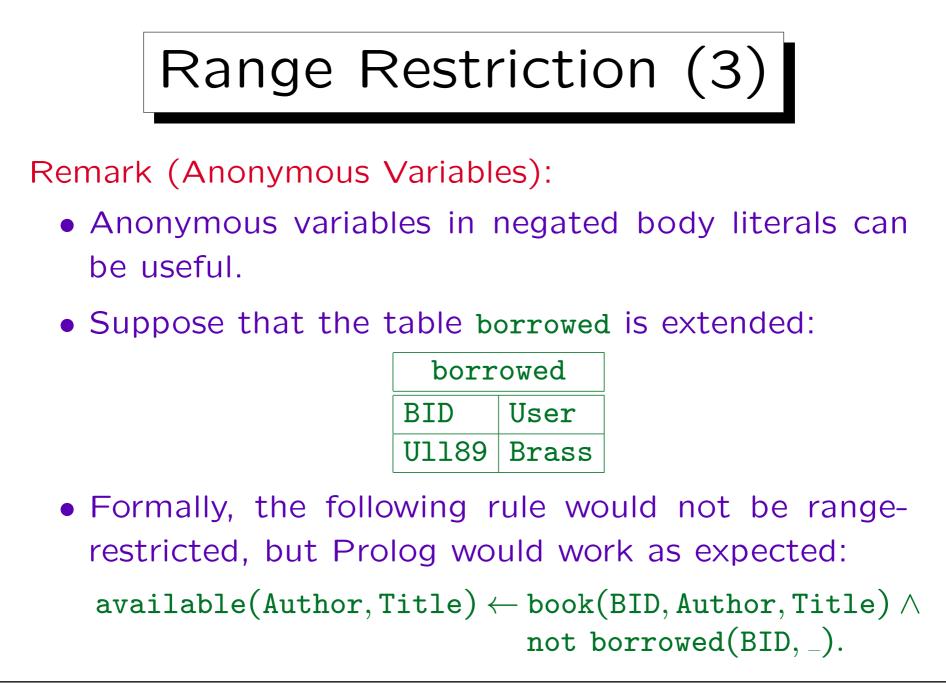


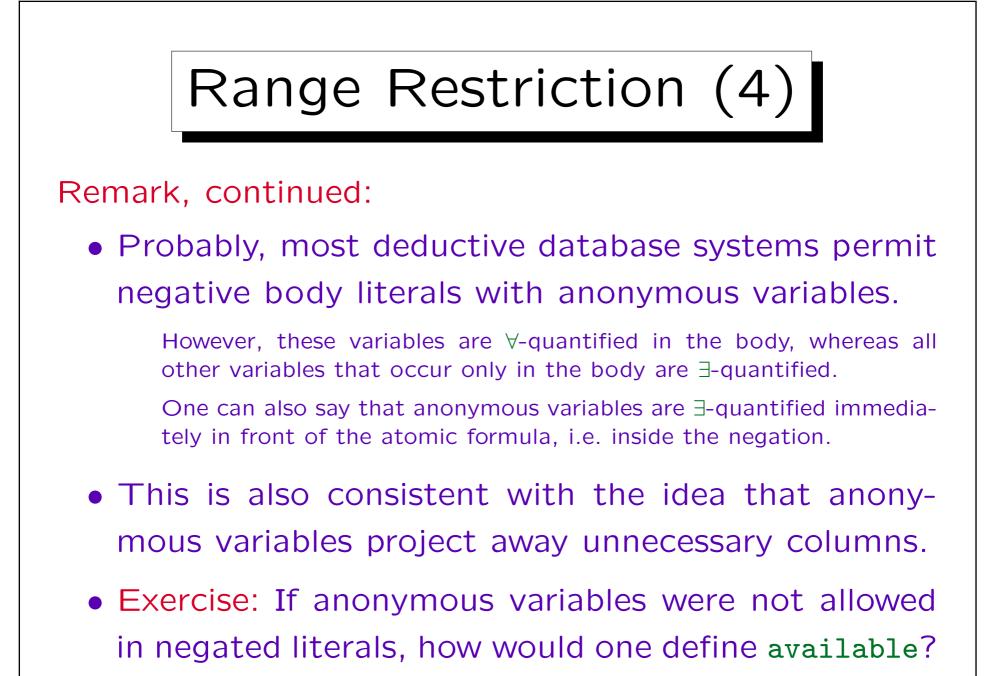




Stefan Brass: Logic Programming and Deductive Databases Univ. Halle, 2017



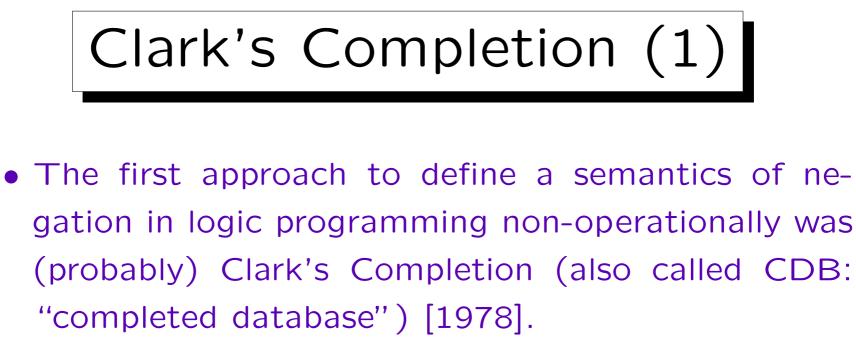






- Let the following EDB-relations be given:
 - ◊ lecture_hall(RoomNo, Capacity).
 - ◊ reservation(RoomNo, Day, From, To, Course).
- Which lecture halls are free on tuesdays, $8^{30}-10^{00}$?
- What is the largest capacity of a lecture hall?
- Is there a time at which all lecture halls are used?

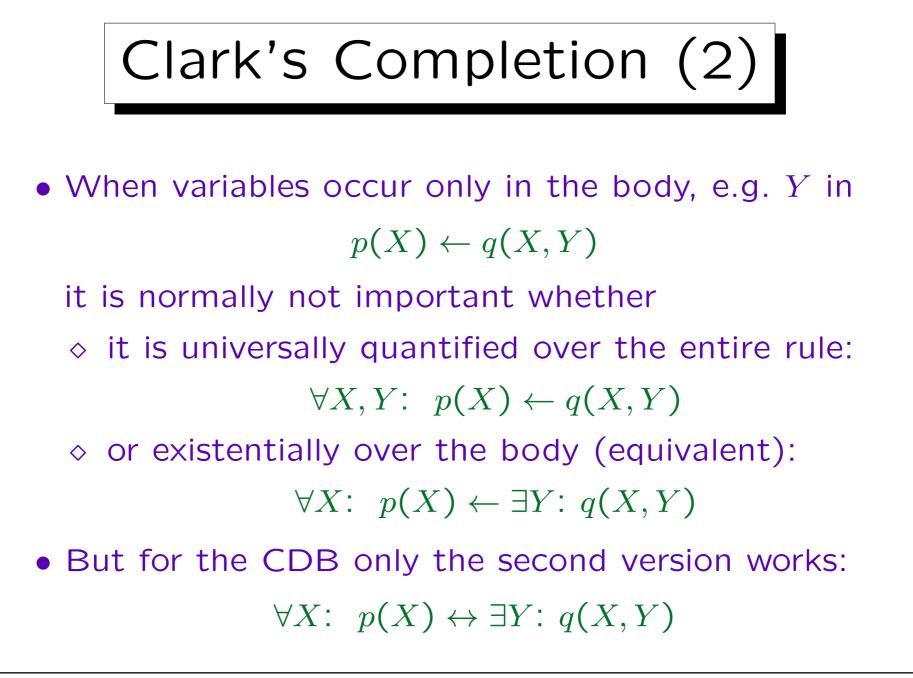
If there is such a time at all, also one of the times in From satisfies this condition (Proof: Go back from the given time, when all lecture halls are used, to the nearest start of a reservation.). Thus, it is not necessary to check all possible times.

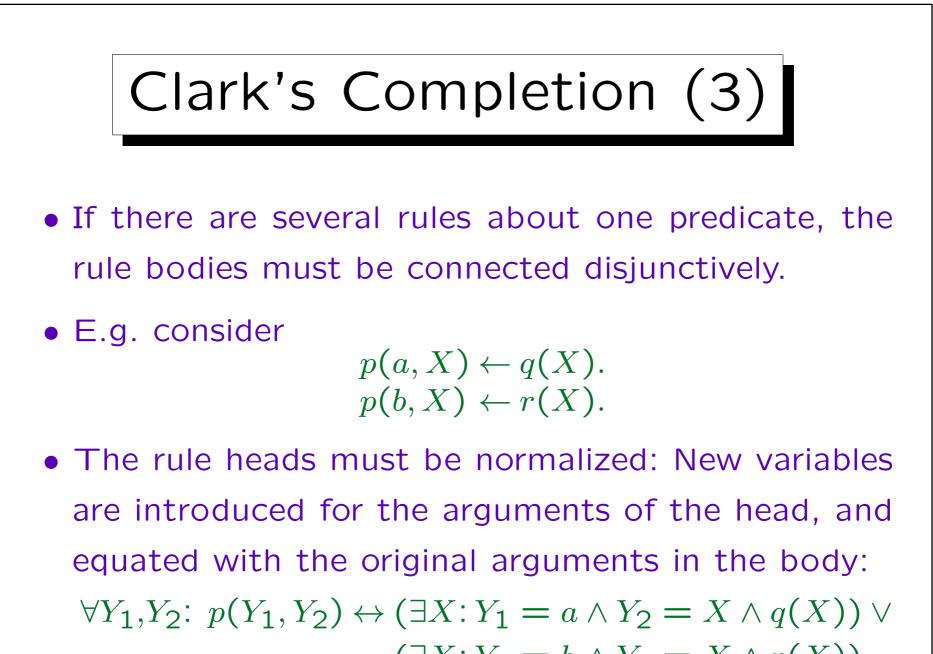


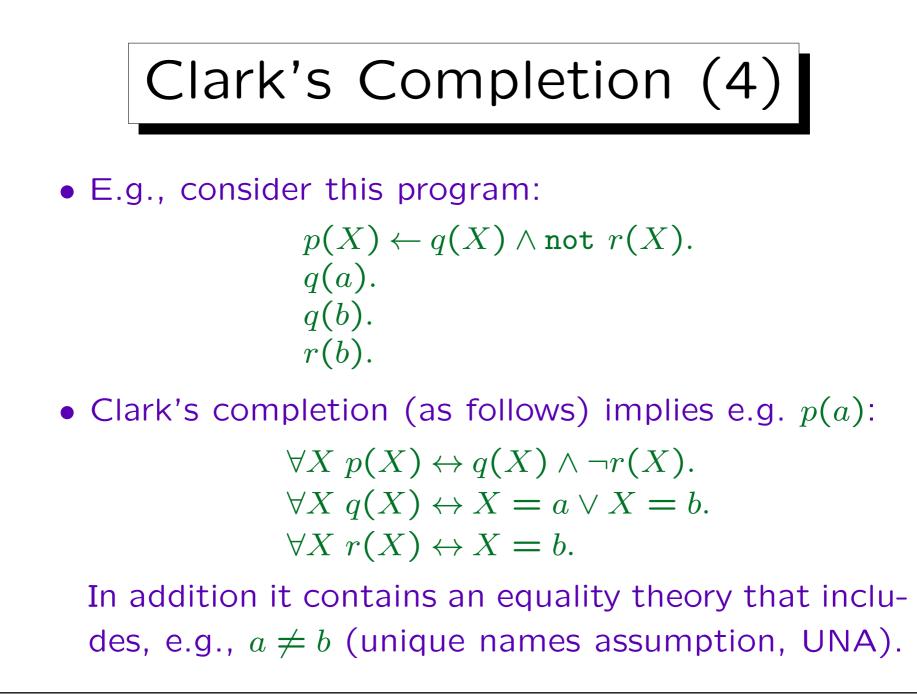
- Basically, the idea was to turn " \leftarrow " into " \leftrightarrow .
- E.g., if the only rule about p is $p(X) \leftarrow q(X) \wedge r(X)$

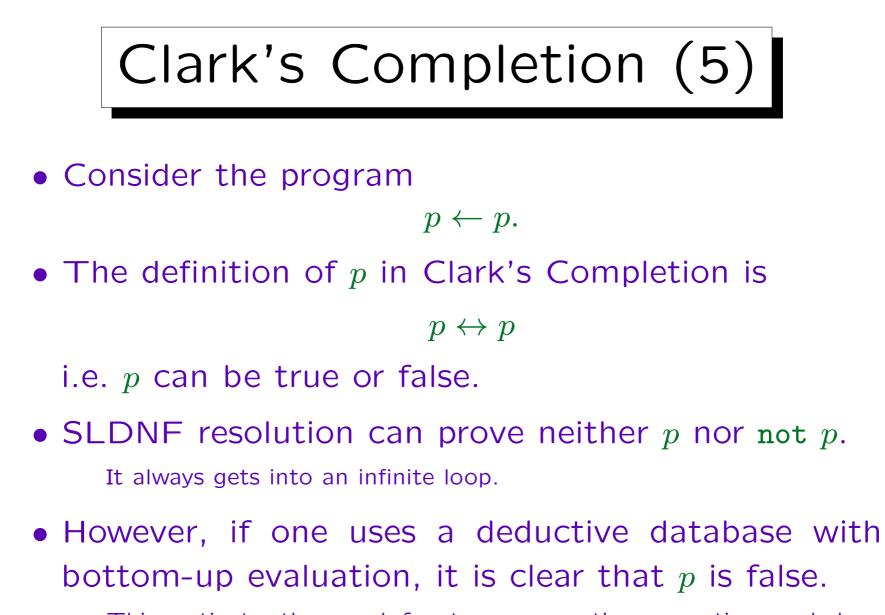
the definition of p in the CDB is (equivalent to)

 $\forall X: p(X) \leftrightarrow q(X) \wedge r(X)$

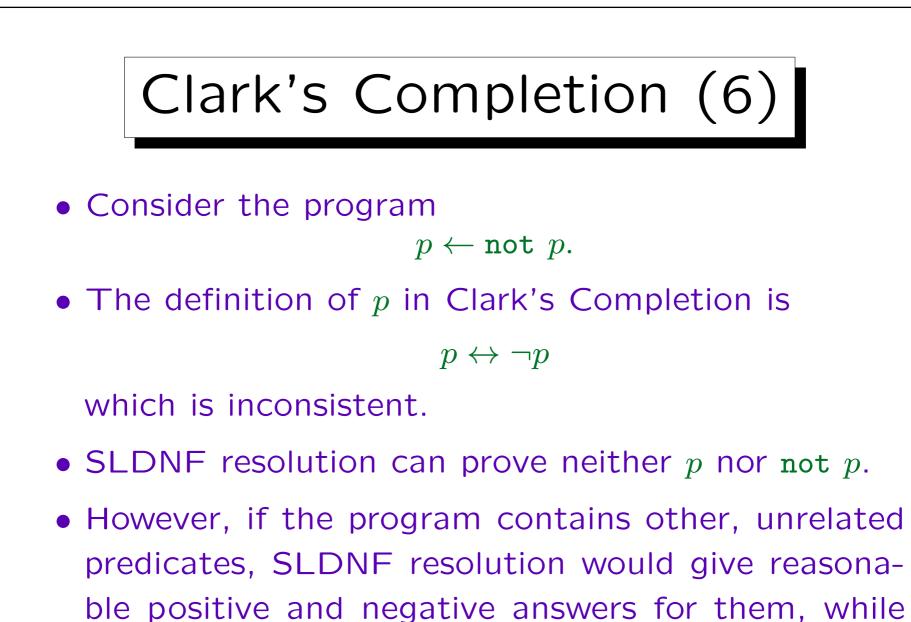




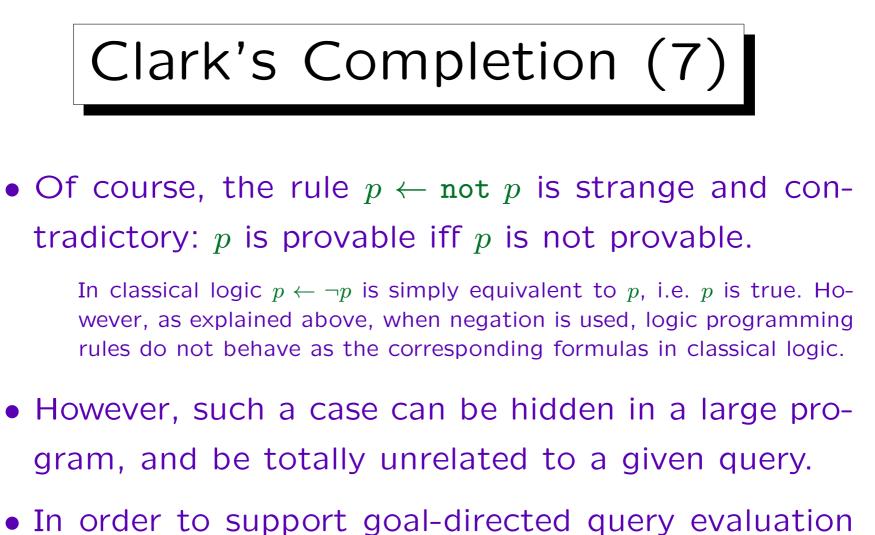




This motivates the search for stronger negation semantics, see below.



Clark's completion implies everything.



procedures, such cases must be excluded or the semantics must "localize" the consistency problem.





- 1. Motivation, Differences to classical logic)
- 2. Syntax, Supported Models

3. Stratification, Perfect Model

- 4. Conditional Facts, Well-Founded Model
- 5. Stable Models

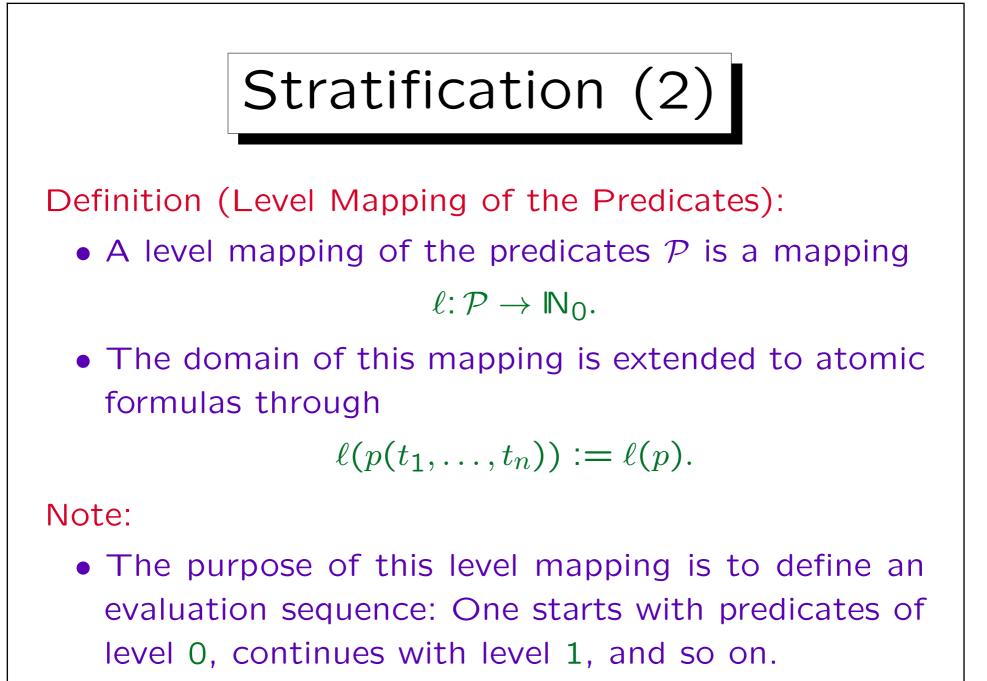
Stefan Brass: Logic Programming and Deductive Databases



- In order to avoid the $p \leftarrow not p$ problem, the class of stratified programs is introduced.
- Note that negative body literals not $p(t_1, \ldots, t_n)$ can be easily evaluated if the complete extension of pwas already computed previously.

Variables among the arguments are already bound to a concrete value because of the range restriction. Thus, one only has to check whether the argument tuple is contained in the extension of p.

 This means that p must not depend on a predicate that depends on not p.
 In short: Recursion through negation is excluded.



Stratification (3)

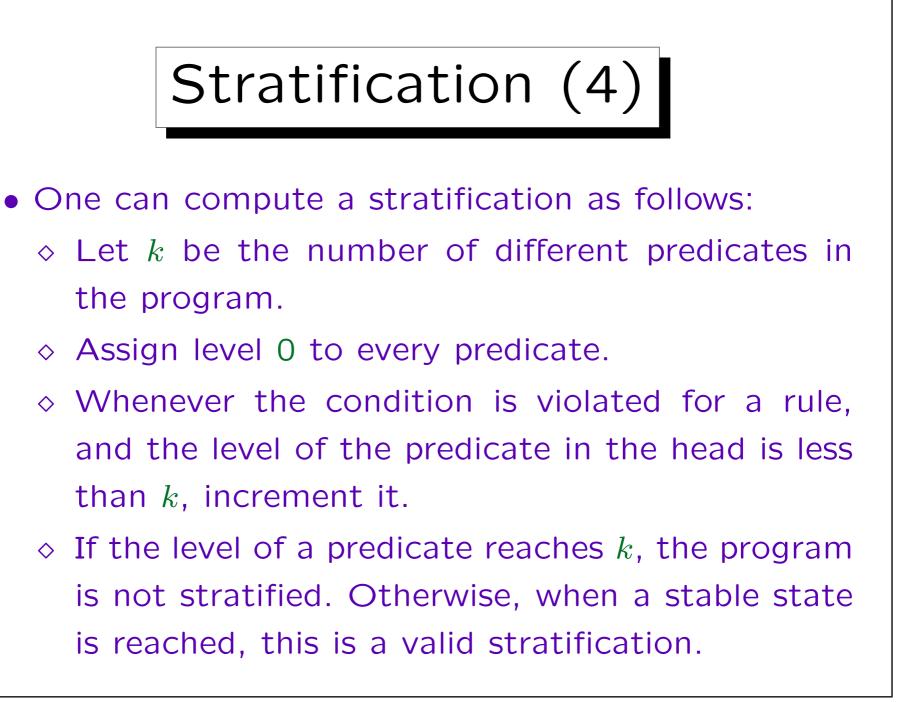
Definition (Stratified Program):

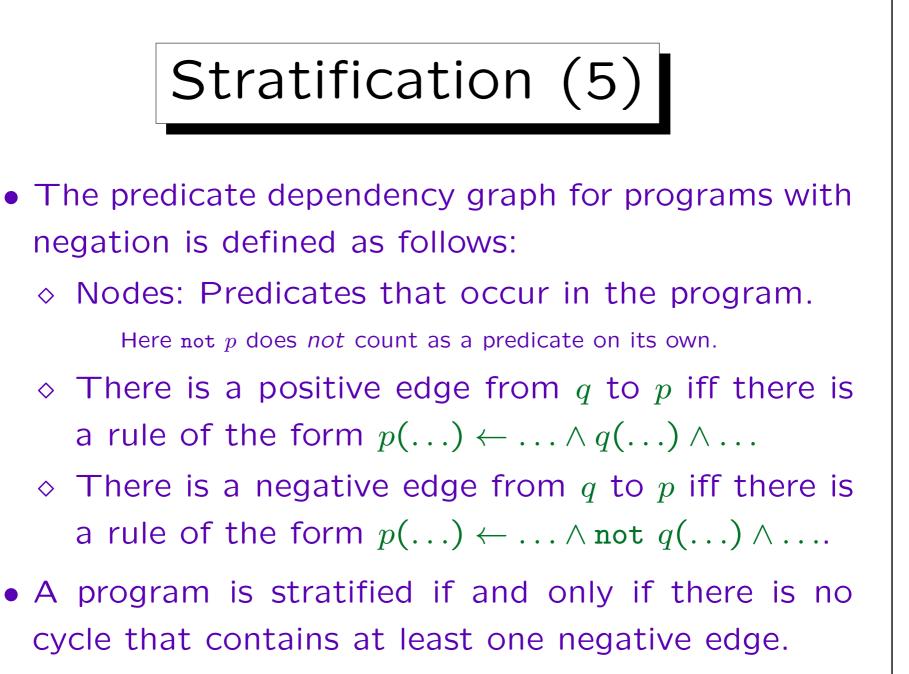
• A program P is stratified if and only if there is a level mapping ℓ such that for each rule

 $A \leftarrow B_1 \wedge \cdots \wedge B_m \wedge \operatorname{not} C_1 \wedge \cdots \wedge \operatorname{not} C_n$

the following holds:

- $\diamond \ \ell(B_i) \leq \ell(A) \text{ for } i = 1, \dots, m, \text{ and }$
- $\diamond \ \ell(C_j) < \ell(A) \text{ for } i = 1, \dots, n.$
- Such a level mapping is called a stratification of P.







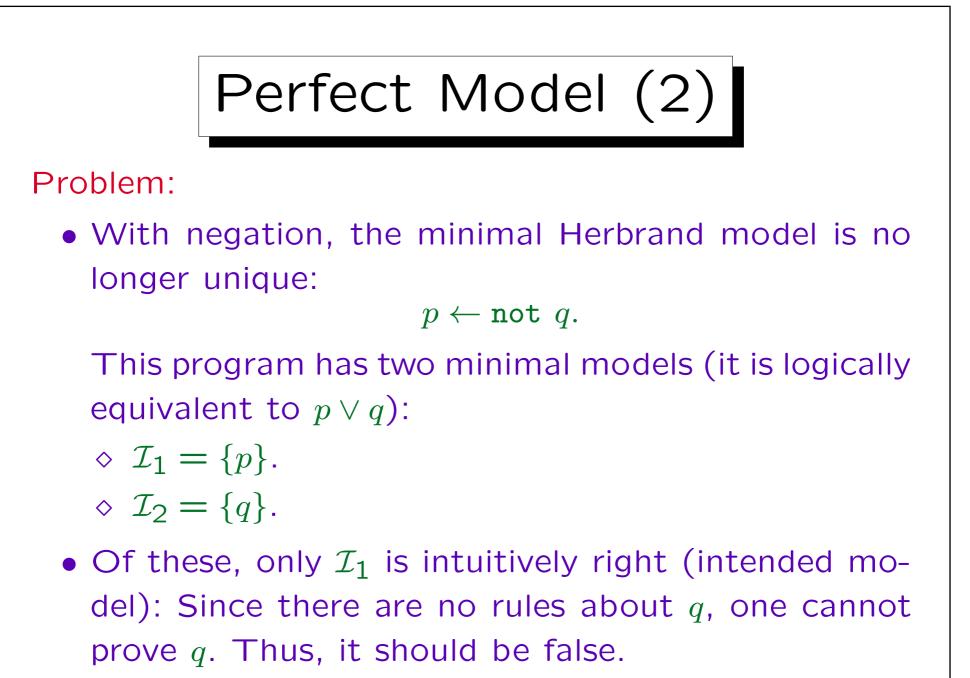
- For defining when an interpretation is a model of a program one treats not like classical negation ¬.
- Thus, an interpretation ${\mathcal I}$ is a model of a program P iff for every rule

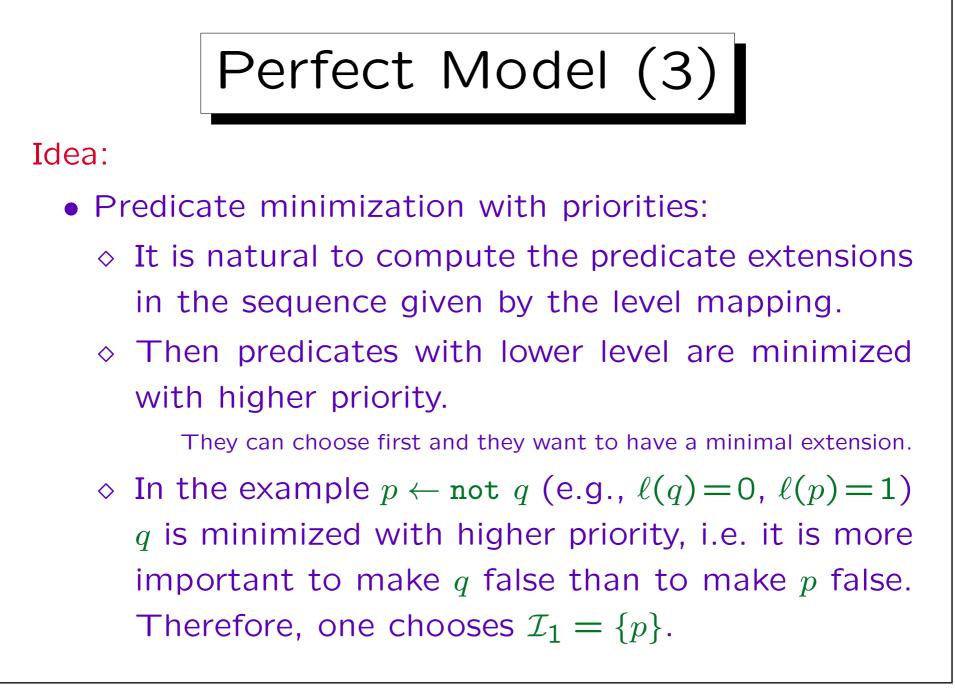
 $A \leftarrow B_1 \land \cdots \land B_m \land \texttt{not} \ C_1 \land \cdots \land \texttt{not} \ C_n$

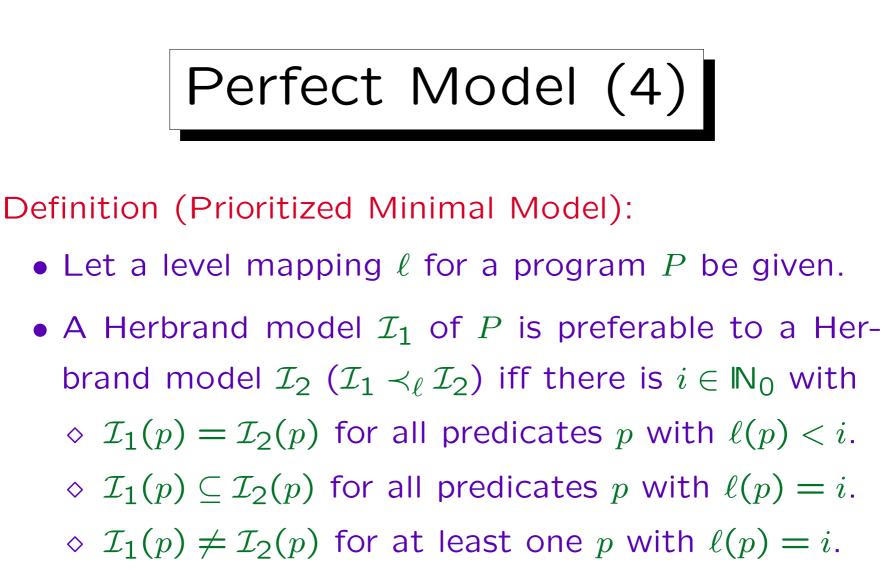
and every variable assignment \mathcal{A} the following holds:

♦ If $(\mathcal{I}, \mathcal{A}) \models B_i$ for i = 1, ..., m and $(\mathcal{I}, \mathcal{A}) \not\models C_j$ for j = 1, ..., n, then $(\mathcal{I}, \mathcal{A}) \models A$.

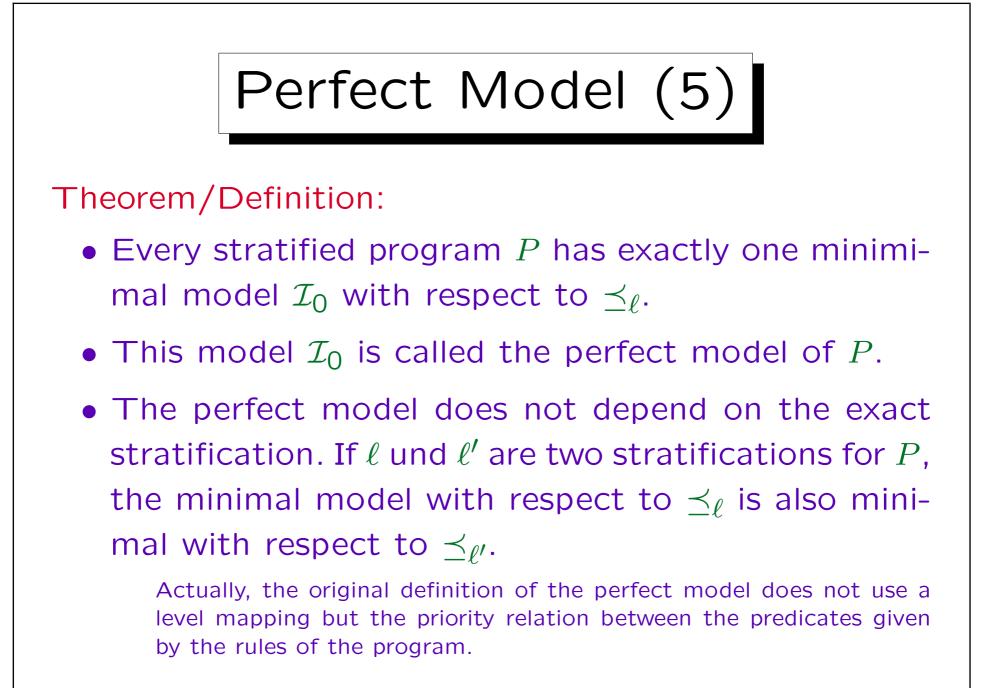
If one identifies again a Herbrand model \mathcal{I} with its set of true facts, a ground literal not $p(c_1, \ldots, c_n)$ is true in \mathcal{I} if and only if $p(c_1, \ldots, c_n) \notin \mathcal{I}$.

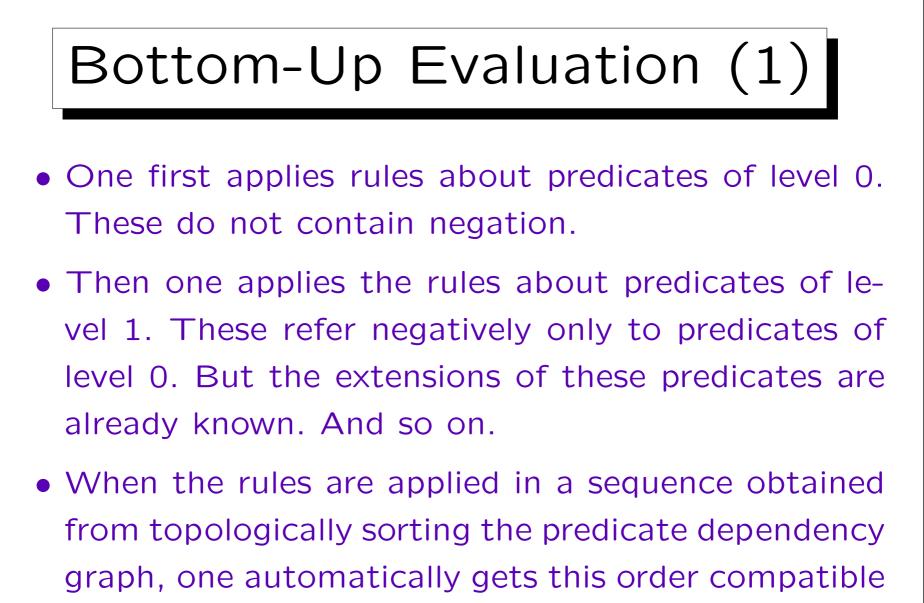




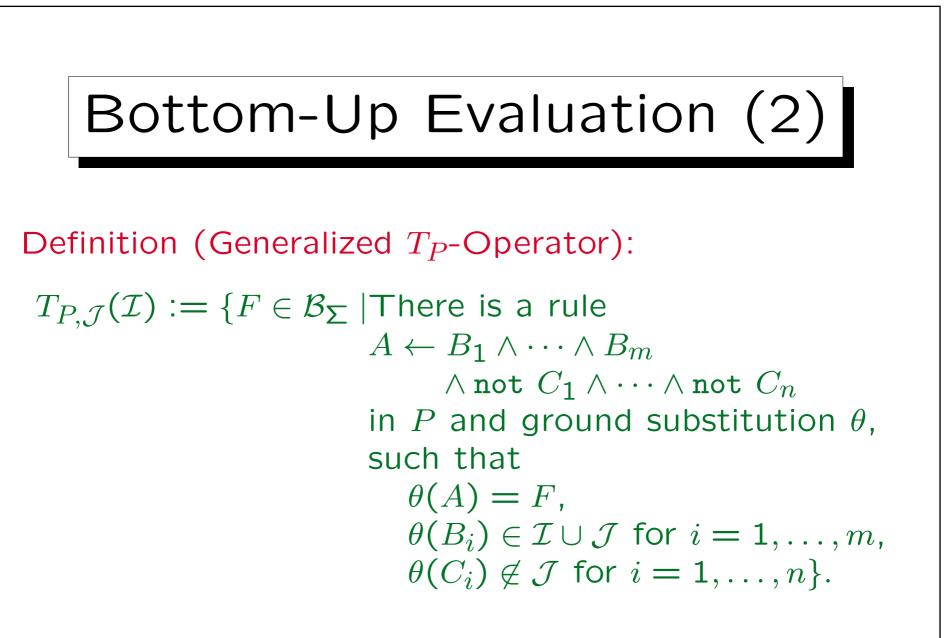


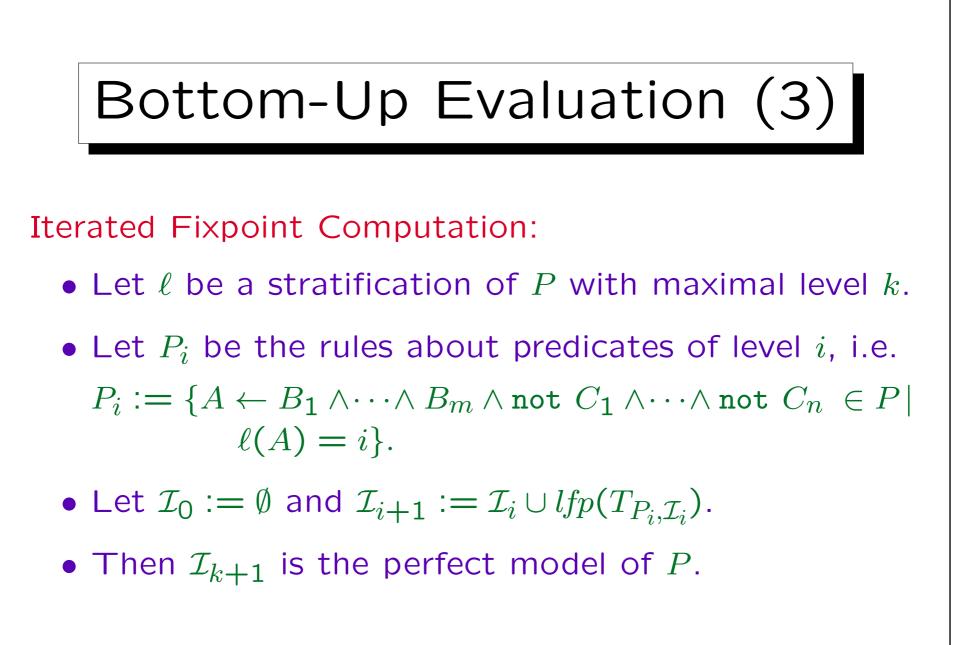
• $\mathcal{I}_1 \preceq_{\ell} \mathcal{I}_2$ iff $\mathcal{I}_1 \prec_{\ell} \mathcal{I}_2$ or $\mathcal{I}_1 = \mathcal{I}_2$.





with the predicate levels.





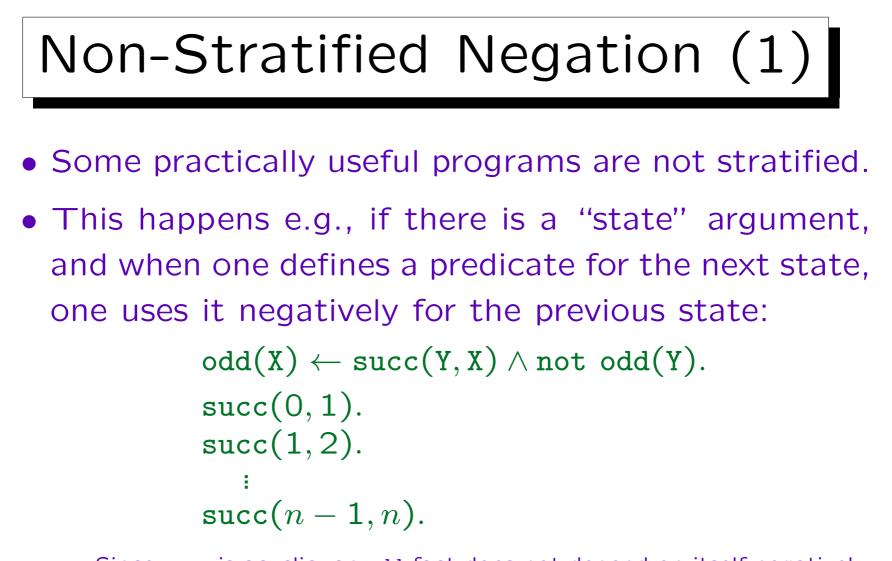




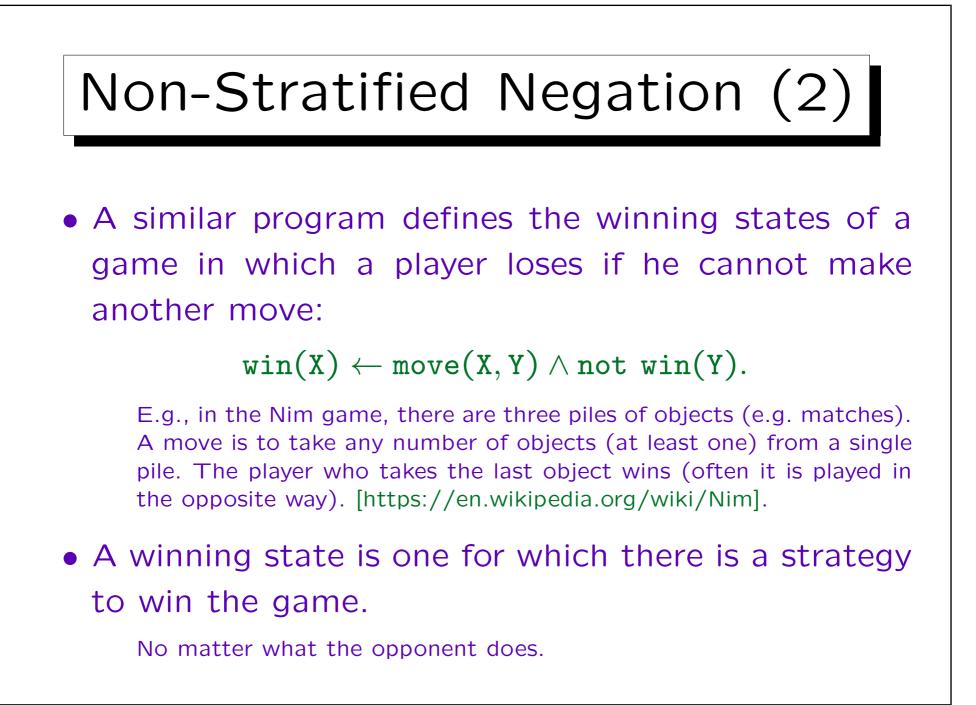
- 1. Motivation, Differences to classical logic)
- 2. Syntax, Supported Models
- 3. Stratification, Perfect Model
- 4. Conditional Facts, Well-Founded Model
- 5. Stable Models

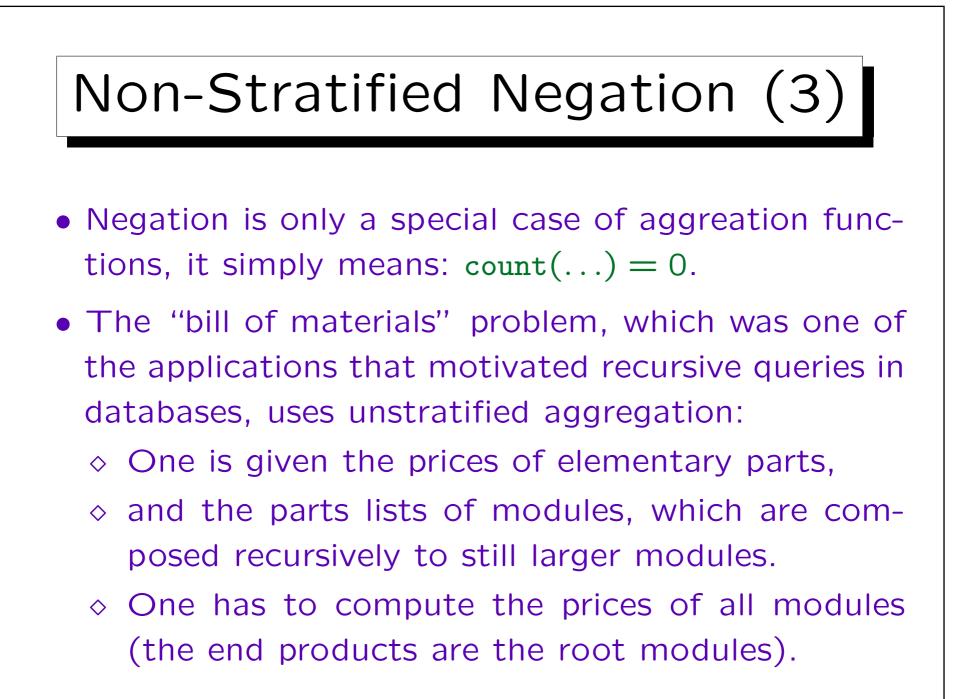
8-45

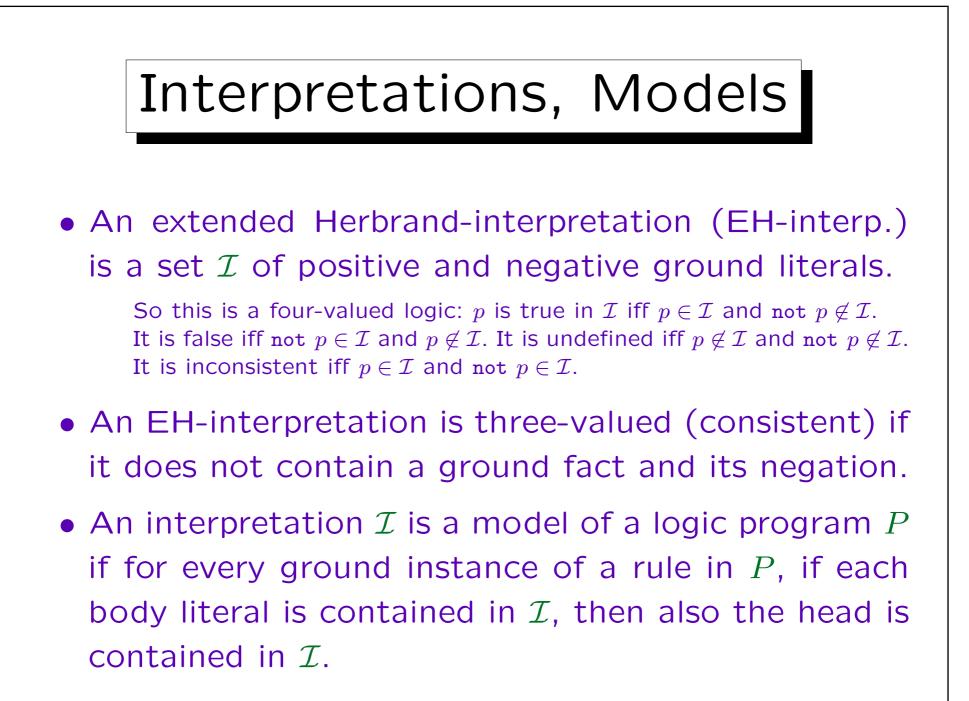
Univ. Halle, 2017

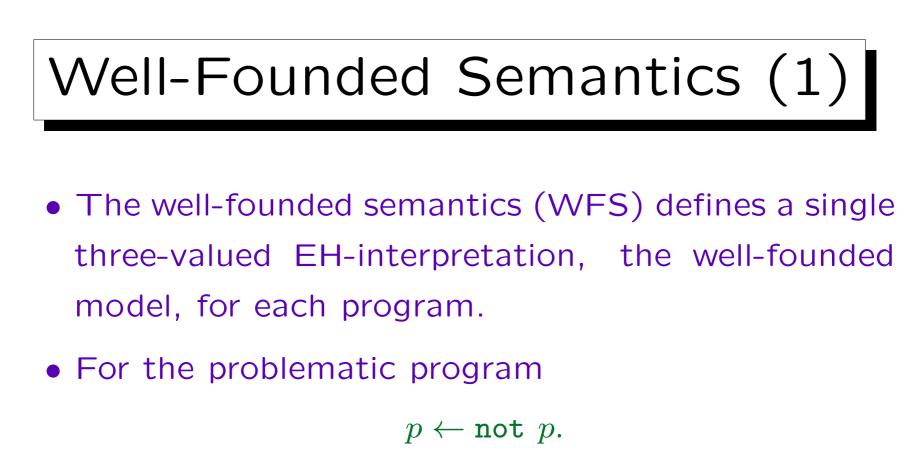


Since succ is acyclic, an odd-fact does not depend on itself negatively. But if one looks only at the predicates, there is the negative cycle. It depends on the data whether a program is "dynamically stratified".





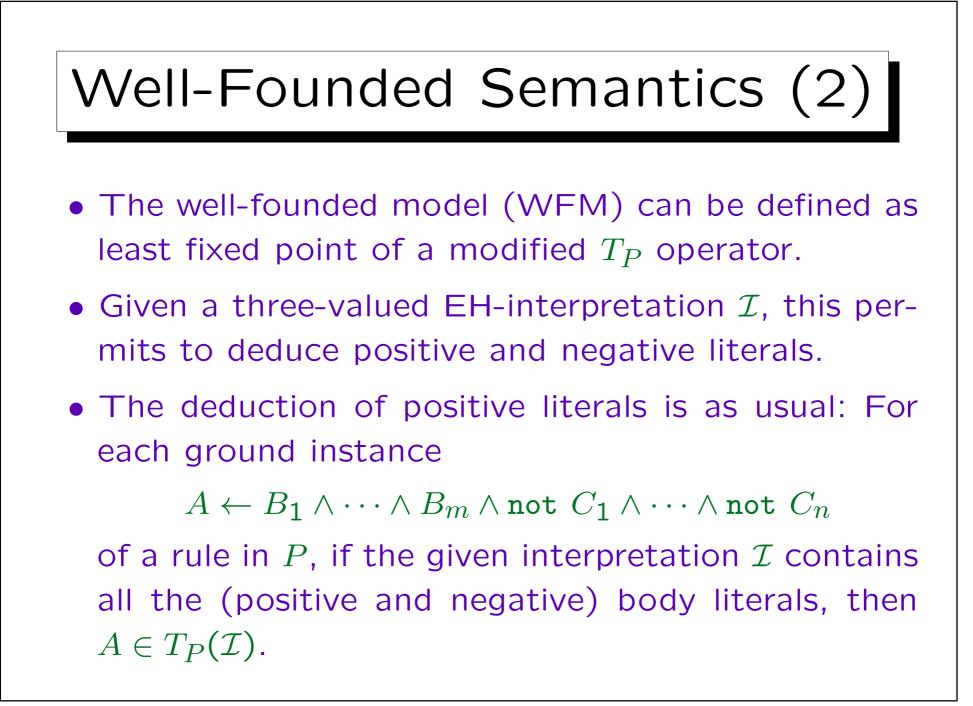


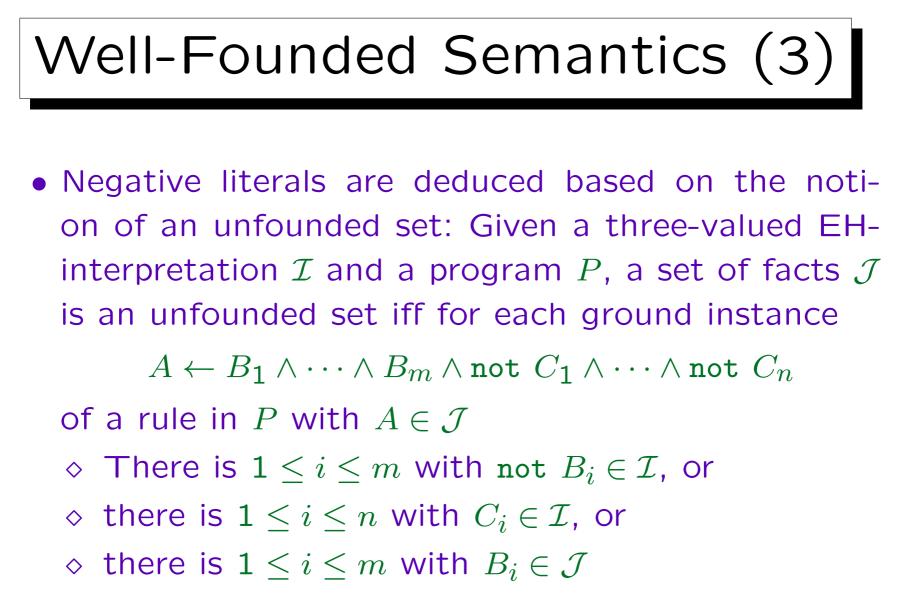


the well-founded model contains neither p nor not p.

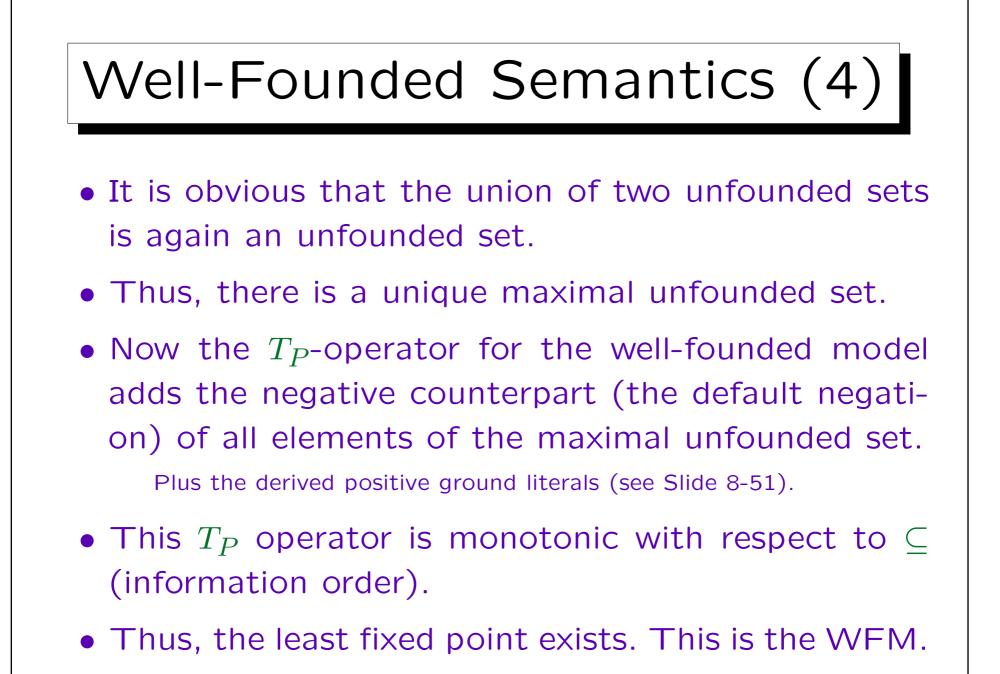
I.e. p has the third truth value "undefined". One can view this as a kind of error value.

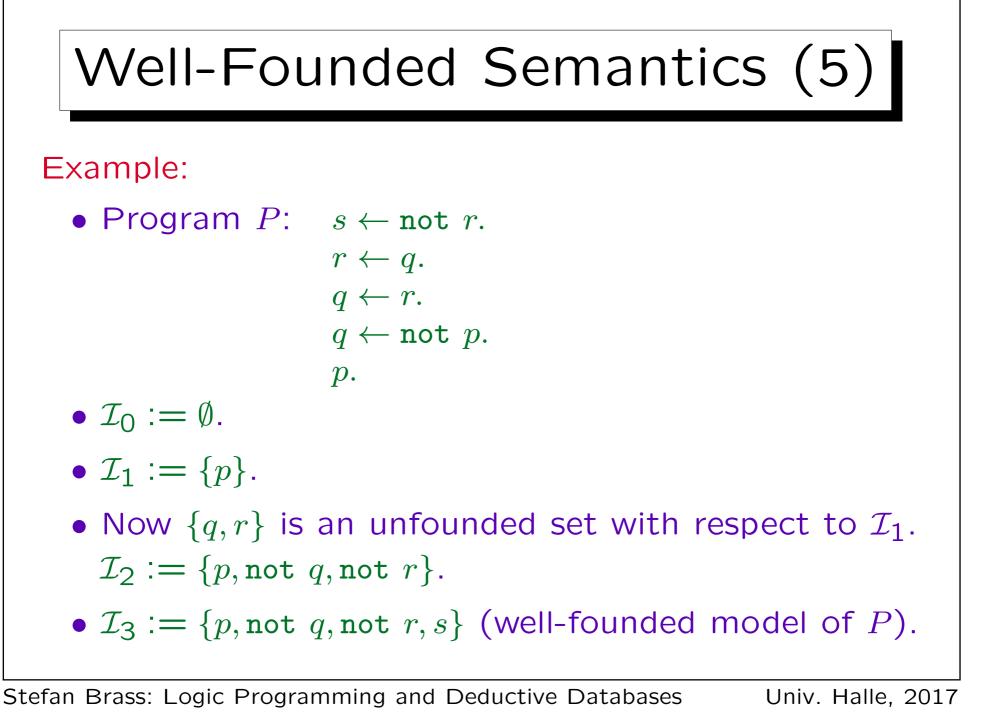
Stefan Brass: Logic Programming and Deductive Databases Univ. Halle, 2017

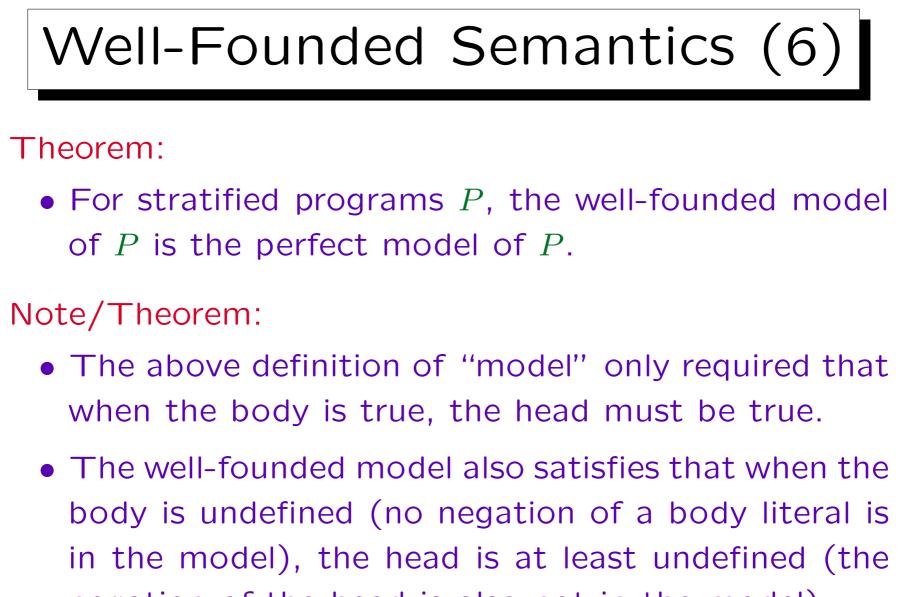




(called "witness of unusability of the rule").







negation of the head is also not in the model).

8. Negation

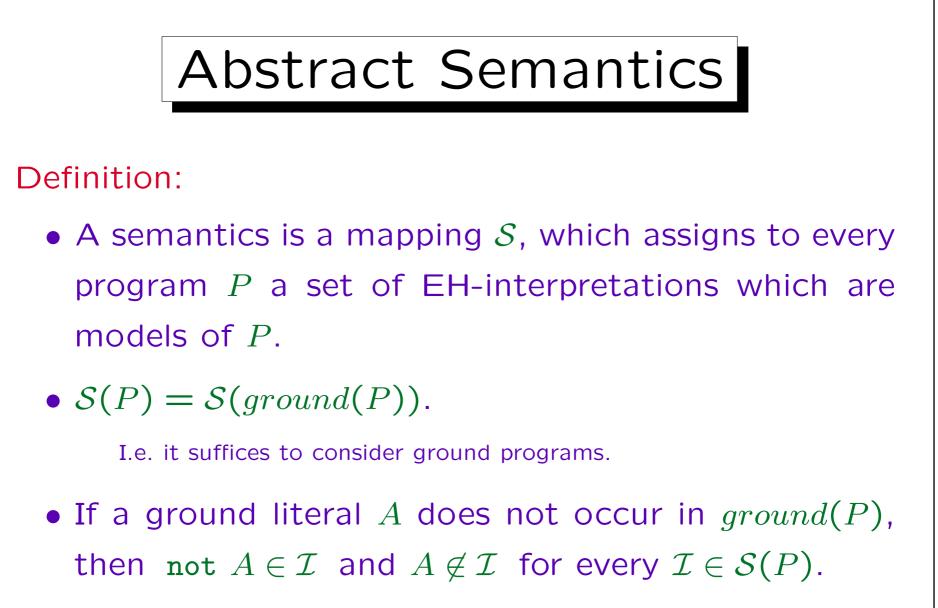


- For stratified programs, the semantics is clear, but stratified programs are not enough in practice.
- There are about 15 proposals for the semantics of nonmonotic negation. The WFS is one of them.

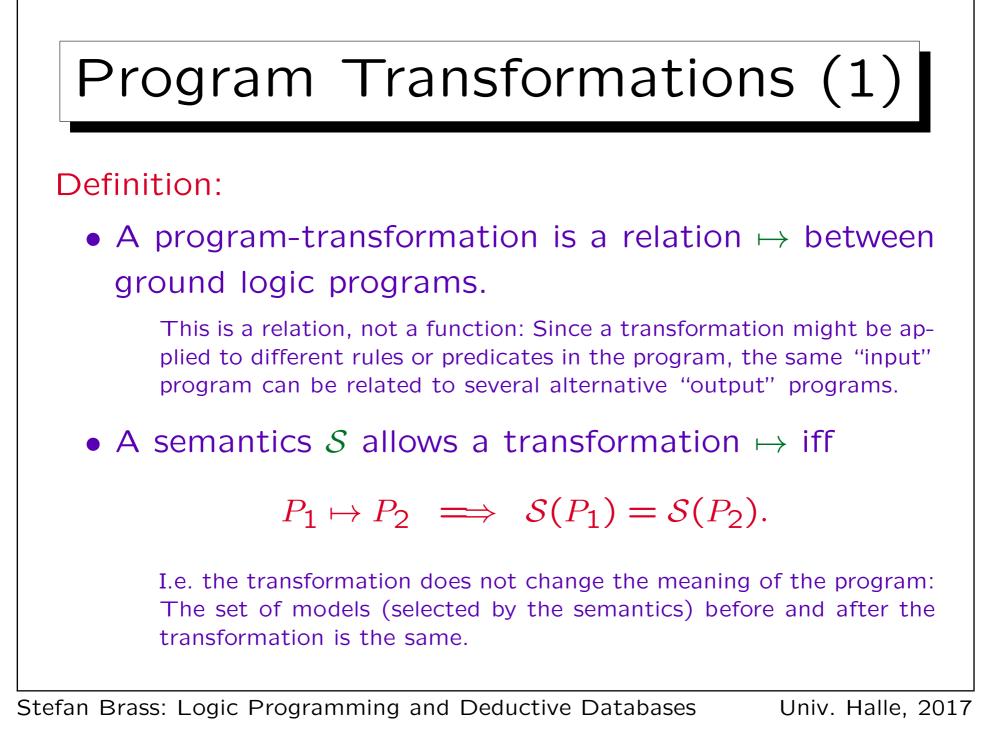
Actually, the well-founded semantics and the stable model semantics "survived", the others are only interesting for specialists. The stable model semantics lead to a new programming formalism, "answer set programming".

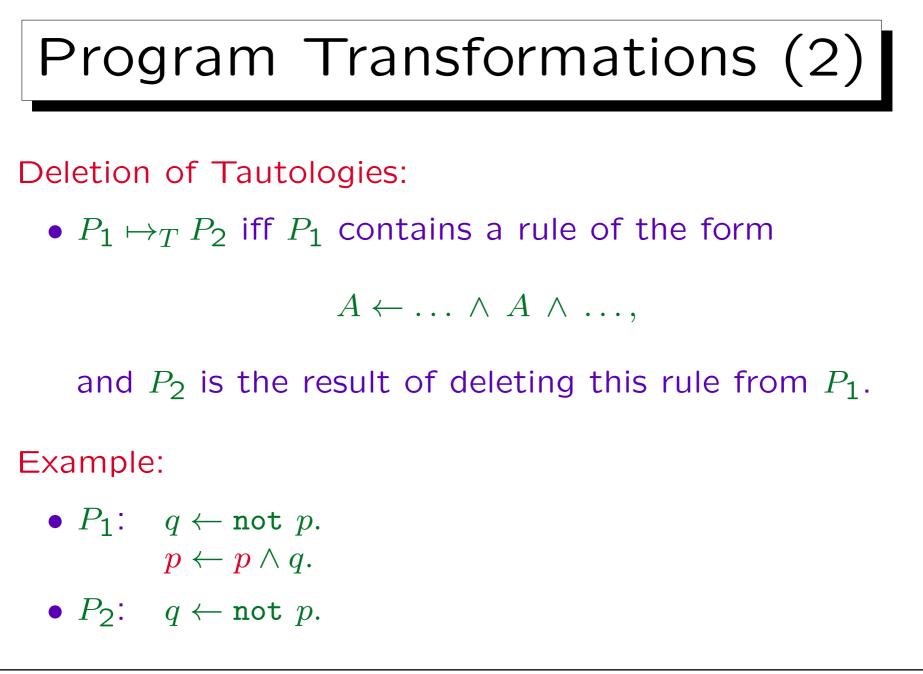
- Which one(s) are natural and free of surprises?
- Are there good semantics we do not know yet?

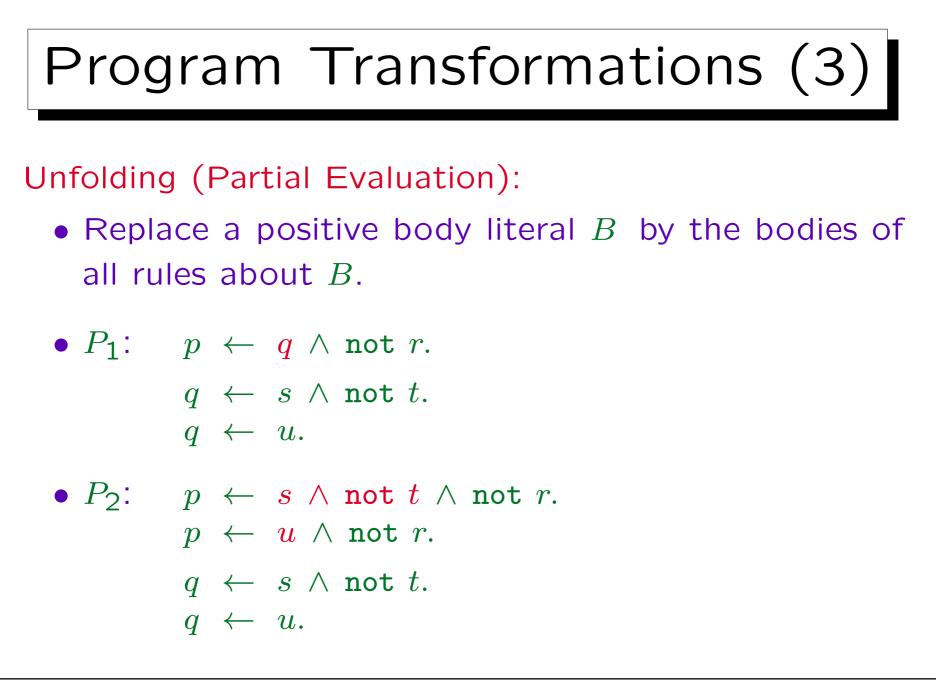
Stefan Brass: Logic Programming and Deductive Databases Univ. Halle, 2017

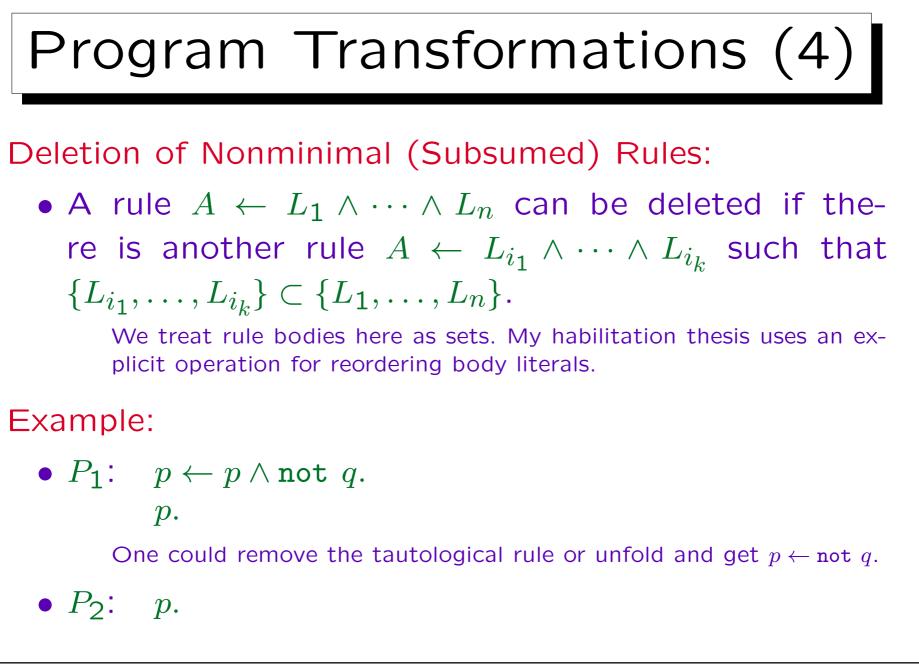


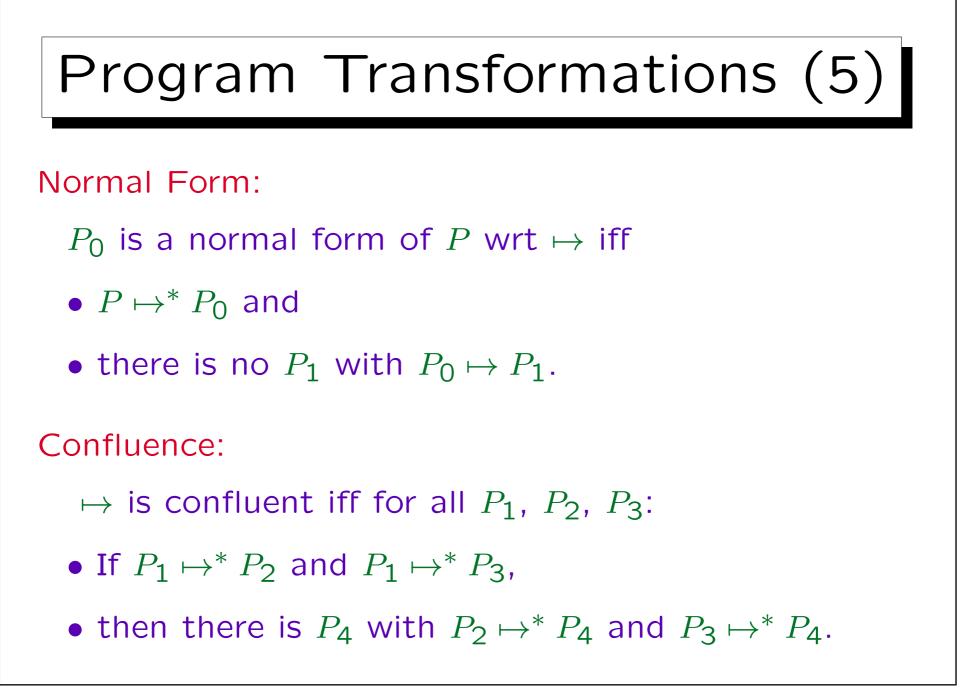
I.e. in this obvious case, A must be false.

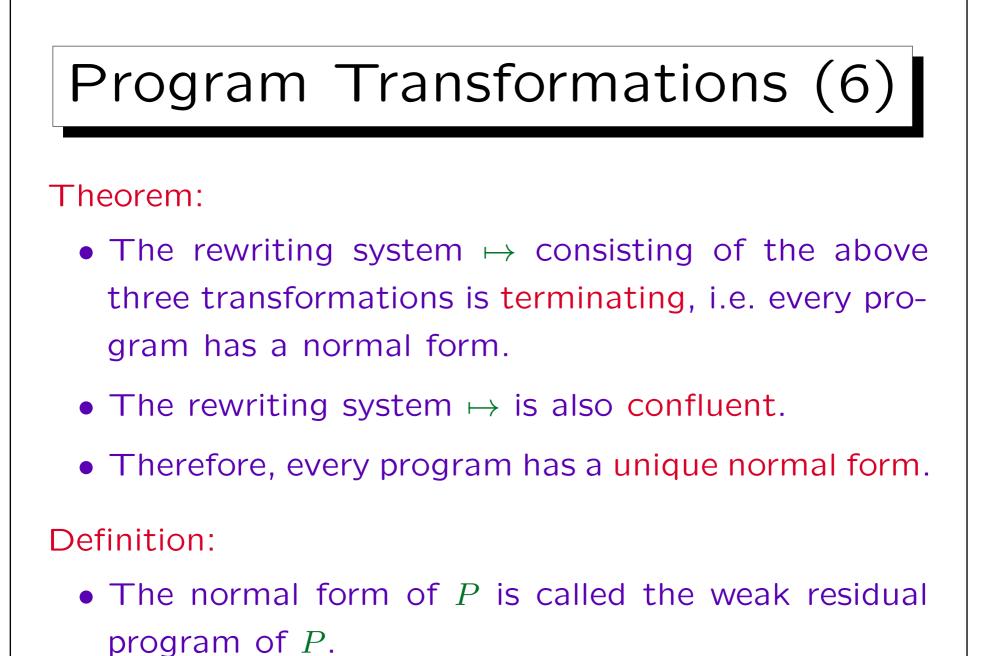




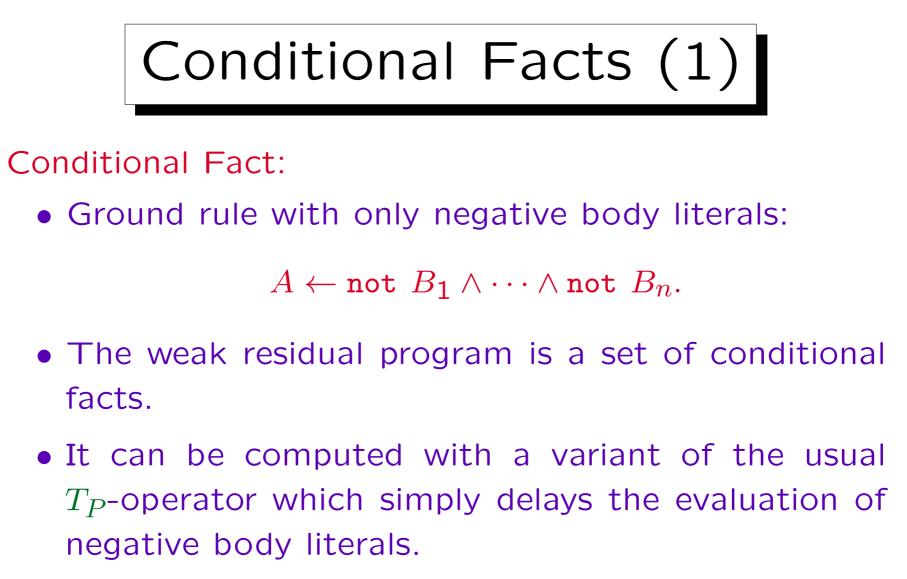




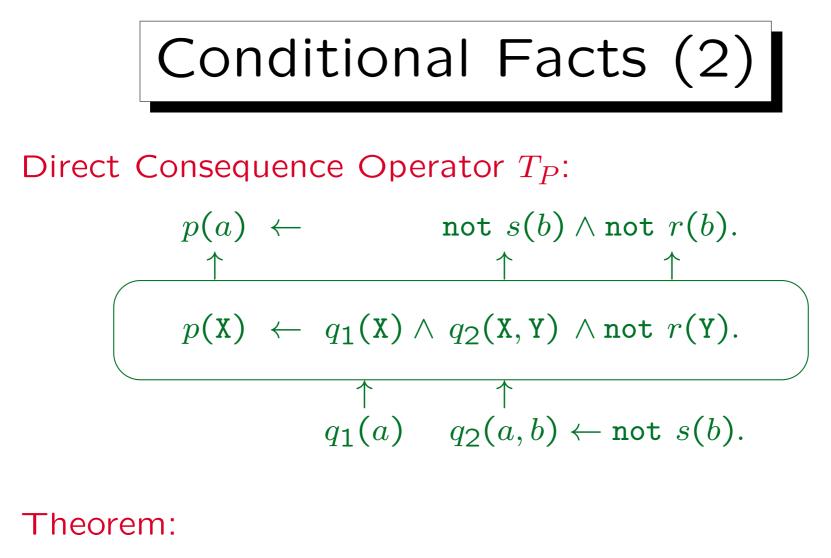




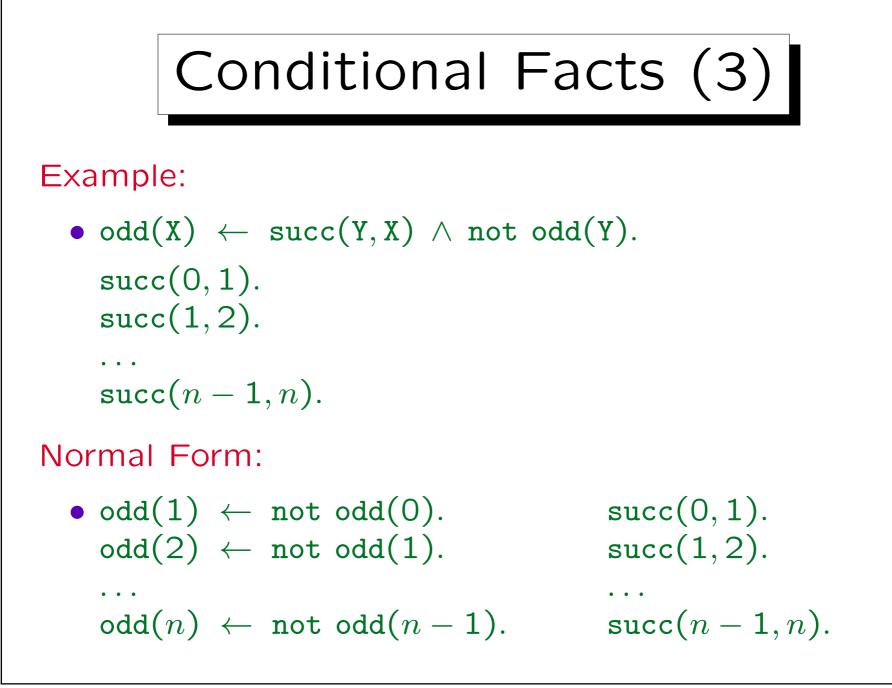


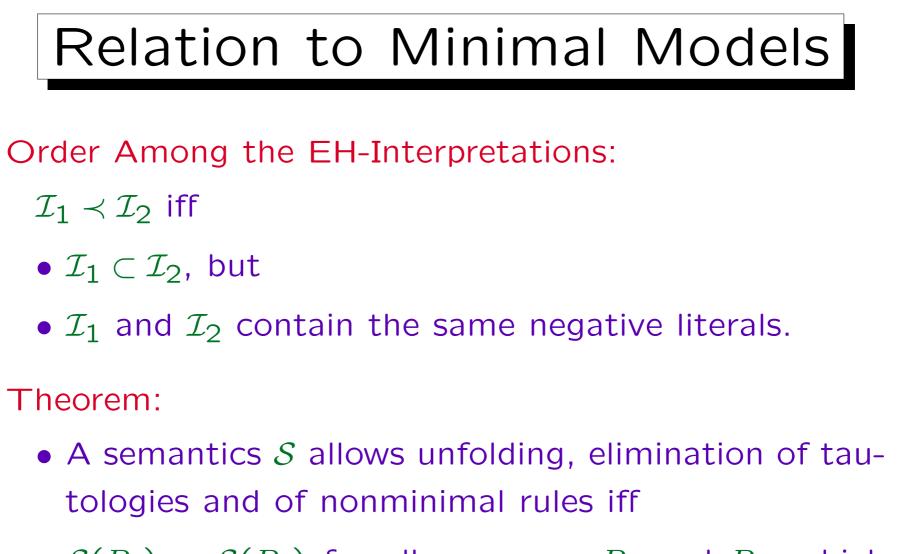


The range restriction ensures that all variables in the rule will be bound if we insert conditional facts for the positive body literals.



• $lfp(T_P)$ (without nonminimal cond. facts) is exactly the normal form of ground(P).





• $S(P_1) = S(P_2)$ for all programs P_1 and P_2 , which have the same set of \prec -minimal EH-models.



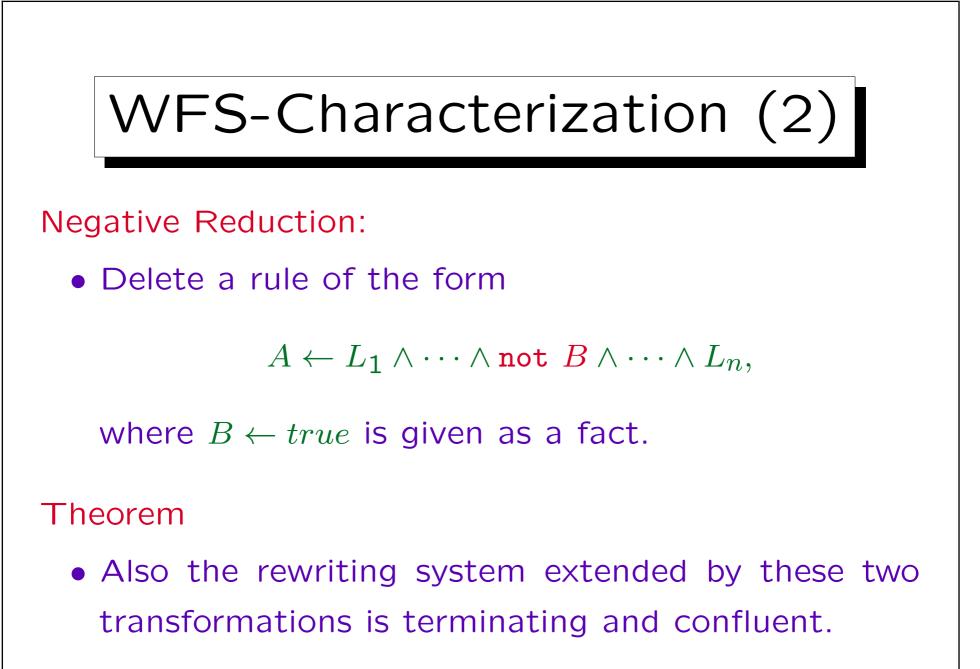
Positive Reduction:

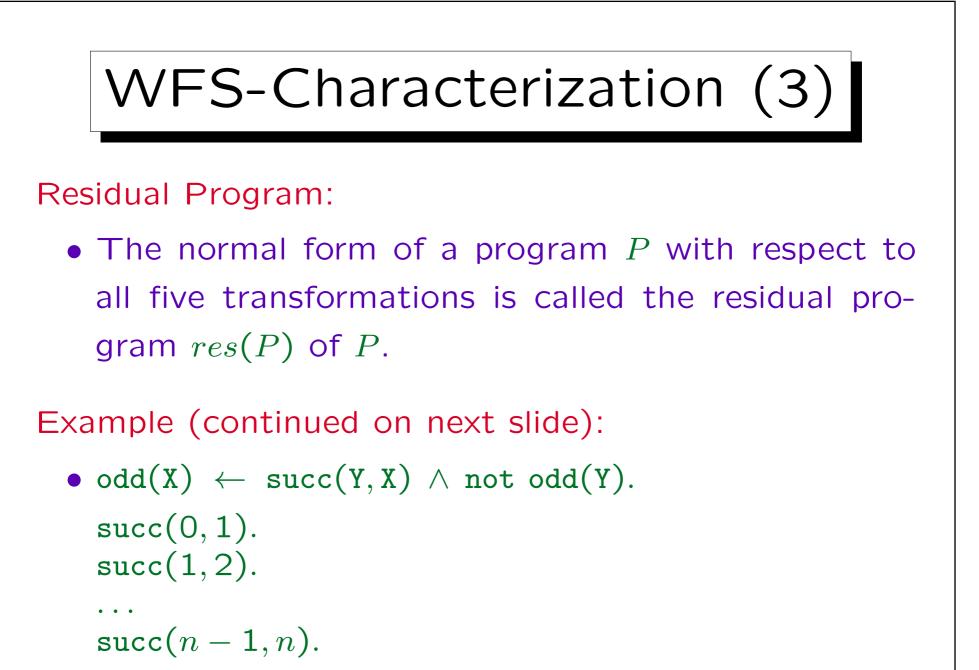
• Replace a rule of the form

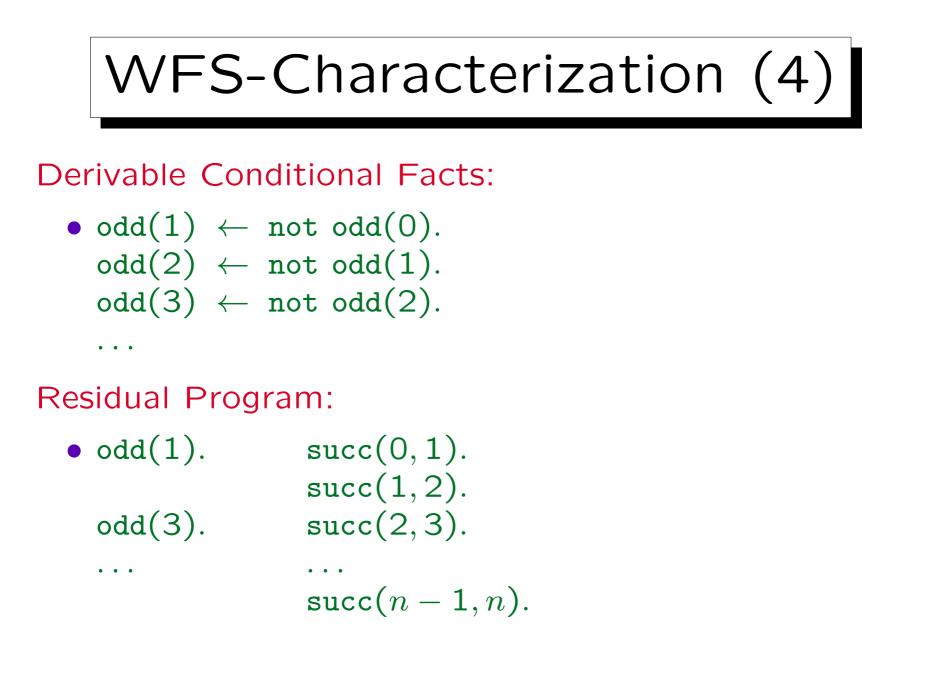
 $A \leftarrow L_1 \land \cdots \land L_{i-1} \land \mathsf{not} \ B \land L_{i+1} \land \cdots \land L_n,$

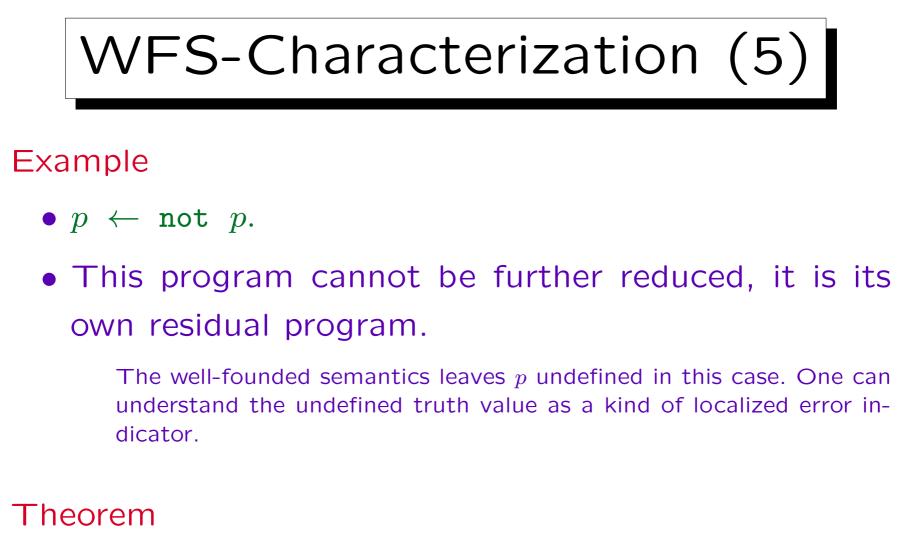
where B occurs in no rule head, by

$$A \leftarrow L_1 \wedge \cdots \wedge L_{i-1} \wedge L_{i+1} \wedge \cdots \wedge L_n.$$









• The well-founded semantics allows the above five transformations.



Theorem:

- The well-founded model of P can be directly read from the residual program res(P):
 - A is true in the well-founded model iff res(P) contains the fact $A \leftarrow$ true.
 - A is false in the well-founded model iff res(P) contains no rule about A.
 - All other ground atoms are undefined in the wellfounded model.



