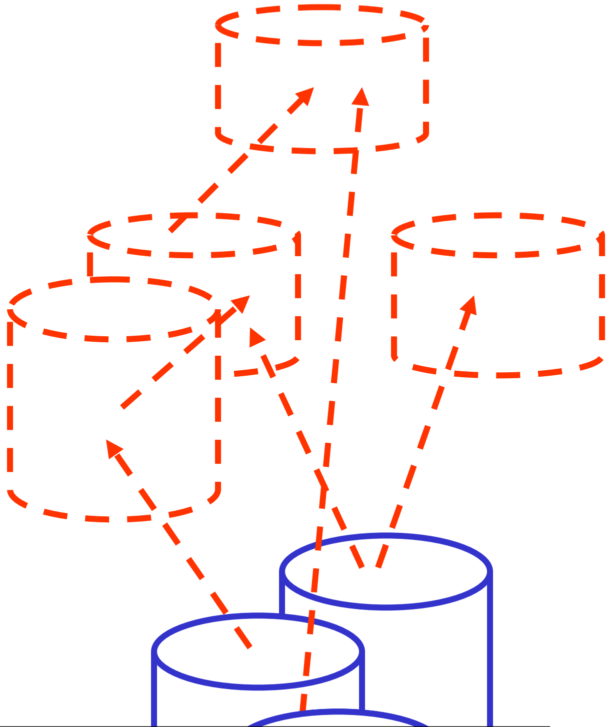
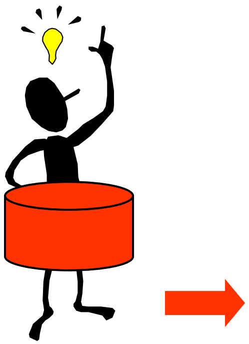
Intelligent Information Systems

SS 2010

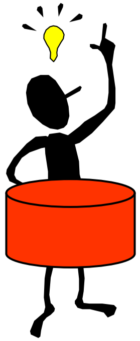
2. Deductive Databases and Datalog





2.1 Deductive DB: Overview
2.2 Datalog: Learning by Doing
2.3 Datalog: Facts and Rules
2.4 DDL and DML for Datalog

2.1 Deductive DB: Overview
2.2 Datalog: Learning by Doing
2.3 Datalog: Facts and Rules
2.4 DDL and DML for Datalog



2.4.1 Data Definition Language

Schemas in Datalog (1)

- As for each database: Declaration of every relation is required, before the DB can be opened and facts be inserted (or rules be defined).

Database Schema

- In Datalog: There is no agreed notation or convention for schemas; most authors simply don‘t mention a schema explicitly – we have our own notation.

- Datalog is an untyped language (like Prolog), i.e., there are no attributes and no explicit value domains. However, we will use types implicitly throughout.

- In a schema: Only the arity of a relation and its name are declared.

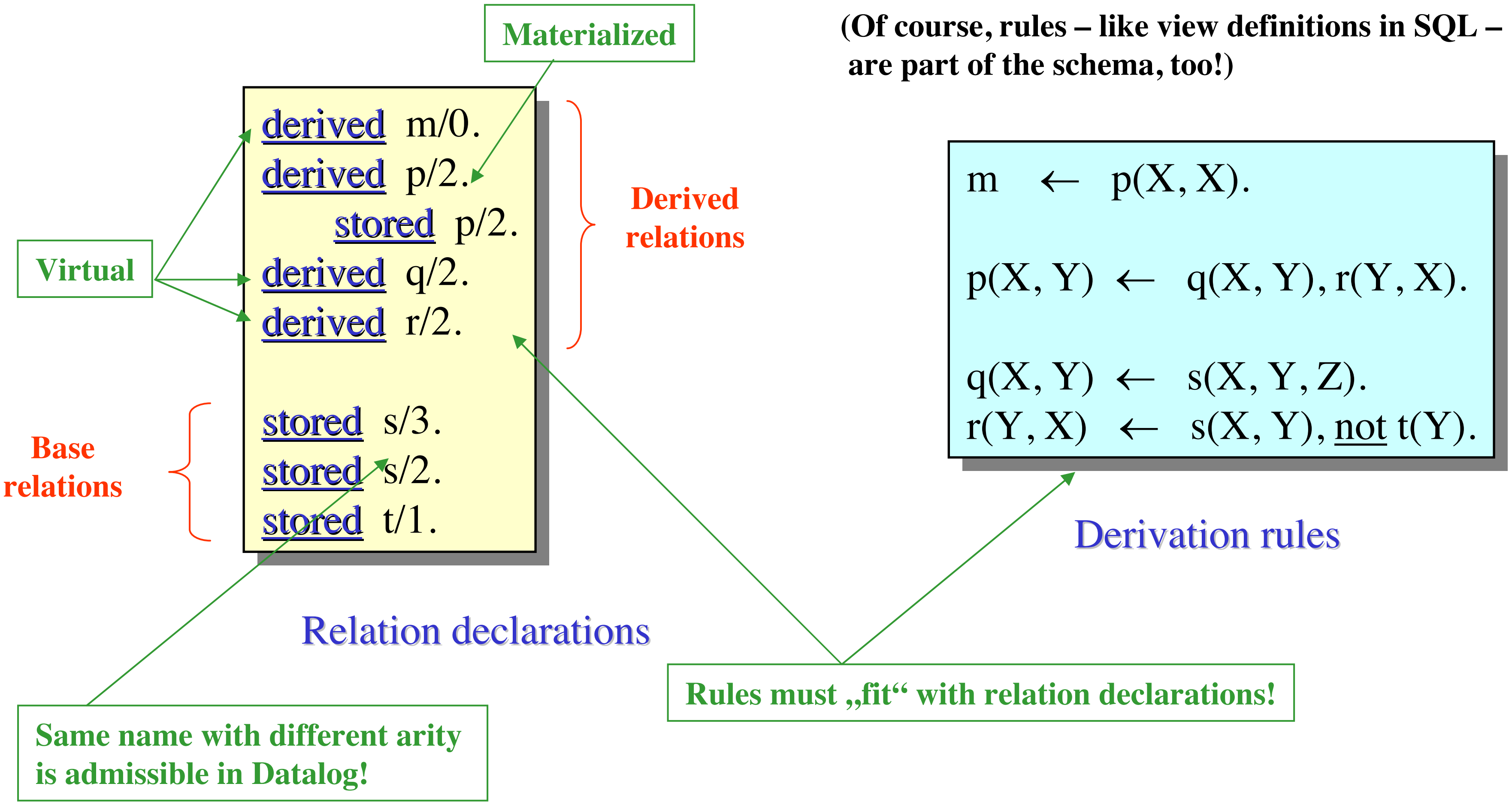
- The "form of definition" of the relation, however, is to be fixed in the schema, too:
  - Relation defined by facts only: stored
  - Relation defined by rules only: derived

engl. "to store": speichern
engl. "to derive": ableiten

- Both variants can be combined, i.e., there are three relation types:
  - Base relation: stored (but not derived)
  - virtual relation: derived (but not stored)
  - materialized relation : both, stored and derived

Schemas in Datalog (2) 2.4

Example of a simple Datalog schema:



(Of course, rules – like view definitions in SQL – are part of the schema, too!)

Materialized

Virtual

derived m/0.
derived p/2.
stored p/2.
derived q/2.
derived r/2.

stored s/3.
stored s/2.
stored t/1.

Derived relations

Base relations

Relation declarations

$m \leftarrow p(X, X).$

$p(X, Y) \leftarrow q(X, Y), r(Y, X).$

$q(X, Y) \leftarrow s(X, Y, Z).$
$r(Y, X) \leftarrow s(X, Y), \underline{\text{not}}\ t(Y).$

Derivation rules

Rules must „fit“ with relation declarations!

Same name with different arity is admissible in Datalog!

Schemas in Datalog (3)

Surprisingly, there is no agreed notation for integrity constraints in Datalog either – thus, we introduce our own style for expressing normative rules, too:

```
derived  m/0.
derived  p/2.
      stored  p/2.
derived  q/2.
derived  r/2.

stored  s/3.
stored  s/2.
stored  t/1.
```

```
m      ←  p(X, X).

p(X, Y) ←  q(X, Y), r(Y, X).

q(X, Y) ←  s(X, Y, Z).
r(Y, X) ←  s(X, Y), not t(Y).
```

Constraints may apply to both, stored and derived relations.

Positive condition: „Must always be true!"

```
constraint  t(c).

constraint  not m.
constraint  not q(a, b).
```

Negative condition: „May never be true!"

Integrity constraints

Integrity Constraints in the Genealogy DB (1)

In our sample database representing a family tree a lot of „nonsense“ can be accomodated without integrity constraints, e.g.:

```
born_in('John', 1978).
born_in('Mary', 1933).
born_in('Mary', 1968).

died_in('Jim', 1999).
died_in('John', 1945).
died_in('Jim', 2000).

sex_of('Jim', n).
sex_of('male', m).
```

```
child_of('William', 'Diana', 'Charles').
child_of('John', 'Caroline', 'William').
child_of('Charles', 'Christine', 'John').

married_to('Charles', 'Diana', 1981).
married_to('Charles', 'Anne', 2004).

divorced_from('Diana', 'Charles', 1996).
divorced_from('Elizabeth', 'Charles', 2002).

title_of('Charles', 'King of Great Britain').
title_of('William', 'King of Great Britain').
```

Each of these facts is somehow "inadmissible"! How to prevent them in Datalog?

Integrity Constraints in the Genealogy DB (2)

ICs for the genealogy DB: **(formulated in natural language first)**

IC 1 Each person who has died must have been born before.

IC 2 There is just one person per name.

IC 3 Each person‘s sex is (either) male or female.

IC 4 If two people divorce they have to be married at the time of divorce.

IC 5 In the 'married_to'-relation, the 1st column always contains the name of the wife, whereas the husband is recorded in the 2nd column.

IC 6 (analogously for 'divorced_from')

IC 7 (analogously for 'child_of')

IC 8 The 1st column in the 'sex_of'-relation contains the name of a person.

IC 9 Nobody must be a descendant of himself or of one of his own descendants (i.e., the 'descendant_of'-relation is acyclic).

IC 10 Nobody must be married to a relative of degree 1 or 2.

IC 11 Each title must be held by at most one living person a time.

IC 12 Nobody may get married before being born.

Integrity Constraints as Universal Axioms

IC 1 Each person who has died must have been born before.

- Most integrity constraints are universal statements (like IC1):
  "For all" objects of a particular type a certain condition holds!
- "For all" is circumscribed in natural language most of the time (even negatively):
  "each", "only", "all", "always", "nobody"
  (or is not made explicit at all, as in IC 4 and IC 8)
- In (full) predicate logic, such conditions are expressed using a universal quantifier, in most cases coupled with implication for restricting the scope of the statement:

$$\forall \ X, Y1: \ died\_in(X,Y1) \Rightarrow \exists \ Y2 \ (born\_in(X,Y2) \wedge \ Y2 \leq Y1)$$

- Like SQL, Datalog does not offer any universal quantifier, but only the (implicit) existential quantifier, so that a transformation using Boolean algebra is required:

$$\neg \ \exists \ X, Y1: \ (died\_in(X,Y1) \ \wedge \ \neg \exists \ Y2 \ (born\_in(X,Y2) \wedge \ Y2 \leq Y1 \ ))$$

Expressing ICs in Datalog (1)

- Universal conditions are thus turned into negated existential conditions.

- Existential conditions in Datalog are expressible via parameterless relations (arity 0) defined (once again – a very important concept!) by auxiliary rules:

$$\neg\, \underbrace{\exists\ X, Y1:\ (died\_in(X,Y1)\ \wedge\ \neg \exists\ Y2\ (\ born\_in(X,Y2)\ \wedge\ \ Y2 \leq Y1\ ))}_{p1\ \leftarrow\ died\_in(X,Y1),\ \underline{not}\ (\ .\ .\ .\ .\ )}$$

$\exists\ X, Y1$

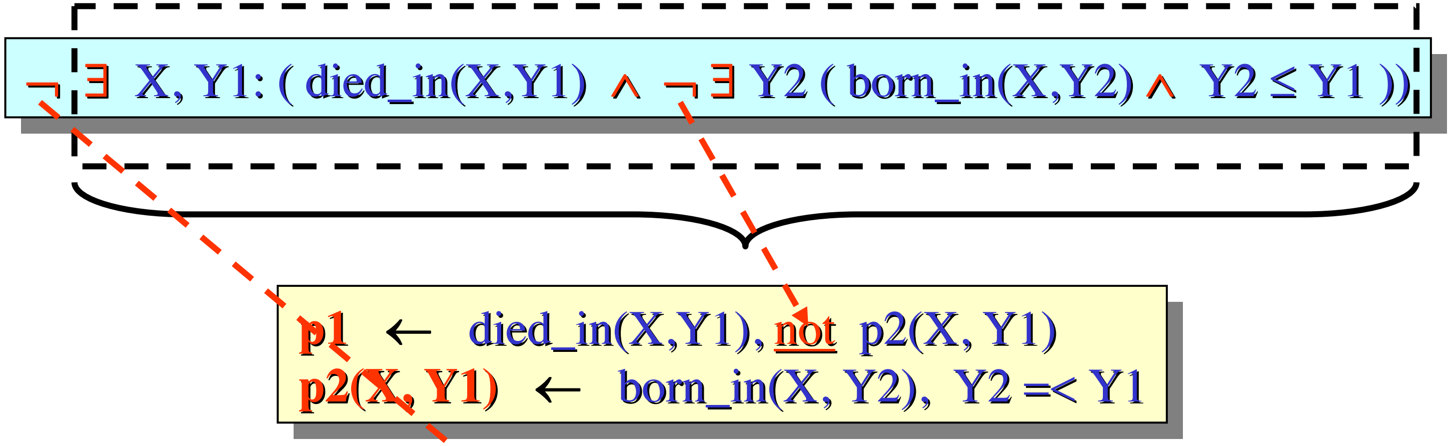
- As nesting is banned from Datalog, the conjunction has to be moved "behind" the negation operator and pushed into another auxiliary rule:

$$p1\ \leftarrow\ \ died\_in(X,Y1),\ \underline{not}\ \ p2(X, Y1)$$
$$p2(X, Y1)\ \ \leftarrow\ \ born\_in(X, Y2),\ \ Y2 =< Y1$$

$\exists\ Y2$

Expressing ICs in Datalog (2)

Thus, the positive part of the existential condition has been properly formulated in Datalog:



$\neg \exists X, Y1: (\, died\_in(X,Y1) \wedge \neg \exists Y2 \,(\, born\_in(X,Y2) \wedge Y2 \leq Y1 \,))$

```
p1         ←  died_in(X,Y1), not p2(X, Y1)
p2(X, Y1)  ←  born_in(X, Y2), Y2 =< Y1
```

What is missing is just the integrity constraint itself (including the negation), which we will identify from now on by the keyword constraint:

```
constraint  not  p1
```

Datalog Schemas: Summary

- A Datalog schema consists of the following parts:

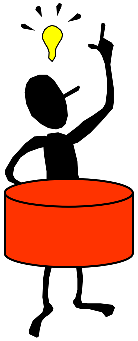
  - Relation declarations of the form [derived | stored] <Name> / <Arity>
    - Relation names are strings beginning with a lower case character.
    - Arities are positive integers or zero.
    - Per couple name/arity occurring in the DB there must be at least one declaration; double declarations (derived and stored) are possible.

  - Derivation rules of the form <Head> ← <Body>.
    - The rule head is a positive literal for a relation declared as derived.
    - The rule head is a conjunction of literals (or just a single literal).
    - Each derived relation is defined by at least one rule.

  - Integrity contraints of the form constraint <Literal>.
    - The literal must be variable-free (a ground literal, positive or negative).

- In addition, various "wellformedness conditions" have to be respected.

2.1 Deductive DB: Overview
2.2 Datalog: Learning by Doing
2.3 Datalog: Facts and Rules
2.4 DDL and DML for Datalog

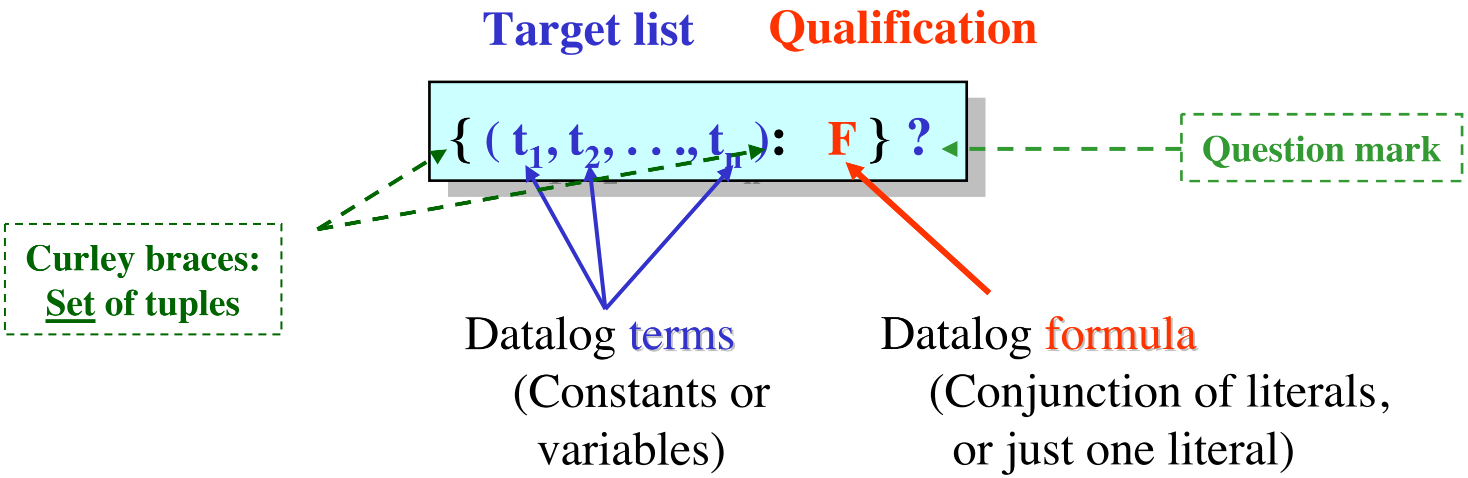


2.4.2 Data Manipulation Language

Queries in Datalog

- As in each query language, there are in Datalog two types of queries in Datalog, too:
  - Retrieval queries serve as a means of access to facts in the DB.
  - Test queries are employed if testing yes/no conditions.

- The syntax of retrieval queries closely resembles the mathematical concept of an intensional set expression {X | f(X)} ("Set of all x, such that f(X) holds")



Target list **Qualification**

$\{ (t_1, t_2, . . ., t_n): F \} ?$

Curley braces:
Set of tuples

Question mark

Datalog terms
(Constants or
variables)

Datalog formula
(Conjunction of literals,
or just one literal)

Retrieval Queries

Examples of retrieval queries (with their corresponding answer sets):

{ (S) **:** son_of(S, 'Elizabeth')} **?**

Which sons does Elizabeth have?

Answer: { ('Charles'), ('Andrew'), ('Edward') }

{ (X) **:** grandchild_of(X, 'Elizabeth'), mother_of(Y, X), not divorced(Y)} **?**

Which grandchildren of Elizabeth have a mother, who is presently not divorced?

Answer : { ('Peter'), ('Zara') }

Analogy Queries ⇔ Rules

Apparently: Queries and rules are syntactically very similar!

Target list	⇔	Rule head
Qualification	⇔	Rule body

e.g.:

{ (X, Y) : father_of(X, Z), child_of(Y,Z) } ?

grandfather_of (X, Y) ← father_of(X, Z), child_of(Y,Z) .

Derivable relation:

Named answer set of a query

Safe Queries

- Due to this analogy

Safety requirement for queries necessary!

- "Safe queries":
  - All variables in the target list must occur in the qualification, too.
  - All variables in negative literals must occur in at least one positive literal, too.

- In analogy with rules, too:
  - Local variables in the qualification are implicitly existentially quantified at the beginning of the qualification part.
  - Answer sets are free of duplicates.

Rules are "stored queries".

Auxiliary Relations for Queries (1)

- Various reasonable queries cannot be expressed up till now, due to syntactic restrictions in Datalog such as forbidden nesting and missing disjunctions, e.g.:

Find all persons who are first or second cousins of William?

- Queries like this require rules to be added, even though these rules are not really suitable as parts of the DB schema (being rather occasional or „ad hoc“ in nature) – in our example:

```
cousin1_or_2_of (X,Y)  ←  cousin_of(X,Y).
cousin1_or_2_of (X,Y)  ←  cousin_of(X,Z), parent_of(Z,Y).
```

Implicit disjunction

- The query itself may then refer to the new rule-defined concept:

{(X) : cousin1_or_2_of (X, 'William'} ?

Auxiliary Relations for Queries (2)

- As such rules better don‘t „pollute“ the schema, it is a good idea to „embed“ them into the respective query, indicating that they will not be retained outside this query:.

⇒ "Local Rules" inside a query are in fact inevitable !

- In the example:

Additional delimiter for lists of local rules!

```
{ (X) : cousin1_or_2_of (X, "William") }
        with  cousin1_or_2_of (X,Y)  ←  cousin_of(X,Y) ;
              cousin1_or_2_of (X,Y)  ←  cousin_of(X,Z), parent_of(Z,Y) ?
```

- The newly introduced keyword with for indicating that rules are local to a query has been inspired by the same word in SQL:1999, where it serves a similar purpose within view declarations.

- Locally defined relations may not be used outside the respective query and are no part of the DB schema!

Special Cases of Retrieval Queries

- Syntactically admissible, but rarely necessary: Constants in the target list

e.g.:

{ (X, **a**, Y) : p(X, Y) } ?

{ (**a**) : p(X, a) } ?

- Finally as a kind of "extreme case": Queries with an empty target list,
  a funny way of expressing Boolean existential queries

∃ X :

{ ( ) : p(X, a) } ?

Antwort { ( ) } ⟶ true
{ } ⟶ false

But that's the way you have to do it in SQL, folks!

Test Queries 2.4

- Rather than forcing users to formulate test queries in this style as "degenerate" retrieval queries (as in SQL), „our“ Datalog offers a special syntax for this:
  - based on parameterless rule heads (implicit ∃),
  - or consisting of ground literals only (or both)

∃ X :

```
exists_widower
          with    exists_widower ← widower(X) ?

single('Charles') ?
```

- Negated or conjunctive test queries are possible, too :

```
single('Charles'), widower('Charles')  ?

not  exists_orphan
           with exists_orphan ← orphan(X) ?
```

Integrity Constraints vs. Test Queries 2.4

- Important: It is not sufficient to introduce a parameterless relation defined by a derivation rule for expressing a constraint!

- Reason: Parameterless relations may be queried (as tests), but are not automatically combined with the request to the DBMS to automatically enforce this test!

1)

```
derived consistent/0.

consistent ←   p(X), not q(X).
```

2)

```
derived consistent/0.

constraint   consistent.

consistent ←   p(X), not q(X).
```

3)

```
constraint   consistent
   with   consistent   ←   p(X), not q(X) .
```

1) and 2): Test query "consistent ?" can be posed

3): Relation "consistent/0" is not queriable (as local to the integrity constraint)

1): Answer may be true or false.

2): Answer is always true.

Updates in Datalog 2.4

- For updates (DB changes, i.e., insertions, deletions and modifications of facts) in Datalog there is no generally accepted syntax and semantics in the literature either. Again we need a notation of our own, which seemlessly fits with the rest of the language. (Note that we don‘t consider schema changes in this lecture.)

- We will introduce insertions (+) und deletions (–) only:
  Modifications of individual attribute values can be simulated by combining +/–.

- For now: Only updates of base relations are considered !!

- Syntactical buliding block of all Datalog updates:
  (Literals with a sign + / –)

  Dynamic Literal

- All update statements are delimited by an (imperative) exclamation mark.

e.g.:

```
+  successor('Elizabeth', 'William')  !
–  successor('Elizabeth', 'Charles')  !
```

Updates: Syntax

- Elementary updates (insertions/deletions) of individual facts in a relation are expressed by means of dynamic ground literals, e.g.:

```
+  title('Paul McCartney', 'Sir') !
```

- Conditional updates: Several, simultaneously executable elementary updates depending on a condition (syntactically like a rule body)

```
+  title(X, 'Sir') : member_of(X, 'Beatles') !
```

Dynamic literal

Condition

- Safety requirement for conditional updates: In analogy to rules and queries

- For conditional updates, local rules are possible, too:

```
+   p(X) :   r(X),  not s(X)
                          with    s(X) ← t (X,Y) !
```

Analogy Updates ⇔ Queries ⇔ Rules

- The analogy already observed for queries and rules carries over to conditional updates:

dynamic literal	⇔	target list	⇔	rule head
condition	⇔	qualification	⇔	rule body

e.g.:

{ (X, Y) : father_of(X, Z), child_of(Y,Z) } ?

grandfather_of (X, Y) ← father_of(X, Z), child_of(Y,Z) .

+ grandfather_of (X, Y) : father_of(X, Z), child_of(Y,Z) !

- For elementary updates/facts/test queries similarly:

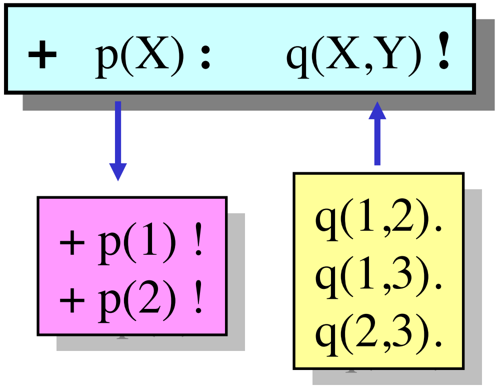
+ born_in('Paul', 1943) !

born_in('Paul', 1943) .

born_in('Paul', 1943) ?

Updates: Semantics

- Conditional updates are interpreted in a set-oriented manner (so are retrieval queries and derivation rules!):



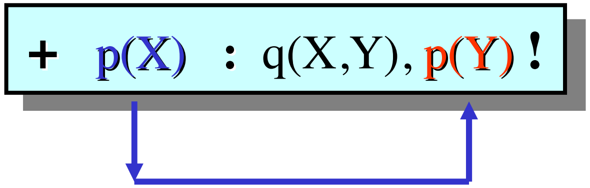
+ p(X) : q(X,Y) !

+ p(1) !
+ p(2) !

q(1,2).
q(1,3).
q(2,3).

- All p-insertions are performed simultaneously.
- The condition is completely evaluated before any change takes place.

- Condition evaluation and DB-update are performed strictly separately in two phases. Thus, an update may not cause any retroactive effects on the evaluation of its own condition:

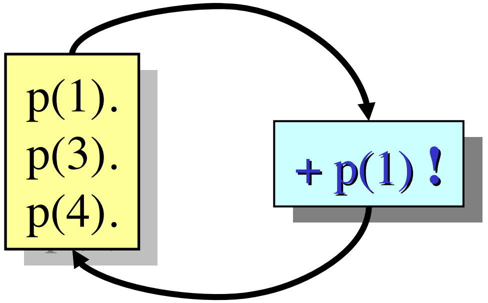


+ p(X) : q(X,Y), p(Y) !

Redundant Updates

- Elementary updates which don‘t have any changing effect on the current state are accepted: Redundant updates

e.g.:



(similarly for - p(2) !)

- This is useful in particular for conditional updates where some of the variable bindings resulting from condition evaluation have an effect, whereas others don‘t:

e.g.:

~~p(1).~~	q(1).
	q(2).
p(3).	
~~p(4).~~	q(4).

- p(X) : q(X) !

- Transactions for databases are " blocks of changes " either executed entirely, or not at all.

⇒ State transitions between DB states are expressible via transactions which could not be executed individually due to integrity violations.

- In SQL: Transactions are sequences, introduced and concluded by special keywords.

- For Datalog: "Set-oriented" transaction concept seems to be appropriate, i.e.:

Simultaneous execution of several changes without fixing any ordering of steps.

- Syntax format for transactions:

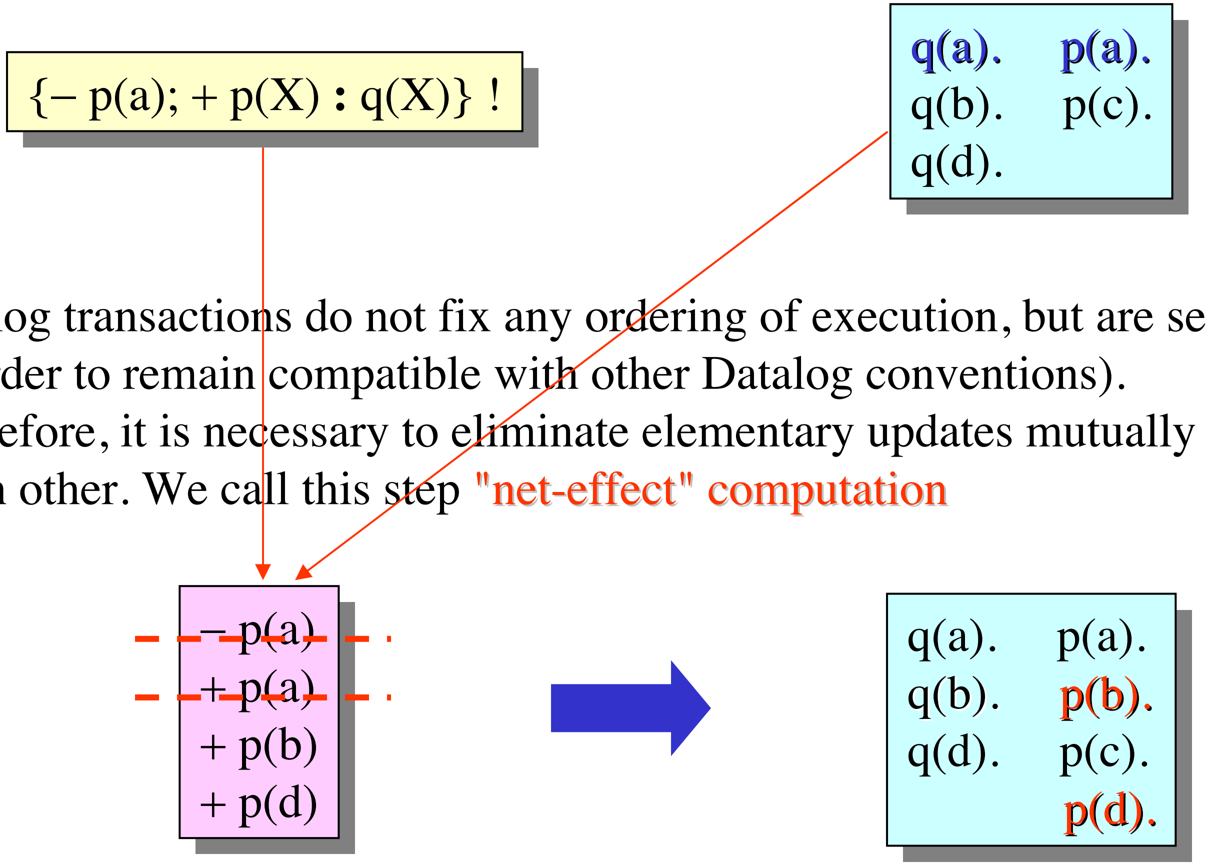
$\{ U_1; U_2; \ldots; U_n \}$!

(U_i elementary or conditional updates without '!')

Contradictoy Updates and Net Effect

- If using conditional updates withín a transaction, it may happen that contradictory elementary updates result from evaluating different parts of the transaction. Here, *contradictory* means that the same elementary update appears with positive and negative sign, e.g.:

- Datalog transactions do not fix any ordering of execution, but are set-oriented (in order to remain compatible with other Datalog conventions). Therefore, it is necessary to eliminate elementary updates mutually compensating each other. We call this step "net-effect" computation



{– p(a); + p(X) : q(X)} !

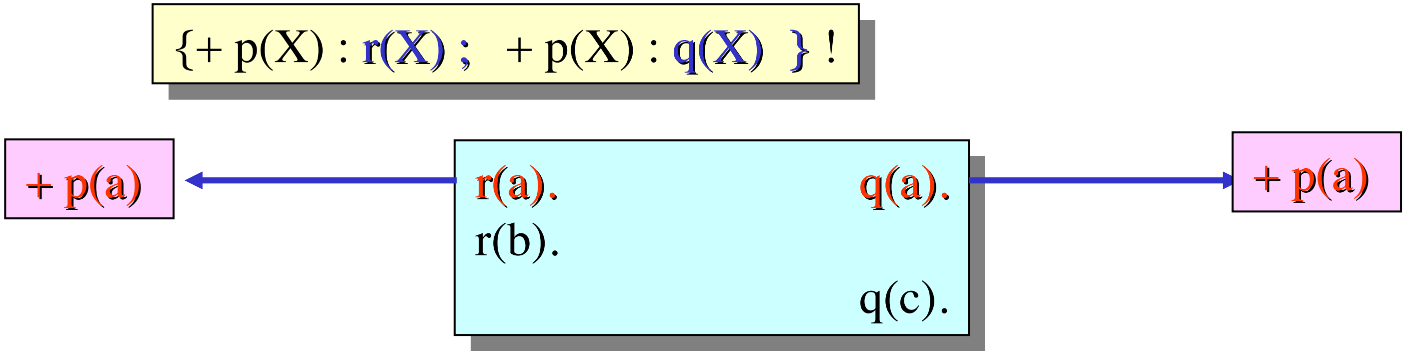
q(a). p(a).
q(b). p(c).
q(d).

~~– p(a)~~
~~+ p(a)~~
+ p(b)
+ p(d)

q(a). p(a).
q(b). p(b).
q(d). p(c).
p(d).

Net Effect Compuation and Duplicates

- It may happen that different parts of a transaction appear to cause the same elementary update to be executed several times:



{+ p(X) : r(X) ;   + p(X) : q(X) } !

- This kind of "redundancy" among dynamic literals is to be treated according to the general design principles of Datalog, too. As duplicates are eliminated in queries and during rule application (due to sets being free of duplicates), we will eliminate all dynamic literals appearing more than once before computing contradictory updates and net effect.

- Common, recurring principle underlying all these conventions:
  - No duplicates in Datalog (all results are sets).
  - No ordering in Datalog.

DML: Summary

- Datalog‘s DML offers syntactical means for expressing queries and updates.

- Queries are either . . .
  - . . . retrieval queries in the format { <Target list> | <Qualification> } . . .
  - . . . or variable-free test queries in the format <Qualification>

    (Qualifications are literals or conjunctions of literals.)
  - All queries are terminated by the symbol ?.

- Updates are either . . .
  - . . . dynamic literals in the format + <Literal> or – <Literal> . . .
  - . . . or transactions in the format {<List of updates>}.
  - All updates are terminated by the symbol !.

- Queries and updates have to be safe – like rules.

- Queries and updates have a set-oriented semantics.

- Updates are executed only if integrity constraints are not violated after the change.