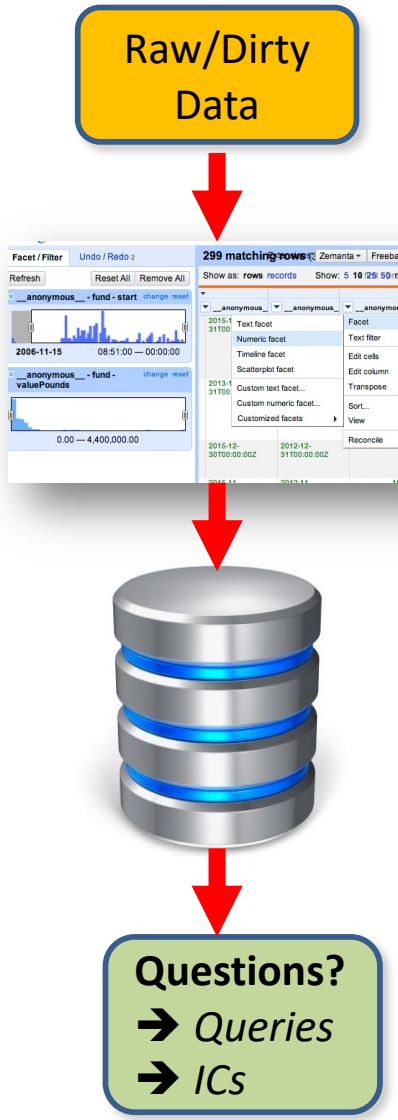
From Syntax to Schema & Semantics

- After ***pattern-based cleaning***
  - regular expressions, OpenRefine, …
- … load data into a database system!
- … and then exploit database technology:
  - ***queries*** & ***integrity constraints***!

- **Relational Databases**
  - *Logic-based* approach first: **Datalog**
    - Facts, rules, queries, integrity constraints
    - Rich body of research; theory & practice!
  - *Relational data everywhere*: **SQL**



Semantics / ICs in Conceptual Models (ER, UML, OO, …)

- What we *can* express:
  - e.g., functional dependencies (key constraints),
  - inclusion dependencies (foreign key constraints),
  - cardinality constraints,
  - hierarchical constraints (class hierarchy), …
- What we *cannot* express:
  - e.g., general arithmetic constraints,
  - application-specific constraints and knowledge,
  - purpose, goals, other general requirements, …

Dirty/Clean data as In-/Consistent data

Students	*StuNum*	*StuName*
	101	*john bell*
	102	*mary stein*
	104	*claire stevens*
	107	*pat norton*

Enrollment	*StuNum*	*Course*
	104	*comp150*
	101	*comp100*
	101	*comp200*
	105	*comp120*

Figure 1.1: A database instance.

Source: [bertossi2011database]

- Important **Integrity Constraints** (ICs)
  - **Functional Dependencies** FD (≈ **key** constraints)
    - one or more columns provide a unique key
  - **Inclusion Dependencies** ID (≈ **referential** ICs)
    - Child_Table.FK → Parent_Table.PK

Is this consistent?

Students	*StuNum*	*StuName*
	101	*john bell*
	101	*joe logan*
	104	*claire stevens*
	107	*pat norton*

Enrollment	*StuNum*	*Course*
	104	*comp150*
	101	*comp100*
	101	*comp200*

Figure 1.2: An inconsistent instance.

Students1	*StuNum*	*StuName*
	101	*john bell*
	104	*claire stevens*
	107	*pat norton*

Students2	*StuNum*	*StuName*
	101	*joe logan*
	104	*claire stevens*
	107	*pat norton*

Figure 1.3: Two repairs.

Source: [bertossi**2011**database]

Integrity Constraints as Logic Rules & Queries

The two particular kinds of integrity constraints presented above, and also other forms of ICs, can be easily expressed in the language of first-order (FO) predicate logic [Enderton, 2001], which in relational databases usually takes the form of the *relational calculus* [Abiteboul et al., 1995]. For example, the FD above can be expressed by the symbolic sentence

$$\forall x \forall y \forall z((Students(x, y) \wedge Students(x, z)) \longrightarrow y = z), \quad (1.1)$$

whereas the referential constraint above can be expressed by

$$\forall x \forall y(Enrollment(x, y) \longrightarrow \exists z Students(x, z)). \quad (1.2)$$

Notice that this language of FO predicate logic is determined by the database schema, whose predicates are now being used to write down logical formulas. We may also use "built-in" predicates,

Source: [bertossi**2013**generic]

```
ic_violated(X) :- student(X,Y), student(X,Z), Y != Z.
```

First-Order Queries + Datalog: 4-in-1

- SQL — SELECT ... FROM ... WHERE ...
- Relational Algebra (RA) — $\sigma, \pi, \bowtie, \delta, \cup, \setminus$
- Relational Calculus (RC) — $\forall x F,\ \exists x F,\ F \wedge G,\ F \vee G, \neg F$
- **Datalog** — $\approx$ RC + Recursion

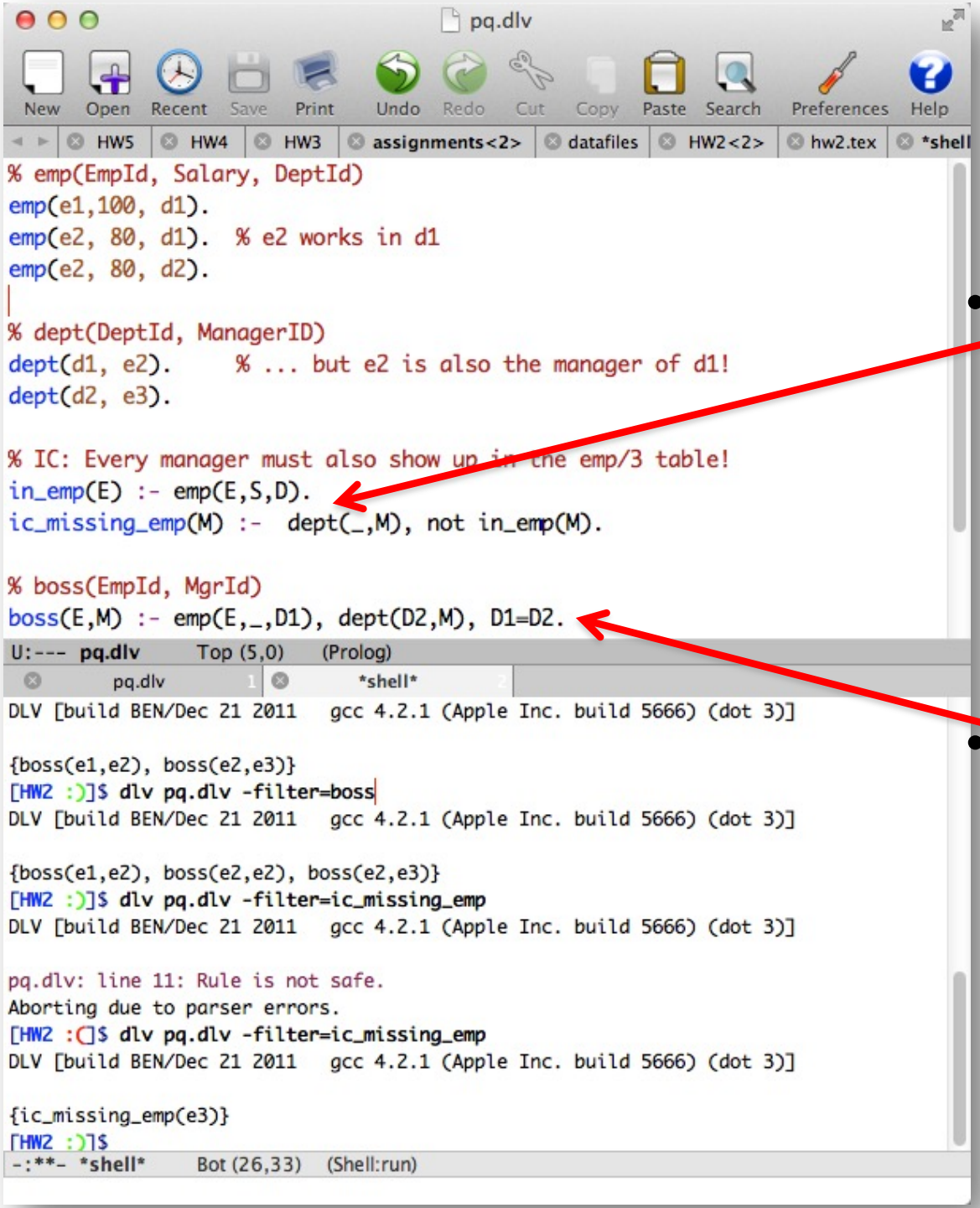
EXAMPLE: Given relations `employee(Emp, Salary, DeptNo)` and `dept(DeptNo, Mgr)`, find all (employee, manager) pairs:

- SQL:
```
SELECT Emp, Mgr
FROM employee, dept
WHERE employee.DeptNo = dept.DeptNo
```
- RA: $\pi_{\mathtt{Emp},\mathtt{Mgr}}(\mathtt{employee} \bowtie \mathtt{dept})$
- RC: $F(\mathtt{Emp},\mathtt{Mgr}) =$
  $\exists \mathtt{Salary},\ \mathtt{DeptNo} : (\mathtt{employee(Emp,\ Salary,\ DeptNo)} \wedge \mathtt{dept(DeptNo,Mgr)})$
- **Datalog**: `boss(Emp,Mgr) ← employee(Emp, Salary, DeptNo), dept(DeptNo,Mgr)`

Two more ways to write the same query:

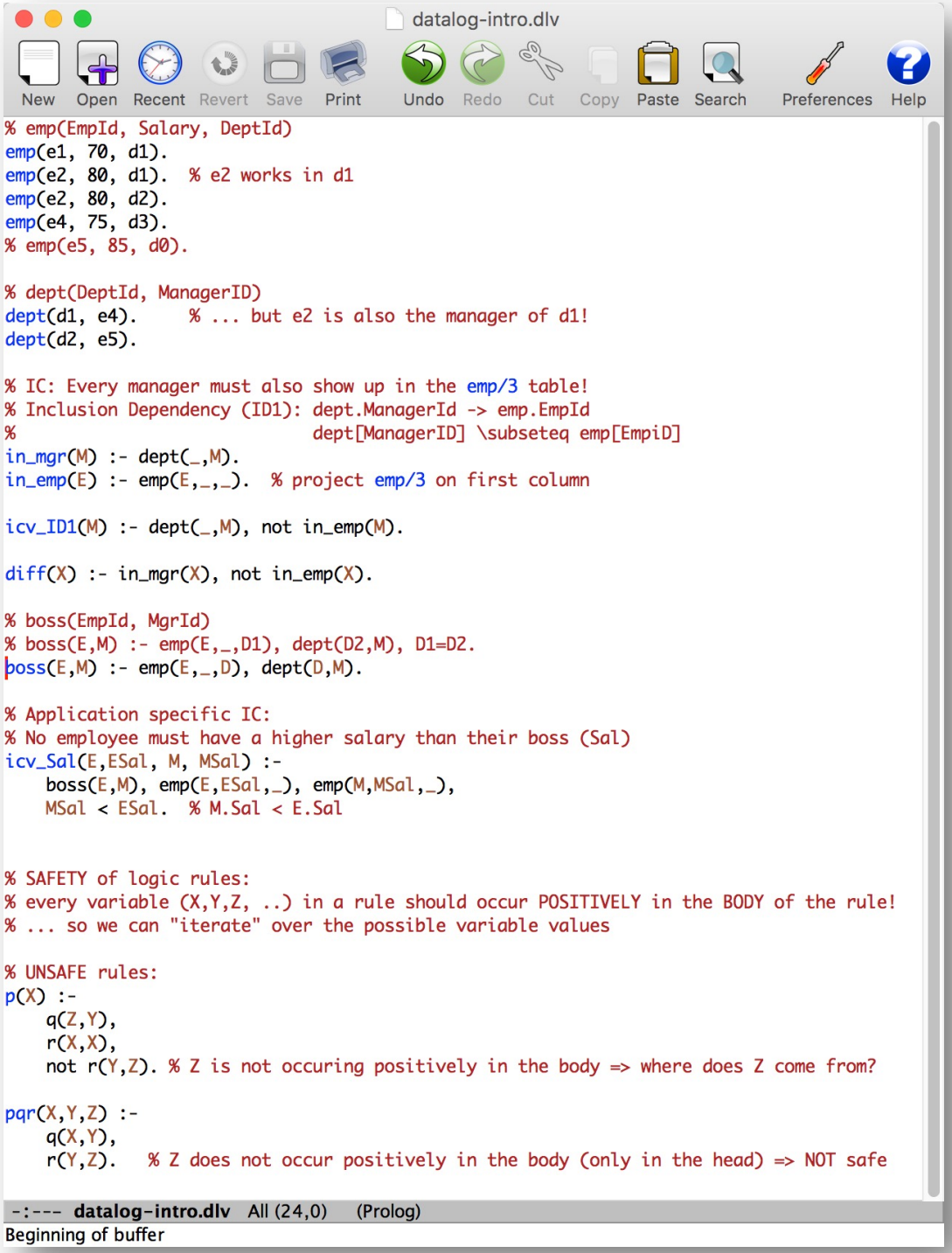
```
boss(E,M) :- employee(E,S,D1), dept(D2,M), D1=D2.
boss(E,M) :- employee(E,_,D), dept(D,M).
```

ASP/DLV Demo (inside Emacs)

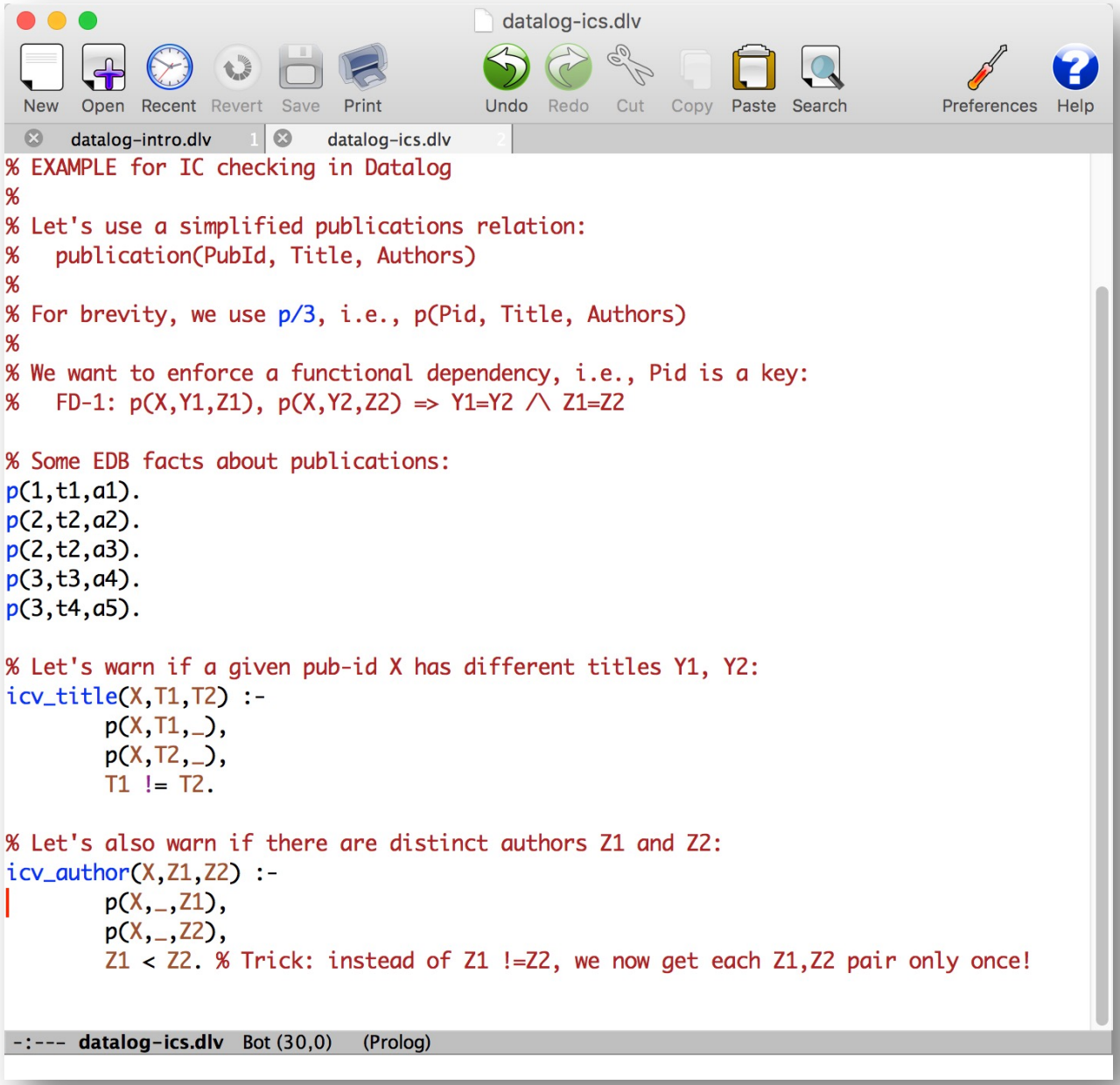


- An **integrity constraint** (IC): every dept manager should be in the emp/3 table (= foreign key constraint)
- A **query**: list employees with their “boss(es)”, i.e., the managers of the departments the employee works for.

Integrity Constraints in Datalog

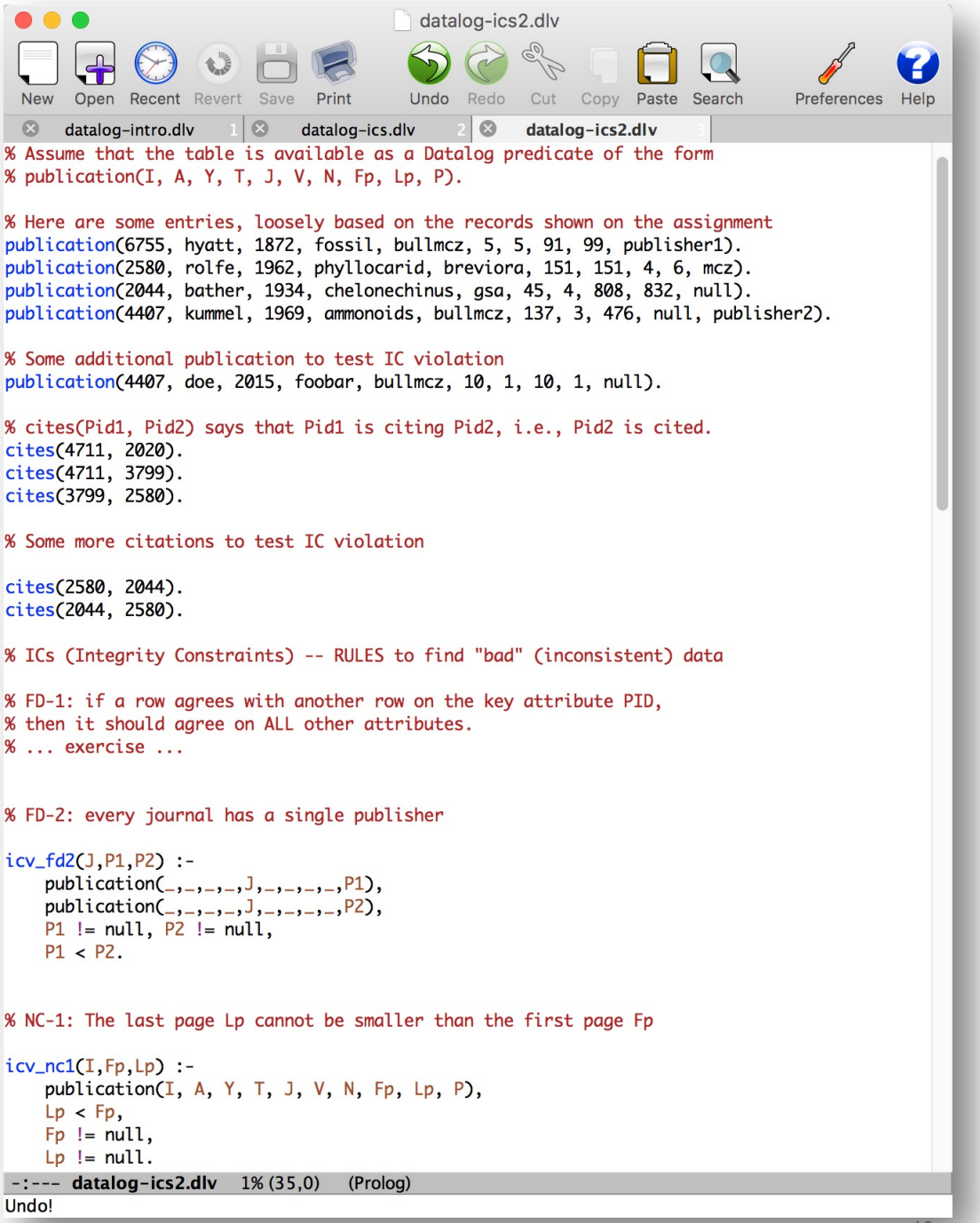


Integrity Constraints in Datalog



ICs in Datalog

=> Assignment



From IC Checking to IC **Repair** ..

Fig. 1 A database instance

Students

StuNum	*StuName*
101	john bell
102	mary stein
104	claire stevens
107	pat norton

Enrollment

StuNum	*Course*
104	comp150
101	comp100
101	comp200
105	comp120

Fig. 2 Another instance

Students

StuNum	*StuName*
101	john bell
101	joe logan
104	claire stevens
107	pat norton

Enrollment

StuNum	*Course*
104	comp150
101	comp100
101	comp200

one student name. This condition, called a *functional dependency* (FD), is denoted with $StuNumber \rightarrow StuName$, or $Students : StuNumber \rightarrow StuName$, to indicate that this dependency should hold for attributes of relation *Students*. Actually, in this case, since all the attributes in the relation functionally depend on *StuNum*, the FD is called a *key constraint*.

Integrity constraints can be declared together with the schema, indicating that the instances for the schema should all satisfy the integrity constraints. For example, if the functional dependency $Students : StuNumber \rightarrow StuName$ is added to the schema, the instance in Fig. 1 is consistent, because it satisfies the FD. However, the instance in Fig. 2 is *inconsistent*. This is because this instance does not satisfy, or, what is the same, violates the functional dependency (the student number *101* is assigned to two different student names.

From IC Checking to IC **Repair** ..

It is also possible to consider with the schema a *referential integrity constraint* that requires that every student (number) in the relation *Enrollment* appears, associated with a student name, in relation *Students*, the official "table" of students. This is denoted with $Enrollment[StuNum] \subseteq Students[StuNum]$ and is a form of *inclusion dependency*. If this IC is considered in the schema, the instance in Fig. 1 is inconsistent, because student *105* does not appear in relation *Students*. However, if only this referential constraint were associated to the schema, the instance in Fig. 2 would be consistent. The combination of the given referential constraint and the functional dependency creates a *foreign key constraint*: The values for attribute *StuNum* in *Enrollment* must appear in relation *Students* as values for its attribute *StuNum*, and this attribute form a *key* for *Students*.

It can be seen that the notion of consistency is relative to a set of integrity constraints. A database instance that satisfies each of the constraints in this set is said to be *consistent* and *inconsistent* otherwise.

The two particular kinds of integrity constraints presented above and also other forms of ICs can be easily expressed in the language of predicate logic. For example, the FD above can be expressed by the symbolic sentence

$$\forall x \forall y \forall z((Students(x, y) \wedge Students(x, z)) \longrightarrow y = z), \qquad (1)$$

From IC Checking to IC **Repair** ..

Fig. 3 Two repairs of *Students* in Fig. 2

Students

StuNum	*StuName*
101	john bell
104	claire stevens
107	pat norton

Students

StuNum	*StuName*
101	joe logan
104	claire stevens
107	pat norton

3 Repairs and Consistent Answers

The notion of consistency of a database is a holistic notion that applies to the entire database, and not to portions of it. In consequence, in order to pursue this idea of retrieving consistent query answers, it becomes necessary to characterize the consistent data in an inconsistent database first. The idea that was proposed in [2] is as follows: the consistent data in an inconsistent data are the data that are invariant under all possible way of restoring the consistency by performing minimal changes on the initial database. That is, no matter what minimal consistency restoration process is applied to the database, the consistent data stay in the database. Each of the consistent versions of the original instance obtained by minimal changes is called a *minimal repair*,or, simply, a *repair*.

It becomes necessary to be more precise about the meaning of minimal change. In between, a few notions have been proposed and studied (cf. [9,10,35] for surveys of CQA). Which notion to use may depend on the application. The notion of minimal change can be illustrated using the definition of repair given in [2]. First of all, a database instance D can be seen as a finite set of ground atoms (or database tuples) of the form $P(\bar{c})$, where P is a predicate in the schema and $\bar{c}$ is a finite sequence of constants in the database domain. For example, *Students*(*101*, *john bell*) is an atom in the database. Next, it is possible to compare the original database instance D with any other database instance D' (of the same schema) through their symmetric difference $D \,\Delta\, D' = \{A \mid A \in (D \setminus D') \cup (D' \setminus D)\}$.

Repair with DLV (= Datalog + ASP)

repair.dlv

New Open Recent Save Print Undo Redo Cut Copy Paste Search Preferences Help

repair.dlv *shell

```
% STUDENT(Id, Name)
student(101,bell).
student(101,ball).
student(101,logan).
student(104,stevens).
student(107,norton).

% FK: Id -> Name

% Soft constraint
icv_fk(X,Y,Z) :-
        student(X,Y), student(X,Z), Y != Z.

% Hard constraint
% :- icv_fk(_,_,_).

% REPAIR (cf. Section 3.1 [bertossi2013generic.pdf])
del_student(X,Y) v del_student(X,Z) :- icv_fk(X,Y,Z).

new_student(X,Y) :-
        student(X,Y),
        not del_student(X,Y).
```

U:--- repair.dlv Top (12,0) (Prolog)

Wrote /Users/ludaesch/Box Sync/LIS590DCL-FA15/Datalog/ASP/repair.dlv

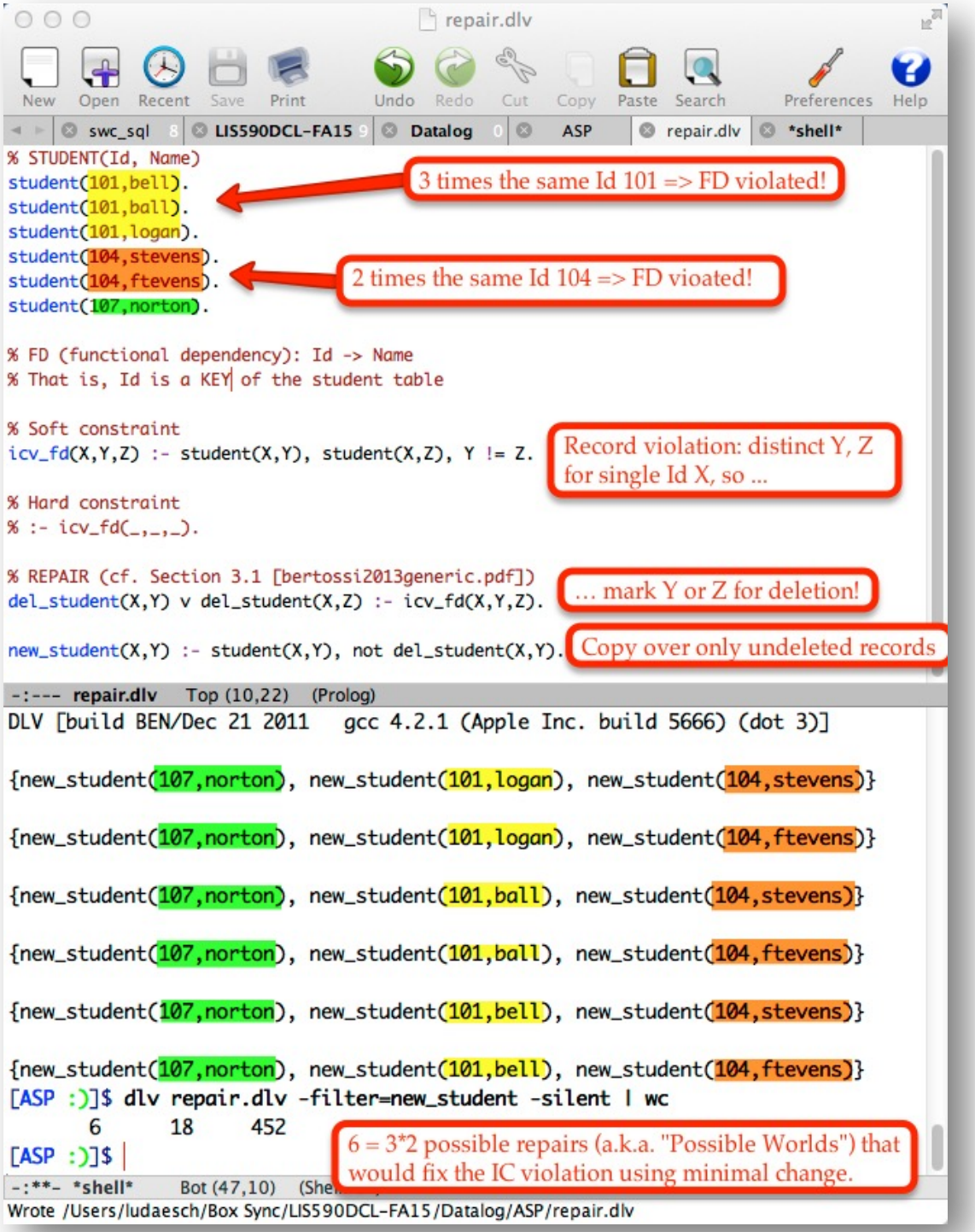
```
[ASP :)]$ dlv repair.dlv -filter=new_student
DLV [build BEN/Dec 21 2011   gcc 4.2.1 (Apple Inc. build 5666) (dot 3)]

{new_student(104,stevens), new_student(107,norton), new_student(101,logan)}

{new_student(104,stevens), new_student(107,norton), new_student(101,ball)}

{new_student(104,stevens), new_student(107,norton), new_student(101,bell)}
[ASP :)]$
```

- Based on [Bertossi2013]
- If an IC is violated, let the system find all possible minimal repairs!

Database Repair (Details)

- In this DB, the FD is violated for both
  - Id = 101 (3 times) and
  - Id = 104 (2 times),
- yielding 6 (=3*2) possible, minimal repairs

- ASP rules and Example based on earlier reading [Bertossi2013].



Datalog Summary

- You might not realize it, but you now have "**the power of logic**" at your disposal:
  - **Specifying** and **checking** constraints (structural / schema constraints; application semantics)
  - **Datalog**: "The better First-order Logic syntax"
  - **Graph Queries**, **Provenance Queries** (we'll revisit them)
- **Answer Set Programming (ASP)**
  - Datalog + AI-ish "*Generate & Test*" paradigm
  - Use it for **repairing** databases
  - cutting-edge CS research: e.g. Llunatic project