.

Chapter 12
Stable Models/Answer Set Programming

- ASP developed in the late 1990s.
- Introduction to ASP:
  Answer Set Programming: A Primer. T. Eiter, G. Ianni, T. Krennwallner.
  In “Reasoning Web. Semantic Technologies for Information Systems”, Springer LNCS 5689, 2009.
- XSB: XASP package: embeds stable models into XSB PROLOG programming.
  Not suitable for this lecture.
- smodels+lparse: `http://www.tcs.hut.fi/Software/smodels/`

SITUATION

Usually a program has several stable models (otherwise, the well-founded model is sufficient!)

- either one total well-founded model or one partial well-founded model,
- and either zero or more total stable models,
  or zero or more partial stable models
- if the well-founded model is total, then “everything is clear” and it is the only stable model.
- if the well-founded model $\mathcal{W}(P) = (T, F)$ is partial, its T and F are “guaranteed”.
  Stable models deal with the atoms that are undefined in the well-founded model (= the intersection of all stable models).

  According to Lemma 11.2, each stable model is an extension of $\mathcal{W}_P = (T, F)$, i.e., for every 3-stable model (T', F'), $T' \supseteq T$ and $F' \supseteq F$.

Example: Win-Move Game

Consider again the small win-move game from Slides 649 and 657:

```
pos(a). pos(b). pos(c). pos(d).
move(a,b).  move(b,a).  move(b,c).  move(c,d).
win(X) :-  move(X,Y), not win(Y).
lose(X) :- pos(X), not win(X).
```

[Filename: Datalog/winmove-small.s]

The well-founded model has been derived on Slide 657:

$\mathcal{W}(P) = (\{\text{win(c)}, \text{lose(d)}\}, \{\text{lose(c)}, \text{win(d)}\})$ is partial, since win(a)/lose(a) and win(b)/lose(b) are undefined.

(Total) Stable Models:

$\mathcal{S}_1(P) = (\{\text{win(a)}, \text{win(c)}, \text{lose(b)}, \text{lose(d)}\}, \{\text{win(b)}, \text{win(d)}, \text{lose(a)}, \text{lose(c)}\})$ and
$\mathcal{S}_1(P) = (\{\text{win(b)}, \text{win(c)}, \text{lose(a)}, \text{lose(d)}\}, \{\text{win(a)}, \text{win(d)}, \text{lose(b)}, \text{lose(c)}\})$.

Here, $\mathcal{W}(P) = (\{\text{win(c)}\}, \{\text{win(d)}\})$ provides the "intended" application-specific interpretation of the result: c is won, d is lost, a and b are drawn.

From the stable models one can only conclude that it is possible to "fix" a to be won, and then b would also be lost (or vice versa).

Example: Choice between Alternatives

Consider again the program $P =$ "porq" from Slide 634 and 662:

```
q(a) :- not p(a).
p(a) :- not q(a).
```

[Filename: Datalog/porq.s]

Three stable models: `lparse -n 0 --partial pq.s|smodels`

- $\mathcal{W}(\mathcal{P}) = (\emptyset, \emptyset)$ is the well-founded model, and a partial stable model,
- $\{p(a)\}$ whose 3-valued representation is $(\{p(a)\}, \{q(a)\})$ is a total stable model, and
- $\{q(a)\}$ whose 3-valued representation is $(\{q(a)\}, \{p(a)\})$ is also a total stable model.

Depending on the application

- the well-founded model tells that nothing is known.
- the two stable models tell what are the two possibilities that have to be considered (in a more complex scenario, this would already exclude some alternatives)
- underspecification: the two stable models tell that the user is allowed to state additional facts (according to his preferences).

12.1 Answer Set Programming

Different idea than

- Stratified Datalog$^{\neg}$: Query Answering
- Prolog: “Declarative” Prolog Programming – the “search” is encoded into the SLD-resolution-tree.
- Rules as derivation: “if body holds, then derive head”.

Answer Set Programming: Specify a problem declaratively and leave the reasoning to the ASP solver (from [EIK09]):

- Rules as assertions: “if body holds, then head holds”.
- Allow disjunction and negation in the head.
- Possibility of modeling constraints;
- Reasoning with incomplete information; and
- Possibility of modeling preferences and priority.
- Spatial and temporal reasoning (here, the notorious Frame Problem is challenging).

SMODELS + LPARSE

- call `lparse -n 0 -d none [-partial]` *filename* | `smodels`
- smodels requires “...” for strings,
- smodels does not accept the “_” don’t-care underscore.
  Use X1, X2 etc. and constrain it by a domain predicate (see next slide).
- smodels does not accept decimal numbers – only (positive and negative) integers
  - decimals would not allow a grounding of the program.
    (integers are only little better ...)

  $\Rightarrow$ thus, mondial.P cannot be used.

Grounding the Program in smodels: Domain Predicates

(note: important for writing programs)

- Computation of stable models is based on grounding the program (cf. generating the reduct of a program on Slide 631)
- For grounding the rules (i.e., generate all relevant ground instances of each rule), smodels internally looks for *"domain predicates"* whose extension can be precomputed by a simple Datalog$^{\neg}$ subprogram of P without extended rules (cf. smodels Manual Sections 4.4.2 and 4.4.3):
  - union, intersection, join, set difference (with one negative literal!)
- every rule must contain at least one domain predicate atom.
- the ground instances of the domain predicates are precomputed and used for grounding. (cf. the example for the reduct of the win-move game – generate only ground instances of `win(X) :- move(X,Y), not win(Y)` s.t. `move(a,b)` is in the EDB).
- If lparse/smodels complains about

  ```
  <line_nr>: weakly restricted rule: <....>.
             weakly restricted variables: <var>
  ```

  introduce a domain predicate and use it in the rule (in the following: thing(X) with appropriate definition of thing).

Disjunction Revisited: Stable Models

```
mainDish(X) :- meal (X,Y).
drink(Y)    :- meal (X,Y).
vegetarian(X) :- mainDish(X), not nonvegetarian(X).      %% xor(veg, nonveg)
nonvegetarian(X) :- mainDish(X), not vegetarian(X).
porc(X) :- mainDish(X), nonvegetarian(X), not beef(X), not fish(X).  %% xor (porc,beef,fish)
beef(X) :- mainDish(X), nonvegetarian(X), not porc(X), not fish(X).
fish(X) :- mainDish(X), nonvegetarian(X), not porc(X), not beef(X).
whitewine(Y) :- meal(X,Y), fish(X).
whitewine(Y) :- meal(X,Y), porc(X).
redwine(Y) :- meal(X,Y), beef(X).
wine(Y) :- drink(Y), redwine(Y).                  %% wine = redwine u whitewine
wine(Y) :- drink(Y), whitewine(Y).
alcoholic(Y) :- drink(Y), wine(Y).
tomatojuice(Y) :- meal(X,Y), vegetarian(X).
meal(a,b).
nonvegetarian(a).
```

[Filename: Datalog/meals.s and meals.P]

- consider the question: is b alcoholic?
- meal/2, mainDish/1 and drink/1 act as domain predicates.

Example (Cont'd)

- Three (total) stable models: a is either porc, beef or fish. In either case, white/red wine is served with it, thus alcoholic(b) holds.
- Exercise: give the AFP computation.
- well-founded model:
  - $val_{\mathcal{W}(P)}(\text{fish}(b)) = val_{\mathcal{W}(P)}(\text{porc}(b)) = val_{\mathcal{W}(P)}(\text{beef}(b)) = u$,
  - $val_{\mathcal{W}(P)}(\text{alcoholic}(b)) = u$
- note: XSB yields a=fish, *not* computing the well-founded model.

(cf. also formal example from Slide 669)

Logical Rules: Local and Global Semantics

- Wrt. logical rules, the concept of disjunction does not exist:
  there are several *isolated* rules that *derive* wine(X)
  (in Description Logics, wine would be a union class Wine $=$ RedWine $\sqcup$ WhiteWine).
- The *global* semantics of the well-founded model and of stable models is always based on the *local* semantics of rules.
- any application-specific semantics (= interpretation of the model) has to be defined outside of the LP framework.
- Consider I agreeing with the well-founded model up to $val_I(\text{fish}(a)) = val_I(\text{porc}(a)) = val_I(\text{beef}(a)) = val_I(\text{whitewine}(b)) = val_I(\text{redwine}(b)) = u$, but then setting $val_I(\text{wine}(b)) = val_I(\text{alcoholic})(b) = t$:
  $I \models P$, but I is not stable!
- Recall Lemma 11.2: "$\mathcal{W}_P = (T, F)$ is the intersection of all 3-stable models of P".
  Above example: This may be smaller than the intersection of all 2-stable models!

Application-Specific Interpretation of LP Notions

- consider all *total* stable models and
  - "cautious reasoning": take facts that are true/false in all of them.
    (results are models of P, but not necessary stable ones)
    (meals example: alcoholic(b))
  - "credulous reasoning": take facts that are true/false in one of them
    (they are possible - somebody reporting them may tell the truth)
    (meals example: whitewine(b), redwine(b), but tomatojuice(b) is definitely false)
- preference by the user
  - interpreted as underspecification:
  - choose a "desired" fact a that is undefined and consider only models that satisfy it,
  - add a to the program and run again
    - without negation in the head/denials (cf. Slide 686):
      one or more new 3-stable models exist
- systematic weight (lparse/smodels support weighted clauses)

Disjunction in Rule Heads

(this syntax is buggy in smodels; see next page for (more expressive) alternative)

- rules of the form
  $A_1|\ldots|A_n$:- $B_1,\ldots,B_m$
  (recall: conjunction is expressed by n rules with the same body)
- stable models in presence of disjunction are not necessarily minimal
- check of minimality is again NP-complete
- smodels yields all stable models
- smodels, Section 4.8.1
- invoke with `--dlp`
- smodels returns "Error in input" (lparse without | `smodels` accepts it)

```
a | b :- c.
c :- not d.
d :- not c.
```

(Total) stable models:

- $\{d\}$, $\{c,a\}$, $\{c,b\}$, $\{c,a,b\}$.

[Filename: Datalog/disj.s]

SMODELS: DISJUNCTION IN THE HEAD VIA CHOICE RULES

- extended rule head: k {a$_1$, ..., a$_n$} m :- *body*
- k to m atoms of the head must be true if *body* is true.

```
mainDish(X) :- meal (X,Y).
drink(Y) :- meal (X,Y).
1{vegetarian(X),nonvegetarian(X)}1 :- meal(X,Y).
1{porc(X), beef(X), fish(X)}1 :- mainDish(X), nonvegetarian(X).
whitewine(Y) :- meal(X,Y), fish(X).
whitewine(Y) :- meal(X,Y), porc(X).
redwine(Y) :- meal(X,Y), beef(X).
wine(Y) :- drink(Y), redwine(Y).
wine(Y) :- drink(Y), whitewine(Y).
alcoholic(Y) :- drink(Y), wine(Y).
tomatojuice(Y) :- meal(X,Y), vegetarian(X).
meal(a,b).
nonvegetarian(a).
```

[Filename: Datalog/meals-disj.s]

Stable Models – Example

Consider the following program (cf. Slide 635):

```
1{p(a), q(a)}1.
p(a).
```

[Filename: Datalog/pporq-choice.s]

- The program has only one *total* stable model: $\{p(a)\}$.
  (which corresponds to the 3-valued $(\{p(a)\}, \{q(a)\}$.)
- The 3-valued interpretation $(\{p(a)\}, \emptyset)$ (`Stable Model: p(a) p'(a) q'(a)`) is also considered a (partial) stable model (!) [Bug or not bug?]

EXPLICIT NEGATION

- well-founded and stable semantics are still based on default negation.
  non-monotonic: adding some positive atom is always allowed and may make before conclusions invalid (and add others).
- explicit negation:
  - $\neg p(a_1, \ldots, a_n)$ as negative facts,
  - $\neg p(X_1, \ldots, X_n)$:- ... in rule heads.
- Explicit negation is monotonic. "Additional" positive information wrt. such atoms would make the program inconsistent.
- applications:
  - in diagnosis systems to explicitly derive negative knowledge,
  - for expressing integrity constraints.
- XSB: WSFX package, XSB Manual Part II.
- smodels: only integrity constraints via *denials* (see next slide).

EXPLICIT NEGATION VIA DENIALS

A *denial* is a constraint that forbids certain database states
(DB: integrity constraints can be formulated as denials)

- Consider a "rule"
  :- p.
  Its semantics is "if p is true, then the empty clause is true", i.e., then, "false" is true.
- means "p must not be true".

```
:- p(a).
q(a) :- not p(a).
p(a) :- not q(a).
```

[Filename: Datalog/porq-denial.s]

- the program has only one stable model, $\{q(a)\}$. It is total.
- note: XSB does not care for such rules (they would be useless there since XSB computes only one model).

Example: Italians Revisited

The italian-vs-english ontology from Slide 546 can be specified via choice rules (and implicit) denials (line 1: either italian or english allowed):

```
0{ italian(X), english(X) }1 :- person(X),      thing(X).
person(X) :- italian(X),                        thing(X).
person(X) :- english(X),                        thing(X).
1{ lazy(X), latinlover(X) }1 :- italian(X),     thing(X).
italian(X) :- lazy(X),                          thing(X).
italian(X) :- latinlover(X),                    thing(X).
0{ gentleman(X), hooligan(X) }2 :- english(X),  thing(X).
english(X) :- gentleman(X),                     thing(X).
english(X) :- hooligan(X),                      thing(X).
gentleman(X) :- latinlover(X),                  thing(X).
italian(e).
thing(e).
```

[Filename: Datalog/italians-english.s]

- do not query for `?- lazy(e)` , but inspect the stable model(s).
- there is a single (total) stable model where `lazy(e)` holds.

SMODELS: MULTIPLE MODELS

- Motivated by the “Ascending, Descending” graphics by M.C.Escher
  `http://en.wikipedia.org/wiki/Ascending_and_Descending`

```
corner(1..4).     % set of corners
higher(1,2).
1{ higher(1,2), higher(1,3), higher(1,4) }1.
1{ higher(2,1), higher(2,3), higher(2,4) }1.
1{ higher(3,1), higher(3,2), higher(3,4) }1.
1{ higher(4,1), higher(4,2), higher(4,3) }1.
:- higher(X,X), corner(X).   % irreflexive
:- higher(X,Y), higher(Y,X), corner(X), corner(Y).  % asymmetric
 % inverse functional
:- higher(Y,X), higher(Z,X), corner(X), corner(Y), corner(Z), Y != Z.
% :- higher(X,Y), higher(X,Z), corner(X), corner(Y), corner(Z), Y != Z.
```

[Filename: Datalog/escherstairs.s]

- two possibilities = two models
- recall Semantic Web: only answers what holds in *all* models

SMODELS: PLANNING

Give a specification P of the workflow including constraints:

- if at least one total stable model S exists, the specification including the constraints is satisfiable.
  S describes the plan that must be followed.
- if several total stable model exist, each one represents a possible execution.
  Different plans mean that choicepoints during the execution exist.
  - they can be decided a priori: add the intended atoms to P:
    - there is at least one stable model, but maybe still several ones.
  - ... or decide them during execution of the workflow (e.g. to be able to react upon external influences).

Planning Example: The Farmer's Puzzle

A farmer, travelling to the market with his dog, a goat, and a cabbage. He has to cross a small river, where a boat can be used.

- When using the boat, he can transport only one item.
- He can cross the river as often as he wants.
- When the dog stays on the same side as the goat, and the farmer is not there, the dog will kill and eat the goat.
- When the goat stays on the same side as the cabbage, and the farmer is not there, the goat will eat the cabbage.

Is it possible for the farmer to bring all items to the other side? If yes, how?

Example (Cont'd)

```
state(1..8).     % estimate the number of necessary states.
side(l).  side(r).
is(farmer,l,1).     % farmer is on the left side in state 1 (with all items)
thing(cabbage).  thing(goat).  thing(dog).
is(X,l,1) :- thing(X).
otherside(X,Y) :- side(X), side(Y), X != Y.
0{ transport(cabbage,N), transport(goat,N), transport(dog,N) }1 :- state(N), not finished(N).
:- transport(X,N), is(farmer,S1,N), is(X,S2,N), S1 != S2,
   thing(X), state(N), side(S1), side(S2).
is(X,S2,M) :- is(X,S1,N), thing(X), otherside(S1,S2), transport(X,N), M = N+1, not finished(N)
   state(N), side(S1).
is(X,S1,M) :- is(X,S1,N), thing(X), not transport(X,N), M = N+1, not finished(N),
   state(N), side(S1).     %% the "frame axiom"
is(farmer,S2,M) :- is(farmer,S1,N), otherside(S1,S2), M = N+1, not finished(N),
   state(N), side(S1).
:- is(cabbage,S,N), is(goat,S,N), not is(farmer,S,N),   state(N), side(S).
:- is(goat,S,N), is(dog,S,N), not is(farmer,S,N),       state(N), side(S).
finished(N) :- is(cabbage,r,N), is(goat,r,N), is(dog,r,N),        state(N).
finished(M) :- finished(N), M = N+1,                              state(N).
:- not finished(8).
```

[Filename: Datalog/farmer.s]

Example (Cont'd)

Two stable models:

1. first carry the goat to the other side (r)
   (it must not be left with the dog or with the cabbage).
2. go back,
3. bring either the dog or the cabbage to the other side,
4. go back with the goat,
5. bring the cabbage or the dog (one is still there) to the other side,
6. go back,
7. take the goat and bring it to the right bank again,
8. continue traveling to the market.

$\Rightarrow$ Step 3 is a choicepoint.

SMODELS: SUDOKU SOLVER

```
% example sudoku content from (german) wikipedia
p(2,9,3).  p(4,8,1).  p(5,8,9).  p(6,8,5).  p(3,7,8).  p(8,7,6).
p(1,6,8).  p(5,6,6).  p(1,5,4).  p(4,5,8).  p(9,5,1).  p(5,4,2).  p(2,3,6).
p(7,3,2).  p(8,3,8).  p(4,2,4).  p(5,2,1).  p(6,2,9).  p(9,2,5).  p(8,1,7).

% general sudoku rules (x = cols, y = rows)
col(1..9).      row(1..9).      num(1..9).
% samesquare expresses the 3x3 subsquares:
samesq(1,2).    samesq(1,3).   samesq(2,3).
samesq(4,5).    samesq(4,6).   samesq(5,6).
samesq(7,8).    samesq(7,9).   samesq(8,9).
samesq(B,A) :-  samesq(A,B), num(A), num(B).

1{p(X,Y,1),p(X,Y,2),p(X,Y,3),p(X,Y,4),p(X,Y,5),p(X,Y,6),p(X,Y,7),p(X,Y,8),p(X,Y,9)}1
   :- col(Y), row(X).

:- p(X,Y1,N), p(X,Y2,N), col(X), row(Y1), row(Y2), num(N), Y1!=Y2.
:- p(X1,Y,N), p(X2,Y,N), col(X1), col(X2), row(Y), num(N), X1!=X2.
:- p(X1,Y1,N), p(X2,Y2,N), col(X1), col(X2), row(Y1), row(Y2),num(N),
        samesq(X1,X2), samesq(Y1,Y2),  X1 + 10 * Y1 != X2 + 10 * Y2.
```

[Filename: Datalog/sudoku.s]

(cf. Sudoku from Slide 584)

Aside: Another Sudoku

- for most strategy-based solvers it is “unsolvable”, requires “trial and error” (which is actually backtracking)

	A	B	C	D	E	F	G	H	I
9	6					2			8
8			7					3	
7		4		1			9		
6		8		9					2
5						7		6	
4			3		5				
3			1			3			4
2		5						7	
1	9				6				

Aside: Code for another Sudoku - easier encoding of input

```
% another sudoku which is unsolvable by rule-based reasoners:
r(9,  6,0,0,0,0,2,0,0,8).
r(8,  0,0,7,0,0,0,0,3,0).
r(7,  0,4,0,1,0,0,9,0,0).
r(6,  0,8,0,9,0,0,0,0,2).
r(5,  0,0,0,0,0,7,0,6,0).
r(4,  0,0,3,0,5,0,0,0,0).
r(3,  0,0,1,0,0,3,0,0,4).
r(2,  0,5,0,0,0,0,0,7,0).
r(1,  9,0,0,0,6,0,0,0,0).

col(1..9).   row(1..9).   num(1..9).  numx(0..9).
```

[Filename: Datalog/sudoku2.s]

```
9{q(1,Y,X1), q(2,Y,X2), q(3,Y,X3), q(4,Y,X4), q(5,Y,X5),
                q(6,Y,X6), q(7,Y,X7), q(8,Y,X8), q(9,Y,X9)}9
  :- r(Y,X1,X2,X3,X4,X5,X6,X7,X8,X9), row(Y),
     numx(X1), numx(X2), numx(X3), numx(X4), numx(X5), numx(X6), numx(X7), numx(X8), numx(X9).
p(X,Y,N) :- q(X,Y,N), col(X), row(Y), num(N).    %%  0s are unknown cells

% general sudoku rules as before ...
% samesquare expresses the 3x3 subsquares:
samesq(1,2).    samesq(1,3).   samesq(2,3).
samesq(4,5).    samesq(4,6).   samesq(5,6).
samesq(7,8).    samesq(7,9).   samesq(8,9).
samesq(B,A) :-  samesq(A,B), num(A), num(B).
```

695

COMMENTS AND LINKS ON ASP AND SUDOKU SOLVING

- None of the rules is “constructive” in the sense of “position (x, y) must be n_1 if ...”, or “position (x, y) must be n_1 of n_2 if ...”
- the description strongly relies on the denials and the solver must find out what is not forbidden.
- Sudoku is a typical *Constraint Satisfaction Problem*; ASP is a certain form of Constraint Solving.
  See e.g. `https://en.wikipedia.org/wiki/Constraint_satisfaction_problem`.
- Sudoku solved by an explicit Constraint Propagation algorithm in python:
  `http://norvig.com/sudoku.html` by Peter Norvig, a well-known AI scientist.

SMODELS: FURTHER EXAMPLES AND PUZZLES

- the smodels documentation (Section 6.1) contains
  - the Graph 3-Coloring problem (sudoku can also be encoded as graph coloring),
  - some logical puzzles.
- Exercise: encode the Fish Puzzle (see Web page) in ASP and let smodels solve it.

"PROVE" $P \models \varphi$

- Stable models:
  - definition (cf. Slide 659) is based on grounding P with the active domain.

  $\Rightarrow$ Problem solving for concrete cases – answer queries, look for ground atoms.
- It is only possible to show that $\mathcal{S} \models \varphi$ for all *stable* models $\mathcal{S}$ of P:
- Show $P \models_s \varphi$ (1st Alternative):
  - generate all stable models $\mathcal{S}_1, \ldots, \mathcal{S}_n$.
  - for each of them check whether $\mathcal{S}_i \models \varphi$ (if not, it is a witness for a counterexample)
- Show $P \models_s \varphi$ (2nd Alternative):
  - encode $\neg\varphi$ in a program P',
  - run smodels on $P \cup P'$,
  - if there is a stable model, then, $P \not\models_s \varphi$
    (and, again, a witness for a counterexample has been found).

EXAMPLE: PARRICIDES IN GREEK MYTHODOLOGY

(from ESWC'07 SPARQL tutorial by Marcelo Arenas et al)

A parricide is a person who killed his/her father. [Filename: Datalog/parricide.s]

```
1{ parricide(X), nonParricide(X) }1 :- person(X).
person(iokaste).  nonParricide(iokaste).
hasChild(iokaste, oedipus).
person(oedipus).  parricide(oedipus).
marriedto(oedipus, iokaste).    %% irrelevant, but the reason why he became prominent
hasChild(oedipus, perineikes).
person(perineikes).     %% unknown whether parricide or not.
hasChild(perineikes, thesandros).
person(thesandros). nonParricide(thesandros).
parentOfParricide(X) :- hasChild(X,Y), parricide(Y).
parentOfNonParricide(X) :- hasChild(X,Y), nonParricide(Y).
parentOfParricideGrandparentOfNonParricide(X) :-
   parentOfParricide(X), hasChild(X,Y), parentOfNonParricide(Y).
existsPoPGoNP :- parentOfParricideGrandparentOfNonParricide(X), person(X).  %% => t/f
```

Does $P \models_S \exists X$: *parentOfParricideGrandparentOfNonParricide(X)*?

- two possibilities: either Iokaste or Oedipus.

FOL Entailment/Proofs vs. Stable Models/ASP

Consider again the italian-vs-english ontology from Slides 546 and 687:

- FOL/Tableaux: prove $\text{Spec}_{\text{ItalEngl}} \models \forall x : \text{italian}(x) \rightarrow \text{lazy}(x)$
  (without considering any ground instance)
- ASP: show that in all stable models of `italians-english.s`, $\text{lazy}(e)$ holds.
- this conclusion is weaker than the one for FOL.
  - e is a "typical" instance of an italian, but
  - recall that Stable Models reasoning is nonmonotonic – more knowledge about e could invalidate this conclusion.

Monotonic vs. Nonmonotonic Reasoning

- FOL:
  $(\forall x : \text{bird}(x) \wedge \neg\text{penguin}(x) \rightarrow \text{flies}(x)) \wedge \text{bird}(\text{tweety}) \not\models \text{flies}(\text{tweety})$
  $(\forall x : \text{bird}(x) \wedge \neg\text{penguin}(x) \rightarrow \text{flies}(x)) \wedge \text{bird}(\text{tweety}) \not\models \neg\text{flies}(\text{tweety})$
  FOL reasoning does not entail any anything about tweety – Open World.
- $(\forall x : \text{bird}(x) \wedge \neg\text{penguin}(x) \rightarrow \text{flies}(x)) \wedge \text{bird}(\text{tweety}) \wedge \text{penguin}(\text{tweety}) \models \neg\text{flies}(\text{tweety})$
  $(\forall x : \text{bird}(x) \wedge \neg\text{penguin}(x) \rightarrow \text{flies}(x)) \wedge \text{bird}(\text{tweety}) \wedge \neg\text{penguin}(\text{tweety}) \models \text{flies}(\text{tweety})$
- FOL is monotonic. More knowledge, more conclusions, no conclusions can/must ever be withdrawn.
- it is not possible to *conclude* things that cannot be actually proven (and that may have to be withdrawn later).

Monotonicity vs. Nonmonotonicity (cont'd)

- ASP:

```
flies(X) :- bird(X), not penguin(X).
bird(tweety).
```
[Filename: Datalog/tweety.s]

```
Stable Model: bird(tweety) flies(tweety) bird'(tweety) flies'(tweety)
```

```
flies(X) :- bird(X), not penguin(X).
bird(tweety).
penguin(tweety).
```
[Filename: Datalog/tweety2.s]

```
Stable Model: bird(tweety) penguin(tweety) bird'(tweety) penguin'(tweety)
```


Monotonicity vs. Nonmonotonicity (cont'd)

This can be used to encode general concepts in nonmonotonic reasoning

- by default, if $p(x)$ holds, and there is no further information, then usually $q(x)$ can be assumed.
- if $p(x)$ holds, and $q(x)$ is consistent with the knowledge, conclude $q(x)$:
- "Circumscription" (J. McCarthy, "Circumscription – A form of non-monotonic reasoning". Artificial Intelligence 13: 27-39, April 1980).

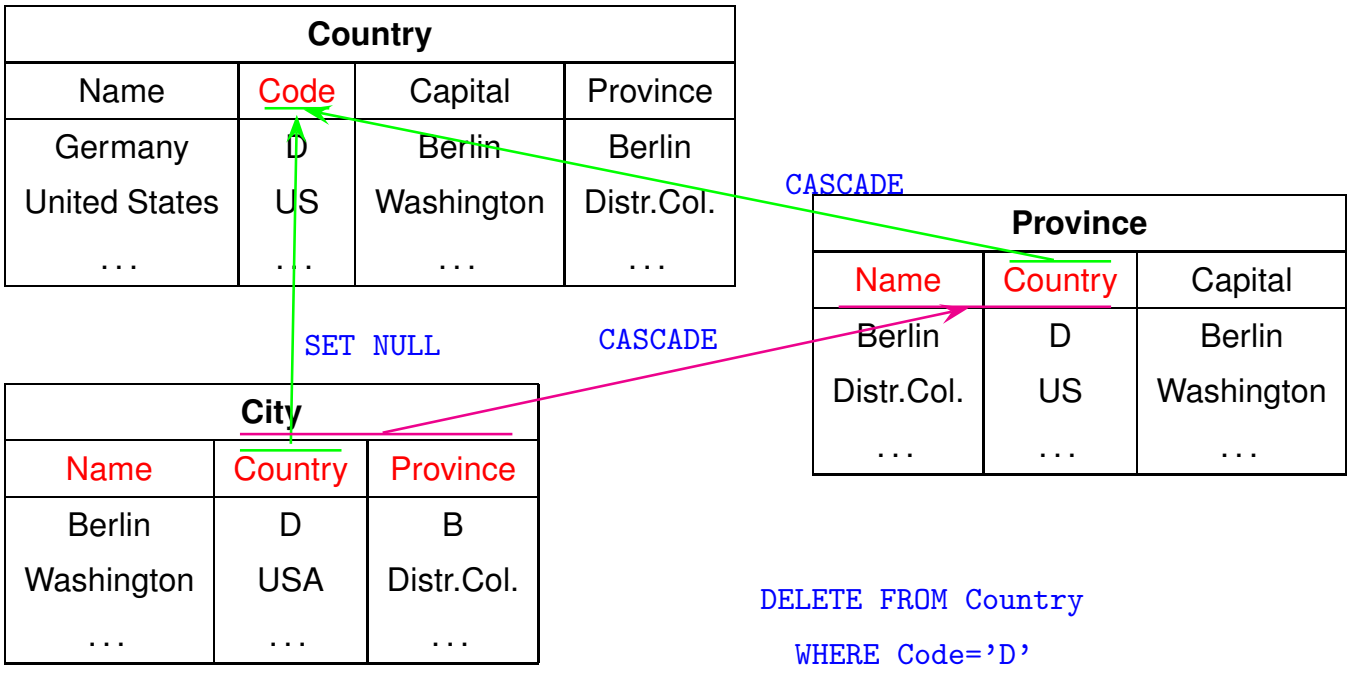
```
flies(X) :- bird(X), not abnormal(X).
abnormal(X) :- penguin(X).
bird(tweety).
```
[Filename: Datalog/tweety-circ.s]

```
Stable Model: bird(tweety) flies(tweety) bird'(tweety) flies'(tweety)
```

- In (T, F)-notation: ({bird(tweety), flies(tweety)}, {penguin(tweety), abnormal(tweety)})
- Note that ({bird(tweety), penguin(tweety), abnormal(tweety)}, {flies(tweety) }) is also a model of P, but it is not stable.
  If learning that penguin(tweety) holds, this model would become (the only) stable one.

12.2 An Application: Semantics of Referential Actions in SQL

Consider again the problem of ambiguous semantics of referential actions from Slide 236:



SQL Standard

- The SQL Standard gives a hard-to-understand *procedural* specification of referential actions.
- Database Systems implemented only ON DELETE CASCADE (as optional alternative to ON DELETE NO ACTION) for a long time (late 90s)
- Nondeclarative semantics of ON DELETE SET NULL or ON UPDATE CASCADE are implemented.
  e.g. Oracle 11: the most recently defined (more exactly: activated) referential action is executed first.
- combination with transactions and PL/SQL and triggers becomes complex.

Concepts

- Intuitive concept of "Event-Condition-Action-Rules" ("ECA Rules"):
  ON DELETE (of referenced tuple) CASCADE (update to referencing tuple(s)
- can be read as a declarative specification how integrity is to be maintained:
  "Whenever the set U of updates includes the deletion of a referenced tuple wrt. a referential integrity constraint $S.\bar{B} \rightarrow R.\bar{A}$, the cascaded update of the referencing tuple(s) must also be contained in U. (etc. for other referential actions)"

$\Rightarrow$ Set-oriented characterization of all updates that "complete" a transaction wrt. referential integrity maintenance.

... from a declarative point of view, the semantics should be easy, and it *must* be unambiguous. This lead us to playing around with Datalog (and Statelog) around Easter 1996:

Publications

The following papers are accessible via `http://dblp.uni-trier.de/` (and partially via the DBIS Publications Web pages):

- B. Ludäscher, W. May und J. Reinert. Towards a Logical Semantics for Referential Actions in SQL. In *Proc. Intl. Workshop on Foundations of Models and Languages for Data and Objects: Integrity in Databases (FMLDO'96)*, Dagstuhl Castle, Germany, 1996.
- B. Ludäscher, W. May und G. Lausen. Referential Actions as Logical Rules. In *Proc. ACM Symposium on Principles of Database Systems (PODS'97)*, pp. 217–224, 1997.
- B. Ludäscher und W. May. Referential Actions: From Logical Semantics to Implementation. In *Proc. Intl. Conference on Extending Database Technology (EDBT'98)*, Springer LNCS 1377, pp. 404-418, 1998.
- W. May und B. Ludäscher. Understanding the Global Semantics of Referential Actions using Logic Rules. In *ACM Transactions on Database Systems (TODS)*, 27(4):343–397, 2002.

12.2.1 Straightforward Logical Semantics: Encoding as Rules

- consider only ON DELETE CASCADE/SET NULL:

```
% prov(Country) refs country(code) on delete cascade
del_province(PN,C,PPop,PA,PCap,PCapProv) :- del_country(N,C,Cap,CapP,A,P),
    province(PN,C,PPop,PA,PCap,PCapProv).
% city(Country) refs country(code) on delete set null
upd_city(CN,C,CP,CPop,Lat,Long,El,CN,null,CP,CPop,Lat,Long,El) :-
    del_country(N,C,Cap,CapP,A,P), city(CN,C,CP,CPop,Lat,Long,El).
% city(Country,Province) refs province(name,country) on delete cascade
del_city(CN,C,PN,CPop,Lat,Long,El) :-  del_province(PN,C,PPop,PA,PCap,PCapProv),
    city(CN,C,PN,CPop,Lat,Long,El).
inconsistent :-  del_city(CN,C,CP,CPop,Lat,Long,El),
                 upd_city(CN,C,CP,CPop,Lat,Long,El,CN2,C2,CP2,CPop2,Lat2,Long2,El2).
country("Germany","D","Berlin","Berlin",356910,83536115).
province("Berlin","D",3472009,889,"Berlin","Berlin").
city("Berlin","D","Berlin",3472009,13,52,null).
del_country("Germany","D","Berlin","Berlin",356910,83536115).
% ?- inconsistent.
```

[Filename: Datalog/refint.s]

- cascaded updates and “inconsistent” are true.

Deletions: Encoding as Rules

- ON DELETE/UPDATE NO ACTION:
  - if there is a referencing tuple that is not deleted (or modified to reference another parent) in the same transaction, then the update is not allowed, e.g., in the reference Organization(City,Country,Province) $\rightarrow$ City(City,Country,Province) ON DELETE NO ACTION ON UPDATE CASCADE
    - a city where an organization has its headquarter cannot be deleted (i.e., when the city is merged with another one, the value must also be changed in the referenced organization tuple in the same transaction),
    - if a city where an organization has its headquarter is renamed or its province changes, then, the update is cascaded to the headquarter foreign key.

$\Rightarrow$ all potential (cascaded) updates during the transaction must be considered.

- `ext_ACTION` (“external”): the updates issued by the user/by the program.
- `pot_ACTION` (“potential”): all updates issued by the user/by the program or resulting from these by any referential action.

12.2.2 Deletions: Encoding as Rules

- Collect all potential updates that are triggered by the external operations:
- Consider only deletions and ON DELETE CASCADE/NO ACTION.

```
pot_del_country(N,C,Cap,CapP,A,Pop) :- ext_del_country(N,C,Cap,CapP,A,Pop),
        country(N,C,Cap,CapP,A,Pop).
% Province(Country) refs Country(code) on delete cascade
pot_del_province(P,C,PPop,PA,PCap,PCapProv) :- pot_del_country(N,C,Cap,CapP,A,Pop),
        province(P,C,PPop,PA,PCap,PCapProv).
% City(Country) refs Country(code) on delete cascade
pot_del_city(CN,C,P,CPop,Lat,Long,El) :-
    pot_del_country(N,C,Cap,CapP,A,Pop),
    city(CN,C,P,CPop,Lat,Long,El).
% City(Country,Province) refs Province(Country,name) on delete cascade
pot_del_city(CN,C,P,CPop,Lat,Long,El) :-
    pot_del_province(P,C,PPop,PA,PCap,PCapProv),
    city(CN,C,P,CPop,Lat,Long,El).
```

[Filename: Datalog/refint1.s]

Deletions: Encoding as Rules

```
pot_del_organization(O,N,Ci,Co,P,E) :- ext_del_organization(O,N,Ci,Co,P,E),
        organization(O,N,Ci,Co,P,E).
% refs from isMember to Country and Organization: ON DELETE CASCADE.
pot_del_isMember(C,O,T) :- pot_del_country(N,C,Cap,CapP,A,Pop),
        isMember(C,O,T).
pot_del_isMember(C,O,T) :- pot_del_organization(O,N,Ci,Co,P,E),
        isMember(C,O,T).
```

[Filename: Datalog/refint2.s]

Deletions: Encoding as Rules

- Organization(City,Country,Province) → City(City,Country,Province) ON DELETE NO ACTION ON UPDATE CASCADE
- block deletions/updates if ON UPDATE NO ACTION and child tuple remains,
- propagate blocking upwards through CASCADEs.

```
blk_del_city(Ci,Co,P,CPop,Lat,Long,El) :- pot_del_city(Ci,Co,P,CPop,Lat,Long,El),
  organization(O,N,Ci,Co,P,E), rem_organization(O,N,Ci,Co,P,E).
rem_organization(O,N,Ci,Co,P,E) :- organization(O,N,Ci,Co,P,E),
  not del_organization(O,N,Ci,Co,P,E).            %%% del not yet defined
blk_del_province(P,C,PPop,PA,PCap,PCapProv) :- pot_del_city(CN,C,P,CPop,Lat,Long,El),
  blk_del_city(CN,C,P,CPop,Lat,Long,El), pot_del_province(P,C,PPop,PA,PCap,PCapProv).
blk_del_country(N,C,Cap,CapP,A,Pop) :- pot_del_province(P,C,PPop,PA,PCap,PCapProv),
  blk_del_province(P,C,PPop,PA,PCap,PCapProv), pot_del_country(N,C,Cap,CapP,A,Pop).
```

[Filename: Datalog/refint3.s]

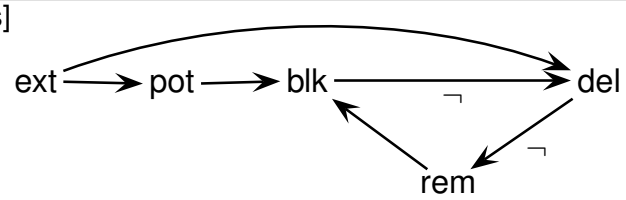
Deletions: Encoding as Rules

- (if any update is blocked do nothing).
- execute (and appropriately cascade) all external updates that are not blocked.

```
del_country(N,C,Cap,CapP,A,Pop) :- ext_del_country(N,C,Cap,CapP,A,Pop),
    country(N,C,Cap,CapP,A,Pop), not blk_del_country(N,C,Cap,CapP,A,Pop).
del_province(P,C,PPop,PA,PCap,PCapProv) :- del_country(N,C,Cap,CapP,A,Pop),
    province(P,C,PPop,PA,PCap,PCapProv).
del_city(CN,C,P,CPop,Lat,Long,El) :- del_country(N,C,Cap,CapP,A,Pop),
    city(CN,C,P,CPop,Lat,Long,El).
del_city(CN,C,P,CPop,Lat,Long,El) :- del_province(P,C,PPop,PA,PCap,PCapProv),
    city(CN,C,P,CPop,Lat,Long,El).
del_isMember(C,O,T) :- del_country(N,C,Cap,CapP,A,P), isMember(C,O,T).
del_organization(O,N,Ci,Co,P,E) :- ext_del_organization(O,N,Ci,Co,P,E),
    organization(O,N,Ci,Co,P,E), not blk_del_organization(O,N,Ci,Co,P,E).
del_isMember(C,O,T) :- del_organization(O,N,Ci,Co,P,E), isMember(C,O,T).
```

[Filename: Datalog/refint4.s]

- Dependency graph:



Deletions Example: delete Germany (not referenced)

```
country("Germany","D","Berlin","Berlin",356910,83536115).
province("Berlin","D",3472009,889,"Berlin","Berlin").
city("Berlin","D","Berlin",3472009,13,52,null).
organization("EU","European Union","Brussels","B","Brabant","1992-02-07").
isMember("D","EU","member").
ext_del_country("Germany","D","Berlin","Berlin",356910,83536115).
% lparse -n 0 refint1.s refint2.s refint3.s refint4.s refint-del-d.s| smodels
```

[Filename: Datalog/refint-del-d.s]

In this case, there is only one stable model (i.e., it coincides with the well-founded model) which is total:

```
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_province("Berlin","D",3472009,889,"Berlin","Berlin")
pot_del_city("Berlin","D","Berlin",3472009,13,52,null)
pot_del_isMember("D","EU","member")
del_country("Germany","D","Berlin","Berlin",356910,83536115)
del_province("Berlin","D",3472009,889,"Berlin","Berlin")
del_city("Berlin","D","Berlin",3472009,13,52,null)
del_isMember("D","EU","member")
rem_organization("EU","European Union","Brussels","B","Brabant","1992-02-07")
```


Deletions Example - delete Belgium (referenced by the EU)

```
country("Belgium","B","Brussels","Brabant",30510,10170241).
city("Brussels","B","Brabant",951580,null,null,null).
province("Brabant","B",2253794,3358,"Brussels","Brabant").
organization("EU","European Union","Brussels","B","Brabant","1992-02-07").
isMember("B","EU","member").
ext_del_country("Belgium","B","Brussels","Brabant",30510,10170241).
% lparse -n 0 refint1.s refint2.s refint3.s refint4.s refint-del-b1.s| smodels
```

[Filename: Datalog/refint-del-b1.s]

Again, a total unique stable model = well-founded model – it contains blockings:

```
ext_del_country("Belgium","B","Brussels","Brabant",30510,10170241)
pot_del_country("Belgium","B","Brussels","Brabant",30510,10170241)
pot_del_isMember("B","EU","member")
pot_del_province("Brabant","B",2253794,3358,"Brussels","Brabant")
pot_del_city("Brussels","B","Brabant",951580,null,null,null)
rem_organization("EU","European Union","Brussels","B","Brabant","1992-02-07")   <<<<
blk_del_city("Brussels","B","Brabant",951580,null,null,null)                    <<<<
blk_del_province("Brabant","B",2253794,3358,"Brussels","Brabant")               <<<<
blk_del_country("Belgium","B","Brussels","Brabant",30510,10170241)              <<<<
```

Deletions Example - delete Belgium and the EU

```
ext_del_country("Belgium","B","Brussels","Brabant",30510,10170241).
ext_del_organization("EU","European Union","Brussels","B","Brabant","1992-02-07").
country("Belgium","B","Brussels","Brabant",30510,10170241).
city("Brussels","B","Brabant",951580,null,null,null).
province("Brabant","B",2253794,3358,"Brussels","Brabant").
organization("EU","European Union","Brussels","B","Brabant","1992-02-07").
isMember("B","EU","member").
% lparse -n 0 refint1.s refint2.s refint3.s refint4.s refint-del-b2.s| smodels
```

[Filename: Datalog/refint-del-b2.s]

Again, a total unique stable model = well-founded model:

```
ext_del_country("Belgium","B","Brussels","Brabant",30510,10170241)
ext_del_organization("EU","European Union","Brussels","B","Brabant","1992-02-07")
pot_del_ [...]
del_country("Belgium","B","Brussels","Brabant",30510,10170241)
del_city("Brussels","B","Brabant",951580,null,null,null)
del_province("Brabant","B",2253794,3358,"Brussels","Brabant")
del_isMember("B","EU","member")
del_organization("EU","European Union","Brussels","B","Brabant","1992-02-07")
```


Deletions Example with NO ACTION

- Change the reference from City to Province from CASCADE to NO ACTION:
  (in refint1.s and refint4.s)

```
pot_del_country(N,C,Cap,CapP,A,Pop) :- ext_del_country(N,C,Cap,CapP,A,Pop),
        country(N,C,Cap,CapP,A,Pop).
% Province(Country) refs Country(code) on delete cascade
pot_del_province(P,C,PPop,PA,PCap,PCapProv) :- pot_del_country(N,C,Cap,CapP,A,Pop),
        province(P,C,PPop,PA,PCap,PCapProv).
% City(Country) refs Country(code) on delete cascade
pot_del_city(CN,C,P,CPop,Lat,Long,El) :-
    pot_del_country(N,C,Cap,CapP,A,Pop),
    city(CN,C,P,CPop,Lat,Long,El).
% City(Country,Province) refs Province(name,country) on delete no action <<<<<<<<
blk_del_province(P,C,PPop,PA,PCap,PCapProv) :-
  pot_del_province(P,C,PPop,PA,PCap,PCapProv),
  city(CN,C,P,CPop,Lat,Long,El), rem_city(CN,C,P,CPop,Lat,Long,El).
rem_city(CN,C,P,CPop,Lat,Long,El) :- city(CN,C,P,CPop,Lat,Long,El),
                                not del_city(CN,C,P,CPop,Lat,Long,El).
```

[Filename: Datalog/refint1b.s]

Deletions: Encoding as Rules

```
del_country(N,C,Cap,CapP,A,Pop) :- ext_del_country(N,C,Cap,CapP,A,Pop),
    country(N,C,Cap,CapP,A,Pop), not blk_del_country(N,C,Cap,CapP,A,Pop).
del_province(P,C,PPop,PA,PCap,PCapProv) :- pot_del_country(N,C,Cap,CapP,A,Pop),
    del_country(N,C,Cap,CapP,A,Pop), province(P,C,PPop,PA,PCap,PCapProv).
del_city(CN,C,P,CPop,Lat,Long,El) :- pot_del_country(N,C,Cap,CapP,A,Pop),
    del_country(N,C,Cap,CapP,A,Pop), city(CN,C,P,CPop,Lat,Long,El).
del_isMember(C,O,T) :- pot_del_country(N,C,Cap,CapP,A,P),
    del_country(N,C,Cap,CapP,A,P), isMember(C,O,T).
del_organization(O,N,Ci,Co,P,E) :- ext_del_organization(O,N,Ci,Co,P,E),
    organization(O,N,Ci,Co,P,E), not blk_del_organization(O,N,Ci,Co,P,E).
del_isMember(C,O,T) :- del_organization(O,N,Ci,Co,P,E), isMember(C,O,T).
```

[Filename: Datalog/refint4b.s]

- using pot_del is sometimes redundant, and only required for providing smodels with a domain predicate.

Deletions: The Resulting Models

```
% lparse -n 0 refint1b.s refint2.s refint3.s refint4b.s refint-del-d.s| smodels
```

- 3 stable models, 2 of them total, one partial (= well-founded model).

```
ext_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_province("Berlin","D",3472009,889,"Berlin","Berlin")
pot_del_city("Berlin","D","Berlin",3472009,13,52,null)
pot_del_isMember("D","EU","member")
del_province("Berlin","D",3472009,889,"Berlin","Berlin")
del_country("Germany","D","Berlin","Berlin",356910,83536115)
del_city("Berlin","D","Berlin",3472009,13,52,null)
del_isMember("D","EU","member")
```

```
ext_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_country("Germany","D","Berlin","Berlin",356910,83536115)
pot_del_[...]
blk_del_country("Germany","D","Berlin","Berlin",356910,83536115)
blk_del_province("Berlin","D",3472009,889,"Berlin","Berlin")
rem_city("Berlin","D","Berlin",3472009,13,52,null)
```

- in the partial stable model (=the well-founded model), all `pot_del` are true, all `blk_del` and all `del` are undefined.

Deletions: Interpreting the Resulting Models

3 stable models:

1. the well-founded, partial one:
   all `pot_del` are true, the `blk_del` and `del` are undefined.

- two total stable ones:
  2. all `pot_del` are true, the `del` are true, and the `blk_del` are false (since the reason for `blk_del_prov(...)` is deleted).
  3. all `pot_del` are true, the `blk_del` are true, and the `del` are false (since the deletion of the province is not allowed due to blocking).

Application-Specific Priorities (cf. EDBT paper)

- the "intended" one is the stable model (2) that gives priority to deletions against blockings.
- including game-theoretic interpretation:
  - "del_p($\bar{x}$)" is won if deletion is possible
  - counter-moves: "how?" and "what about this referencing tuple q($\bar{y}$)",
  - justification "cascaded from "del_r($\bar{z}$)" and "claim: del_q($\bar{y}$)",
  - infinite games (cycling around via deleted blocking tuples) are won for the deleter.
  - the well-founded model can be used $\rightarrow$ polynomial.

Deletions: Encoding as Rules

- Sets of (external) delete requests are monotonic:
  Let $del(D)$ denote the actual set of (cascaded) deletions on the database.
- $del(D_1 \cup D_2) = del(D_1) \cup del(D_2)$,
- If D_1 and D_2 are admissible (wrt. NO ACTION), then $D_1 \cup D_2$ is admissible
  (but not vice versa, recall the "Belgium" example – $D_1 \cup D_2$ is admissible even though D_1 alone is not admissible).
- with CASCADE and SET NULL, conflicts can arise;
- with DELETE+CASCADE/SET NULL and UPDATE even more conflicts can arise.

12.2.3 Updates: Encoding as Rules

- with updates, the modeling is more involved:
  - updates can create new foreign keys,
  - usually such updates are cascaded from the appropriate parent,
  - but overlapping FKs induce interferences with other K/FKs.
  - Consider `S.X→R.Y ON UPDATE NO ACTION` and $r(a), r(b)$ and $s(a)$.
    A transaction that modifies $r(a) \to r(c)$ and $s(a) \to s(b)$ is admissible.

  $\Rightarrow$ not only tuple-based, but key-foreign-key-based.
  - different cascaded modifications can be applied at the same time to a tuple.
- Recall:
  - SET NULL cannot create new foreign keys – null values cannot violate any SQL integrity constraint (except NOT NULL).
  - actual conflicts in a "diamond" from a single update can only result from CASCADE and SET NULL since then *different* changes are applied.
- the approach must cover the general "worst case", not only "intuitive" cases.

Updates: Encoding as Rules

- for every key and foreign key:
  pot_chg_R_K_1...K_k(X_1,...,X_n,Y_1,...,Y_k) and chg_R_K_1...K_k(X_1,...,X_n,Y_1,...,Y_k)
  (change f/key attributes $K_1, \ldots, K_k$ of tuple $R(X_1, \ldots, X_n)$ to $Y_1, \ldots, Y_k$).
- propagation for every FK/K reference (ON UPDATE CASCADE):
  pot_prp_R_P_R_C_F_1...F_k(X_1,...,X_n,Y_1,...,Y_k)
  (propagate update from R_P's keys to FK $F_1, \ldots, F_k$ of tuple $R_C(X_1, \ldots, X_n)$).
- user updates: projection to the keys and foreign keys $(K_1, ..., K_k)$ of R:
  pot_prp_ext_R_K_1...K_k(X_1,...X_n,Yi_1,...,Yi_k) :-
  ext_mod_R(X_1,...,X_n,Y_1,...,Y_n), (Yi_1,...,Yi_k) != (Xi_1,...,Xi_k).
- collect propagated changes to keys/foreign keys (overlappings!):
  Simplified pattern: Consider two "incoming" propagations from $R_{P_1}.A \to R_C.K_1$ and $R_{P_1}.B \to R_C.K_2$ concern the key (K_1, K_2) of R_C:
  pot_chg_R_C_K_1,K_2(X_1,...,X_n,Y_1,Y_2) :-
  pot_prp_R_{P_1}_R_C_K_1(X_1,...,X_n,Y_1), pot_prp_R_{P_2}_R_C_K_2(X_1,...,X_n,Y_2),
  (Y_1,Y_2) != (X_{K_1},X_{K_2}).
  (for the fully general rule see (CH_1) in the TODS paper)

Updates: Encoding as Rules

Changes of Primary Keys $R_P(K_1, \ldots, K_k)$ are then handled according to the referential actions of their "child tuples"

- $R_C(F_1, \ldots, F_k)$ REFERENCES $R_P(K_1, \ldots, K_k)$ ON UPDATE NO ACTION:
  blk_chg_R_P_K_1,...,K_k(X_1,...,X_n,Y_1,...,Y_k) :-
  pot_chg_R_P_K_1,...,K_k(X_1,...,X_n,Y_1,...,Y_k), rem_refd_R_P_R_C_F_1...F_k(Y_1,...,Y_k).
- $R_C(F_1, \ldots, F_k)$ REFERENCES $R_P(K_1, \ldots, K_k)$ ON UPDATE CASCADE:
  pot_prp_R_P_R_C_F_1...F_k(Z_1,...,Z_n,Y_1,...,Y_k) :-
  pot_chg_R_P_K_1,...,K_k(X_1,...,X_n,Y_1,...,Y_k),
  π[F_1,...,F_k](Z_1,...,Z_n) = π[K_1,...,K_k](X_1,...,X_n).
  and (block propagation if change of child is blocked)
  blk_prp_R_P_R_C_F_1...F_k(X_1,...,X_n,Y_1,...,Y_k) :-
  pot_prp_R_P_R_C_F_1...F_k(X_1,...,X_n,Y_1,...,Y_k),
  blk_chg_R_C_F_1,...,K_k(X_1,...,X_n,Y_1,...,Y_k)
  and (block parent change if propagation to some child is blocked)
  blk_chg_R_P_K_1,...,K_k(X_1,...,X_n,Y_1,...,Y_k),
  pot_chg_R_P_K_1,...,K_k(X_1,...,X_n,Y_1,...,Y_k),
  blk_prp_R_P_R_C_F_1...F_k(Z_1,...,Z_n,Y_1,...,Y_k),
  π[F_1,...,F_k](Z_1,...,Z_n) = π[K_1,...,K_k](X_1,...,X_n).

Updates: Encoding as Rules

Further rules (see TODS paper):

- when a key value remains referenced (child not deleted, not "modified away"),
- when a key value gets newly referenced (insert, "modified towards"),
- when a key value gets newly referencable (insert, "modified towards"),
- when a change of a FK is blocked because the new reference does not exist,
- when 2 changes on a tuple are inconsistent (del/upd, different values (null vs. a value),
- rules that derive the modifications to be finally executed.

Results

- if an update set is admissible, the well-founded model is sufficient: giving priorities to modifications/propagations/changes vs. blockings yields a total stable model and the updates to be executed
- if an update set is not admissible, the stable models indicate what portions of the initial set are admissible, and where the problems are located (mutually excluding updates, missing cascades, unresolved tuples).

12.3 The Limits

Recall:

- ASP is based on atoms (i.e., disjunction is not dealt with as formula, but resolved into atoms)
  (note: in tableaux, it is also broken down to atoms)
- ASP algorithms are internally based on ground atoms and grounding of all rules (with all ground terms=constants of the Herbrand universe).
  (note: tableaux can keep variables and use them on-demand)
- ASP is closely related with Model Checking: generate models by a strategy.

Expressiveness

- Datalog$^\neg$ rules, for ASP extended with:
  - disjunction/choice rules: generate suitable different models
    (cf. tableau: branches)
  - negation: via denials. Discard models.
    (cf. tableau: close branch)

The Limits

The Limits of ASP:

- no existential quantification, object invention:
  - every person has a father who is again a person.
    (tableau: via skolemization and strategical application of the δ-rule)
  - ASP/grounding: would require infinitely many ground instances.
    (tableau: strategic application of the γ-Rule + Blocking strategies (e.g., DL/OWL reasoning))
- $P \models a$ means only that a holds in the minimal/stratified/well-founded model.
  Statements about $\neg a$ only by default negation/closed world.
  (On the other hand: LP provides a CWA reasoning formalism – in contrast to tableaux)

Comparison: Tableaux

- Tableaux can be tailored to any logic: FOL, DL/OWL, ...
  - Open-World: monotonic
  - complex expansion & blocking strategies + heuristics.
- ASP is worst-case exponential,
  but with a polynomial basis: the AFP computation for the well-founded model.

12.4 DB vs. KB: Closed World vs. Open World

Consider the following formula F:

$$\begin{aligned} F \quad \equiv \quad & person(\text{“John”}, 35) \wedge person(\text{“Alice”}, 10) \wedge person(\text{“Bob”}, 8) \wedge \\ & \wedge person(\text{“Carol”}, 12) \wedge person(\text{“Jack”}, 65) \wedge \\ & \wedge child(\text{“John”}, \text{“Alice”}) \wedge child(\text{“John”}, \text{“Bob”}) \wedge \\ & \forall X, Y : (\exists Z : (child(Z, X) \wedge child(Z, Y) \wedge X \neq Y) \rightarrow sibling(X, Y)) \end{aligned}$$

- Does $child$(“John”, “Bob”) hold? – obviously yes.
- Does $G{:}\equiv sibling$(“Alice”, “Bob”) hold?
  - (Relational) Database: $sibling$ is a view. The answer is “yes”.
  - FOL KB: for all models $\mathcal{M}$ of F, G holds. Thus, $F \models sibling$(“Alice”, “Bob”).
- What about $G{:}\equiv sibling$(“Alice”, “Carol”)?
  - (Relational) Database: no. For the database state $\mathbf{D}$, $\mathbf{D} \not\models sibling$(“Alice”, “Carol”).
  - FOL KB: there is a model $\mathcal{M}_1$ of F, where $\mathcal{M}_1 \not\models G$, but there is also a model $\mathcal{M}_2$ of F, where $\mathcal{M}_2 \models G$ (e.g., add the tuple (“John”, “Carol”) to the interpretation of $child$).

  For the Web, $child$(“John”, “Carol”) can e.g. be contributed by another Web Source.

DB vs. KB: Closed World vs. Open World

- What about $G :\equiv child$(“John”, “Jack”)?
  - (Relational) Database: no. For the database state $\mathbf{D}$, $\mathbf{D} \not\models child$(“John”, “Jack”).
  - FOL KB: there is a model $\mathcal{M}_1$ of F, where $\mathcal{M}_1 \not\models G$, but there is also a model $\mathcal{M}_2$ of F, where $\mathcal{M}_2 \models G$.
- Obviously, the KB does not know that a child cannot be older than its parents.

  Add a constraint to F, obtaining F':

  $$F' :\equiv F \wedge \forall P, C, A_1, A_2, : (person(P, A_1) \wedge person(C, A_2) \wedge child(P, C)) \rightarrow A_1 > A_2$$

  - database: this assertion would prevent to add $child$(“John”, “Jack”) to the database.
  - for the KB, $F' \models \neg child$(“John”, “Jack”) allows to *infer* that Jack is not the child of John.

  Such information can be given with the *ontology* of a domain.

DB vs. KB: Closed World vs. Open World

- the Database Model Theory is called *"Closed World"*: things that are not known to hold are *assumed* not to hold.
- the FOL semantics is called *"Open World"*: things that are not known to be true or false are considered to be *possible*.

Consequences on Negation

- in databases there is no explicit negation. It is not necessary to specify that Jack is *not* a child of John.
- in a KB, it would be necessary to state ... $\wedge \neg child($"John", $X)$ for all persons who are known not to be children of John.
  Additional constraints: extend the ontology, e.g., by stating that a person has exactly two parents – then all others cannot be parents – works only for persons whose parents are known. Similarly for the "age" constraint from the previous slide.
- note that the semantics of universal quantification ($\forall$) is also effected: $\forall X : \phi$ is equivalent to $\neg\exists X : \neg\phi$.

Reasoning in Presence of Negation

Obtaining new information (e.g., by finding another Web Source) has different effects on Open vs. Closed world:

- Closed world: conclusions drawn before – "Carol is not a child of John", or "John has exactly two children" from less information can become invalid.
  This kind of reasoning is *nonmonotonic*
- Open world: everything which is not known explicitly is taken into account to be possible (by considering all possible models).
  This kind of reasoning is *monotonic*:
  $$\text{Knowledge}_1 \subseteq \text{Knowledge}_2 \Rightarrow \text{Conclusions}_1 \subseteq \text{Conclusions}_2$$
- Open World can be combined with other forms of nonmonotonic reasoning, e.g., Defaults: "usually, birds can fly". Knowing that Tweety is a bird allows to conclude that it flies.
  Obtaining the information that Tweety is a penguin (which can usually not fly) leads to invalidation of this conclusion.

The current Semantic Web research mainstream prefers Open World without default reasoning.

COMPARISON, MOTIVATION ETC.

Database vs. FOL

Relational Databases	relational schema	tuples	SQL queries	closed world
FOL	signature (predicates +functions)	facts (atoms)	$\mathcal{S} \models \phi$? (yes/no or answer $\psi \models \phi$? variable bindings)	mostly: open world sometimes closed world

Situations and tasks

Given	what to do	how?
facts/database	does $p(\ldots)$ hold in the DB? SQL query	by combining data
facts+constraints (SQL assertions or FOL formulas)	additionally: test if constraints satisfied	equivalent to first situation (query for violating tuples)
facts (DB) rules (KB)	does $p(\ldots)$ hold in DB+rules?	DB+views application of rules
facts (DB) knowledge base KB as FOL formulas	is a formula ϕ entailed by DB+KB?	reasoning, entailment, KB $\models \phi$?