Existential Rules: A Graph-Based View

Marie-Laure Mugnier

Existential Rules:

A Graph-based View

Marie-Laure Mugnier

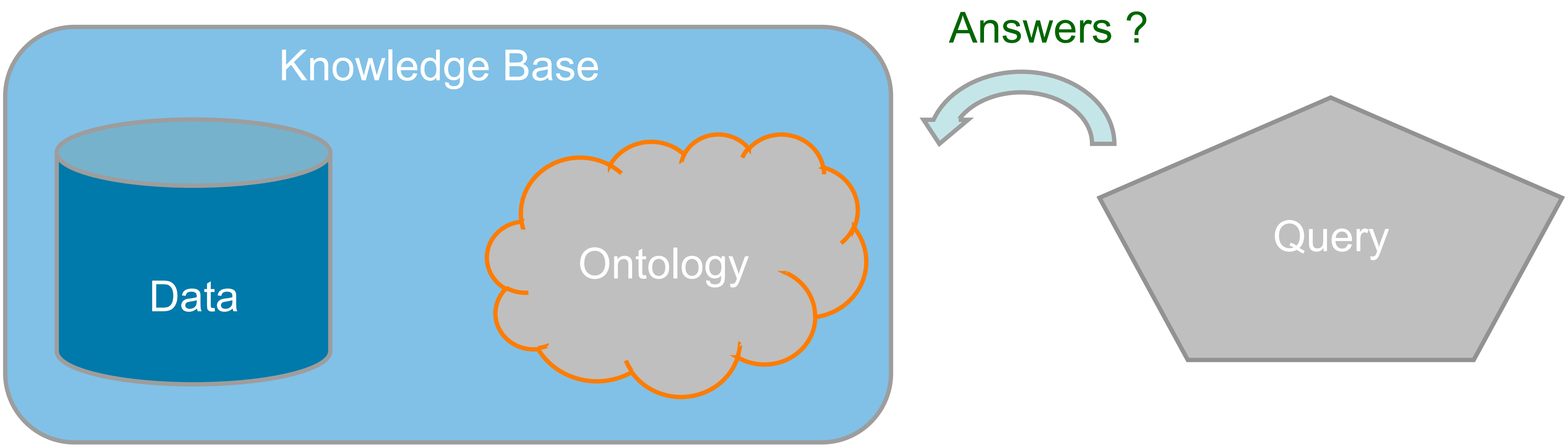
University of Montpellier

Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier

LIRMM

Datalog 2.0, Vienna, 2012

Ontology-based Data Access (OBDA)



Adding an ontological layer:

- to abstract from a specific database schema
- to provide a unified view of mutiple sources
- to infer new facts, thus allowing for data incompleteness

Outline

- Existential rules: a logic- and graph-based framework

- Decidability and algorithmic issues

  - Focus on:
    - tree-shaped saturation in forward chaining
    - piece-based unification in backward chaining

- A (graph) tool for combining decidable classes of rules

Data / Facts

Relational Database

parentOf

a	b
a	c
c	?
…	…

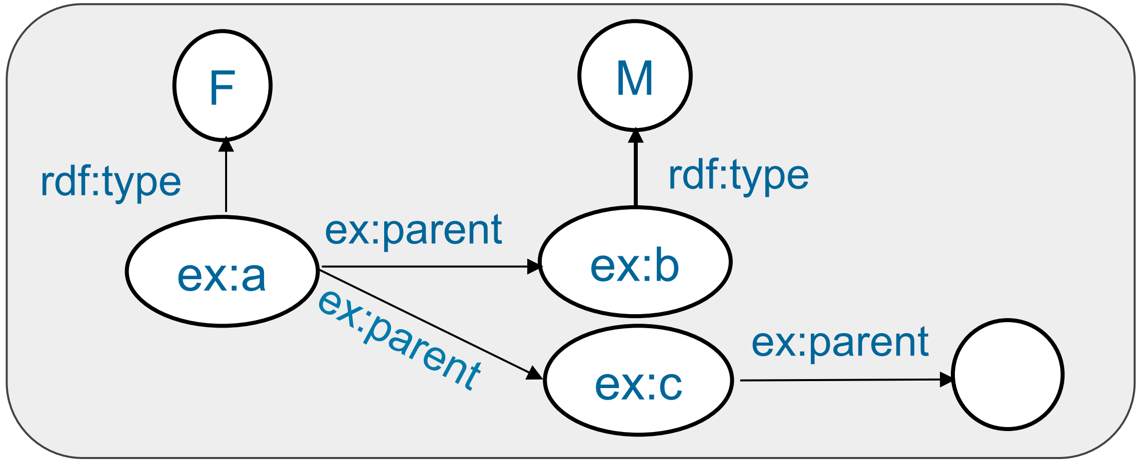
Male

b
…

Fem.

a
…

RDF (Semantic Web)

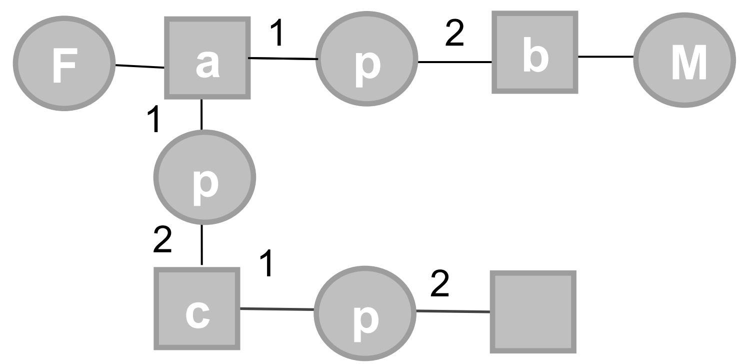


Etc.

Abstraction in first-order logic

$$\exists x(\ \mathrm{parentOf}(a,b) \wedge \mathrm{parentOf}(a,c) \wedge \mathrm{parentOf}(c,x) \wedge F(a) \wedge M(b)\ )$$

Or in **graphs / hypergraphs**



Ontology: Existential Rules

« Value Invention »

$$\forall X\ \forall Y\ (\ \underbrace{B[X, Y]}_{\text{Body}} \rightarrow \exists Z\ \underbrace{H[X, Z]}_{\text{Head}}\ )$$

X, Y, Z : *tuples of variables*

Any conjunction of atoms (on variables and constants)

∀x ∀y (siblingOf(x,y) → ∃ z (parentOf(z,x) ∧ parentOf(z,y)))

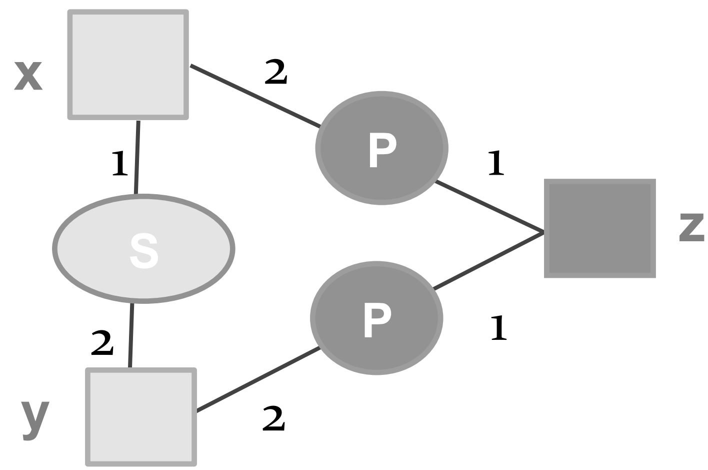
Simplified form: siblingOf(x,y) → parentOf(z,x) ∧ parentOf(z,y)

- Same as Tuple Generating Dependencies (TGDs)
- See also Datalog+/-
- Same as the logical translation of Conceptual Graph rules
- Generalize Description Logics used for OBDA (DL-Lite, $\mathcal{EL}$)

Ontology: Existential Rules

$$\forall X\ \forall Y\ (\ \underbrace{B[X, Y]}_{\text{graph}} \rightarrow \exists Z\ \underbrace{H[X, Z]}_{\text{graph}}\ )$$

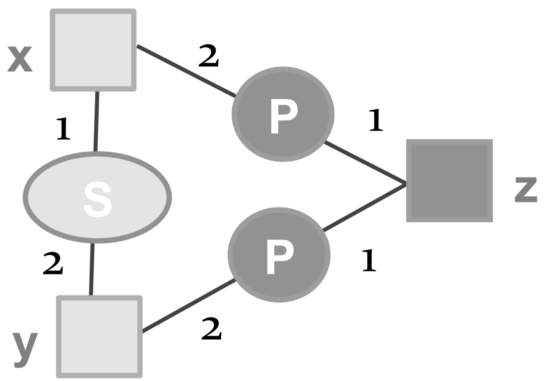
$$\forall x\ \forall y\ (siblingOf(x,y) \rightarrow \exists\ z\ (parentOf(z,x) \wedge parentOf(z,y)))$$



Value Invention

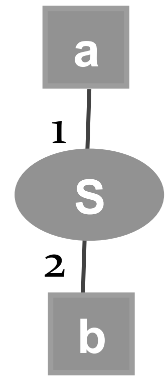
R = ∀x ∀y (siblingOf(x,y) → ∃ z (parentOf(z,x) ∧ parentOf(z,y)))

F = siblingOf(a,b)



h: body → F

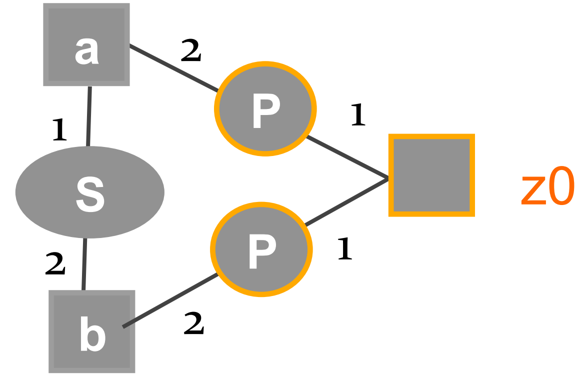
h =*{(x,a), (y,b)}*



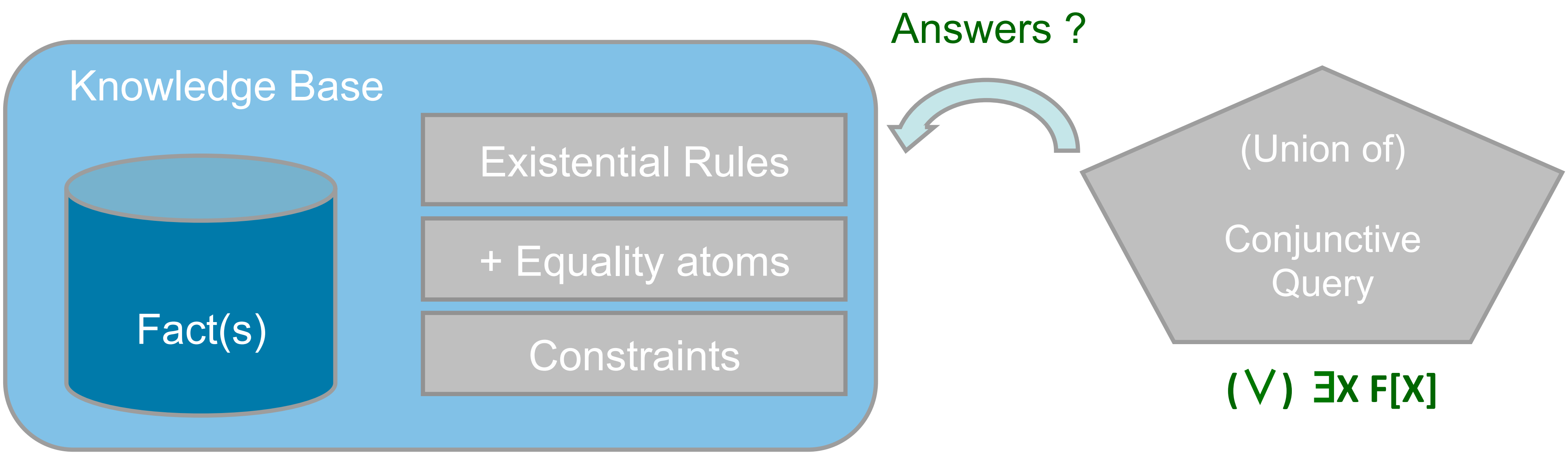
A rule *body* → *head* is applicable to a fact *F* if there is a homomorphism *h: body* → *F*

Then *h(head)* can be « added » to *F*
with renaming existential variables of head

F' = ∃ z0 (siblingOf(a,b)
∧ parentOf(z0,a) ∧ parentOf(z0,a))



Logical /Graphical Framework



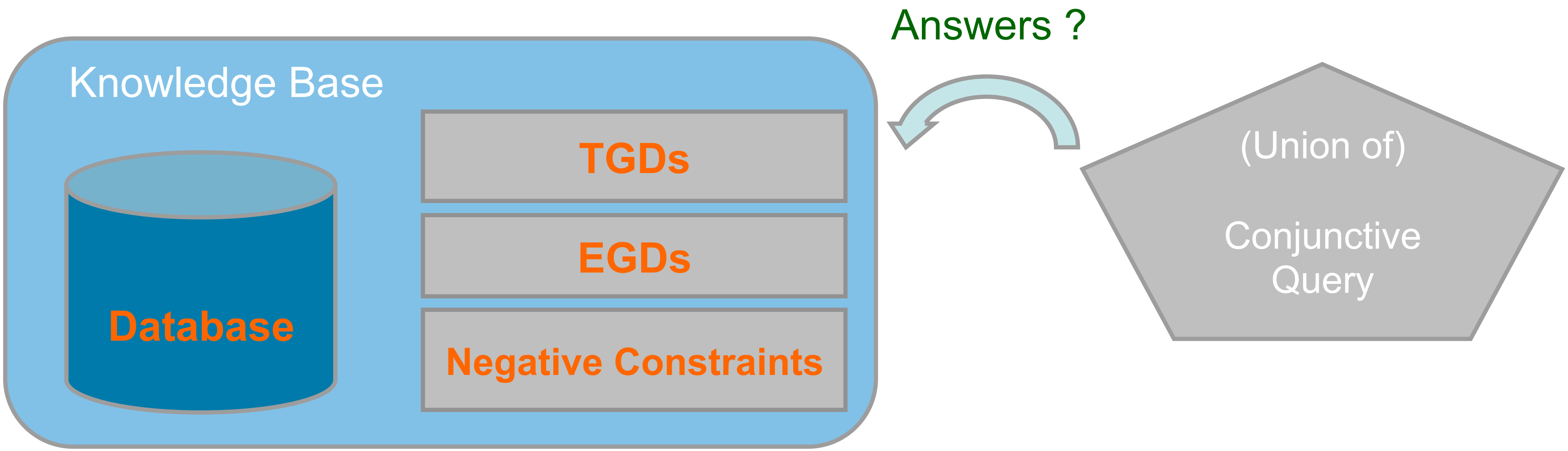
Negative constraint: $\neg\,(\exists X\ B[X])$ or $\forall X\ (B[X] \rightarrow \bot)$

« B[X] must not be found »

Positive constraint: $\forall X\ \forall Y\ (\ B[X, Y] \rightarrow \exists Z\ H[X, Z]\ )$

« if B[X,Y] is found then H[X,Z] must also be found »

Similar Framework: Datalog +/-



[Cali Gottlob Lukasiewicz PODS 2009]

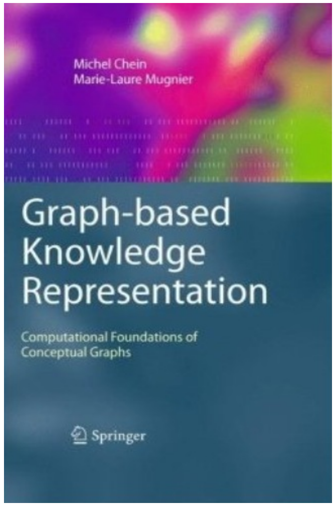
Tuple Generating Dependency = (pure) existential rule

Equality Generating Dependency: $\forall X \ (B[X] \rightarrow x = e)$

The Conceptual Graph Origins

- Conceptual graphs introduced in [Sowa 76] [Sowa 84]
- Specific research line by Montpellier's group since 1992

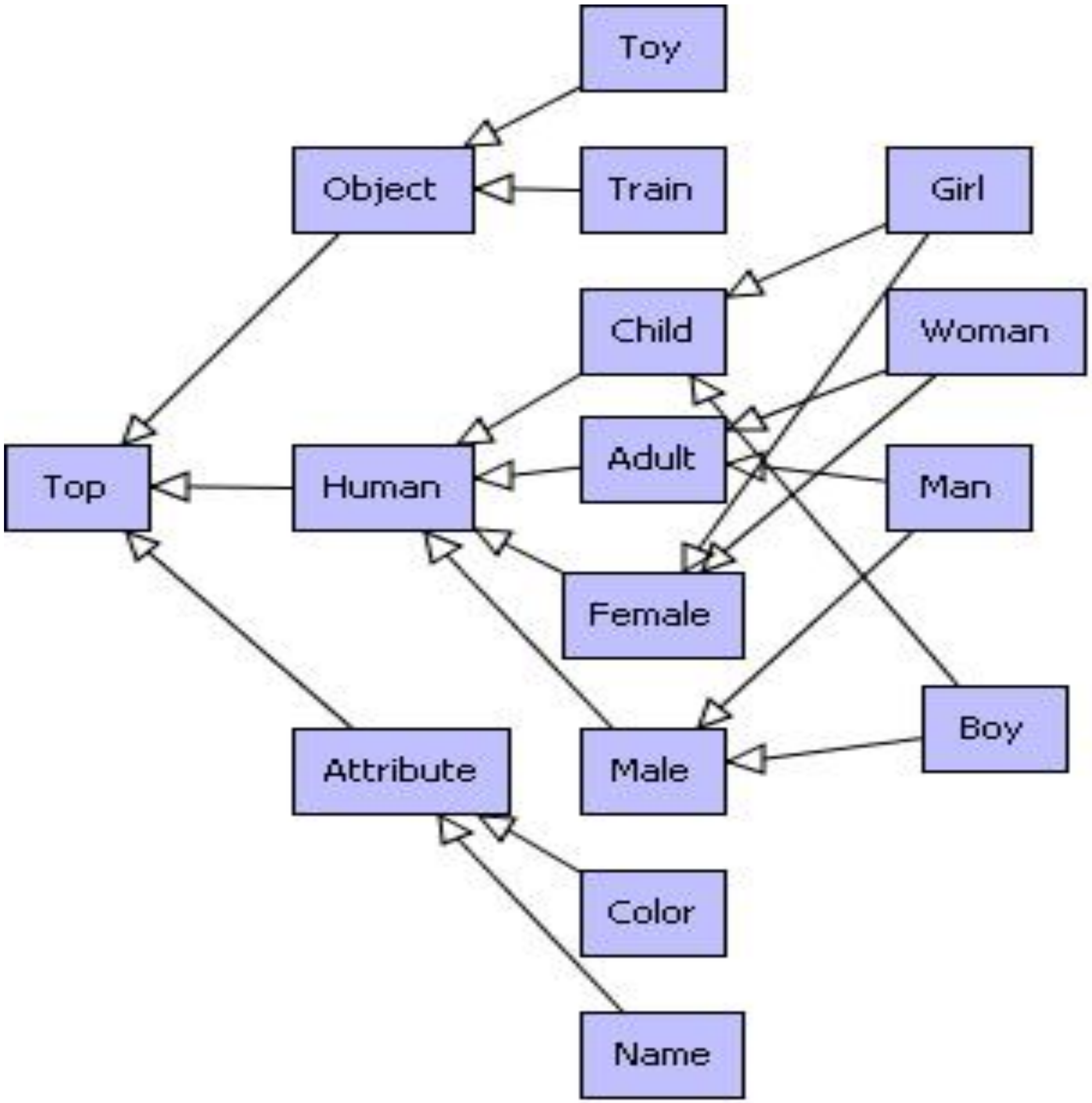
  « Graph-based » knowledge representation and reasoning

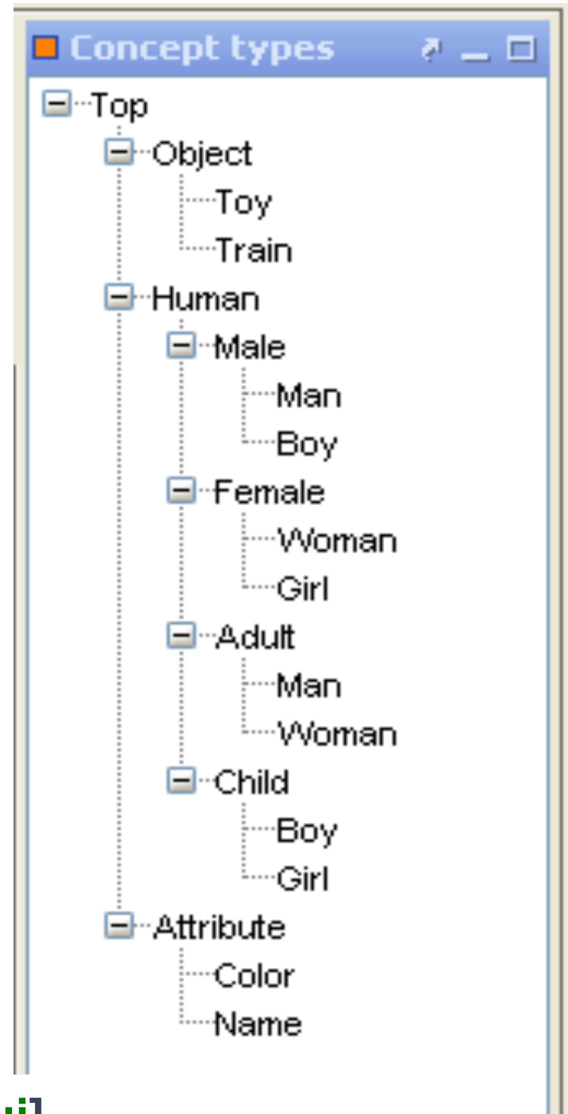


« Graph-Based Knowledge Representation: Computational Foundations of Conceptual Graphs », Chein & M..., Springer, 2009

Conceptual Graph Vocabulary:

1. partially (pre-)ordered set of **concepts**

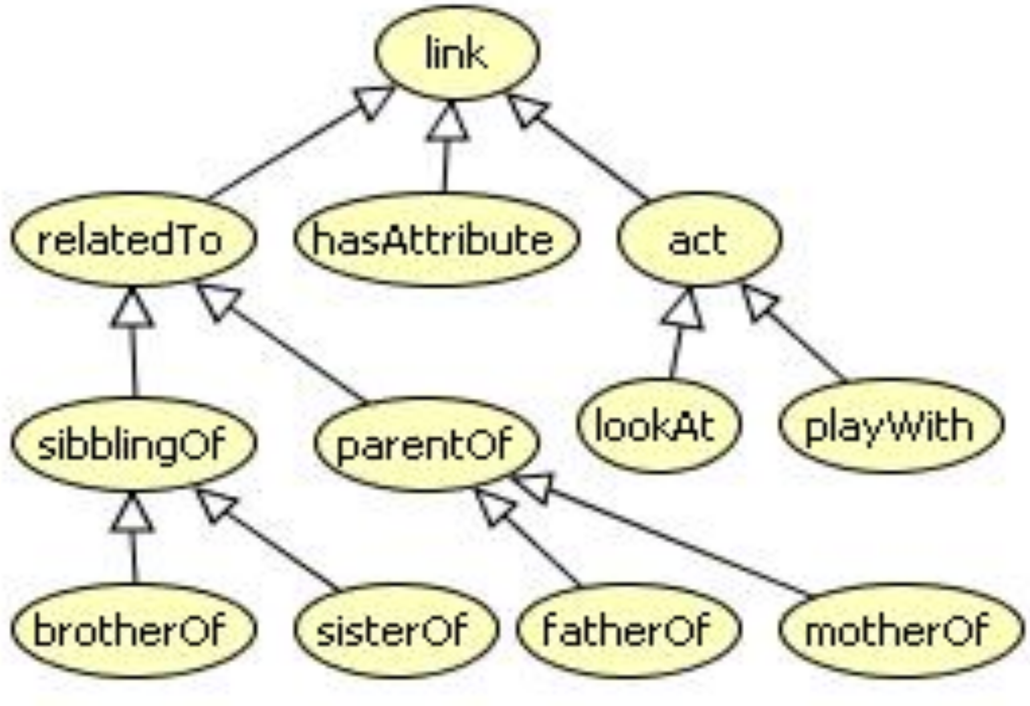


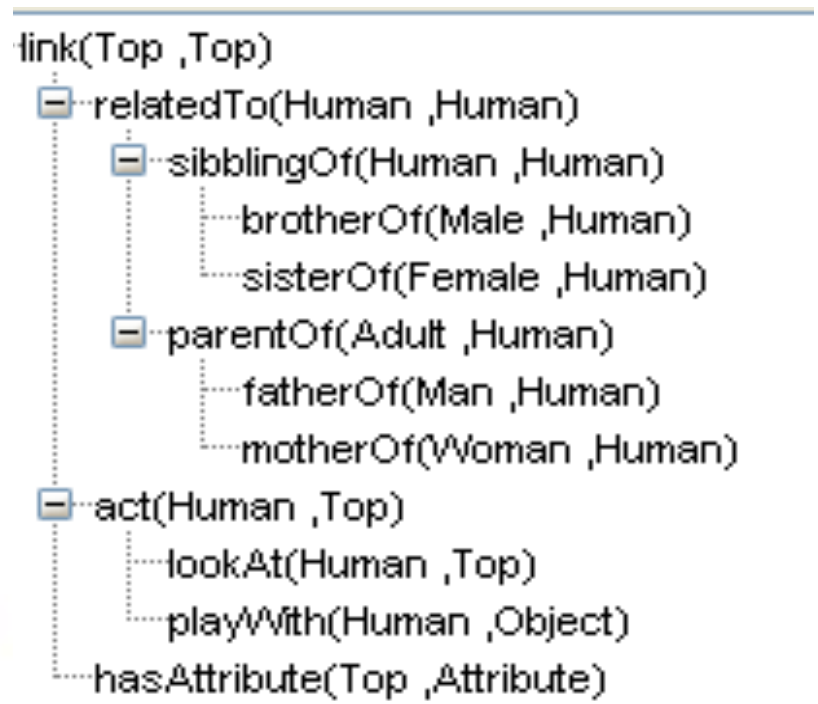


[screenshots from CoGui, http://www.lirmm.fr/cogui]

Conceptual Graph Vocabulary:

2. partially (pre-)ordered set of **relations** with their **signature** [any relation arity allowed]



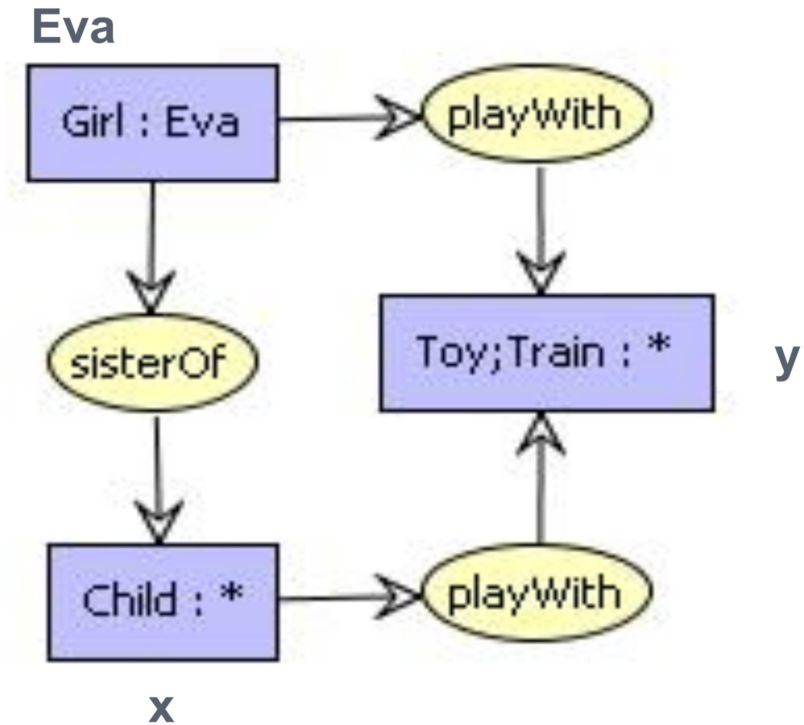


Logical translation (Φ) of the vocabulary: very simple rules

$p < q$ $\quad \forall x_1 \ldots x_k \, (p(x_1 \ldots x_k) \rightarrow q(x_1 \ldots x_k))$

Signature of r $\quad \forall x_1 \ldots x_k \, (p(x_1 \ldots x_k) \rightarrow t_{i1}(x_1) \ldots t_{ik}(x_k))$

Basic Conceptual Graph



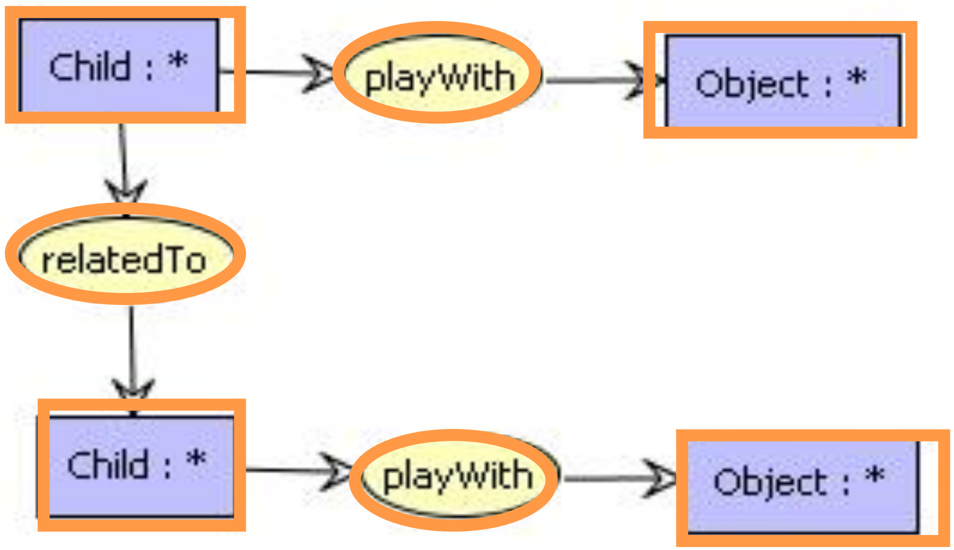
[total order on the edges incident to a relation node]

Logical translation (Φ): existentially closed conjunction of atoms

$$\exists x\, \exists y\, (\mathrm{Girl(Eva)} \wedge \mathrm{Child}(x) \wedge \mathrm{Toy}(y) \wedge \mathrm{Train}(y) \wedge \mathrm{sisterOf(Eva},x) \wedge \mathrm{playWith(Eva},y) \wedge \mathrm{playWith}(x,y))$$

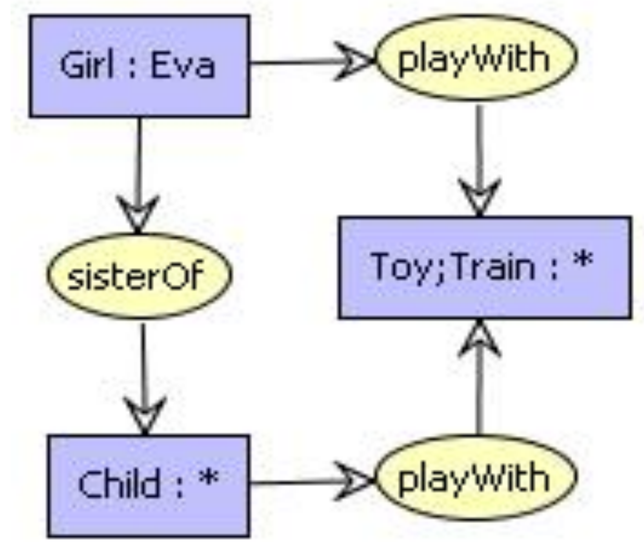
Allows to represent **facts** and **conjunctive queries**

Homomorphism (with concept/relation preorders integrated)



Query *Q*

Fact *F*

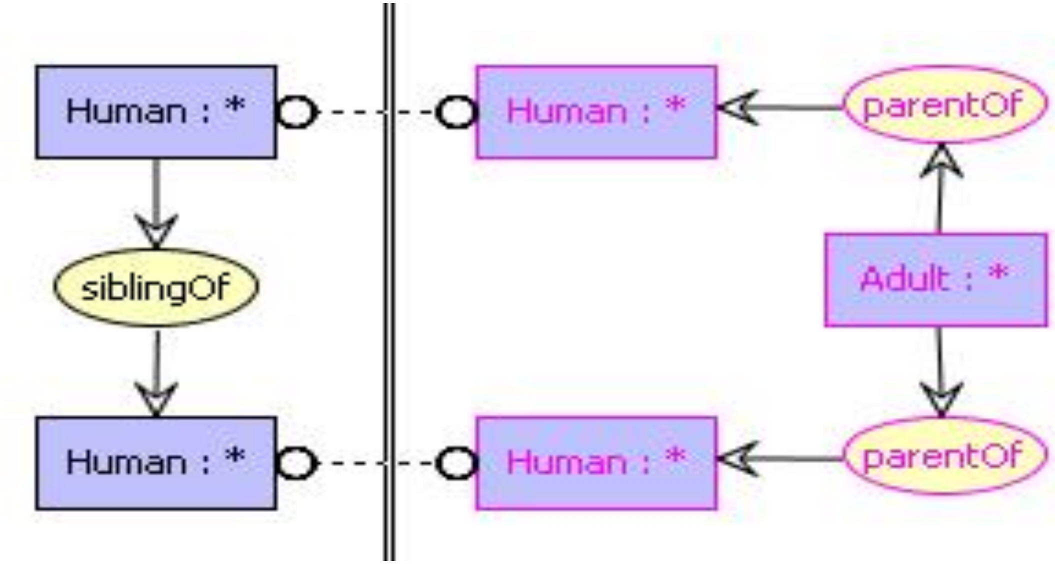


Logical **soundness** [Sowa 84] and **completeness** [Chein M… 92]:
there is a homomorphism from *Q* to *F* iff
Φ(Q) is entailed by Φ(F) and Φ(vocabulary)

The Basic CG fragment restricted to binary relations is equivalent to **RDFS** [Baget ISWC' 05] [Baget+ ICCS' 10]

Richer Fragments (nested graphs, rules, constraints, + negation, …)

- Rule: pair of basic conceptual graphs



$$\forall x\ \forall y\ (Human(x) \wedge Human(y) \wedge siblingOf(x,y) \rightarrow \exists z\ (Adult(z) \wedge parentOf(z,x) \wedge parentOf(z,y)))$$

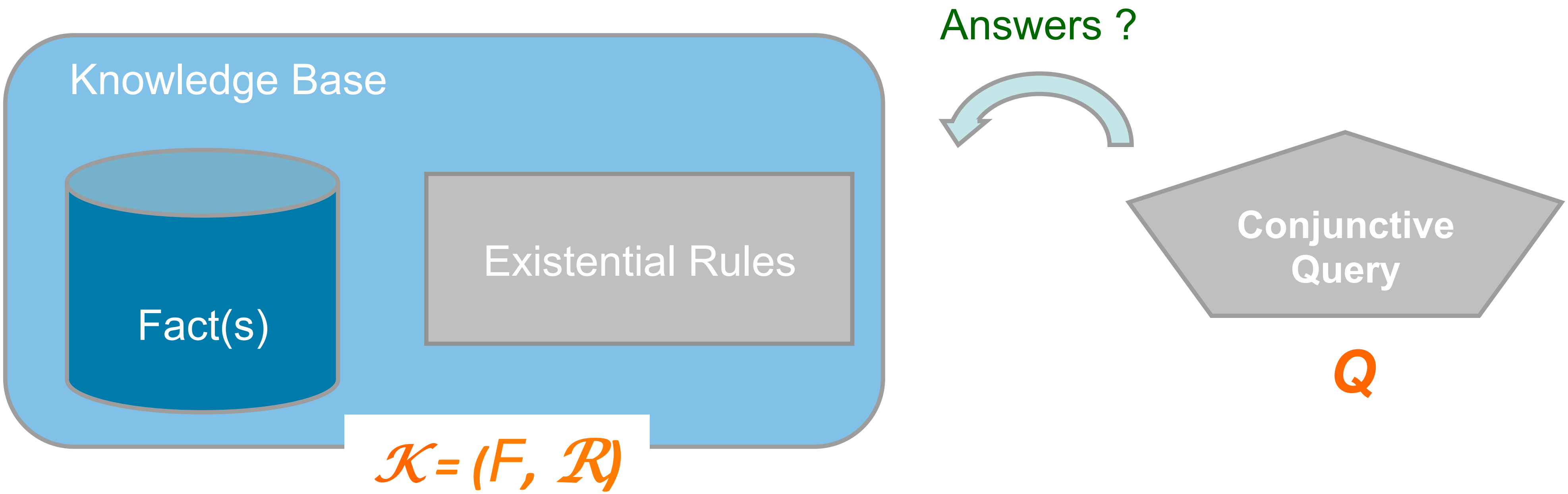
- Sound and complete forward chaining and backward chaining [Salvat M… 1996]
- Several ways of combining rules and constraints [Baget M… JAIR 2002]

The existential rule framework can be seen as a fragment of CGs with a flat vocabulary

Outline

- Existential rules: a logic- and graph-based framework
- Decidability and algorithmic issues
  - Focus on:
    tree-shaped saturation in forward chaining
    piece-based unification in backward chaining
- A (graph) tool for combining decidable classes of rules

Basic Problem

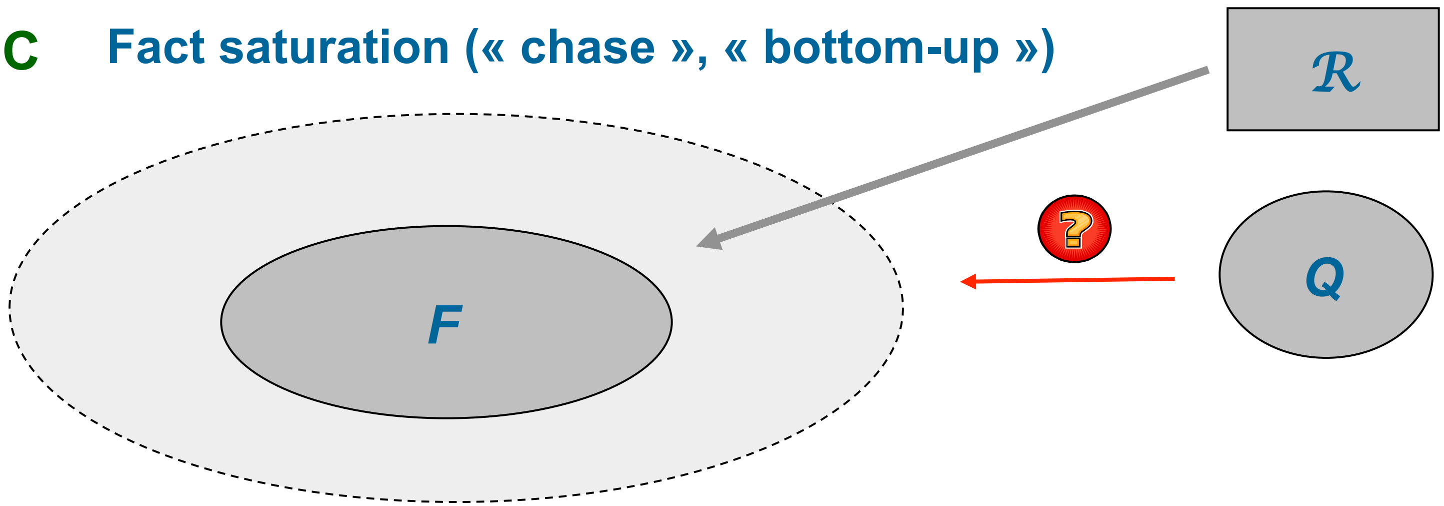


- Conjunctive Query Entailment

Given a KB $\mathcal{K} = (F, \mathcal{R})$ and a (Boolean) conjunctive query Q, is Q entailed by $\mathcal{K}$?

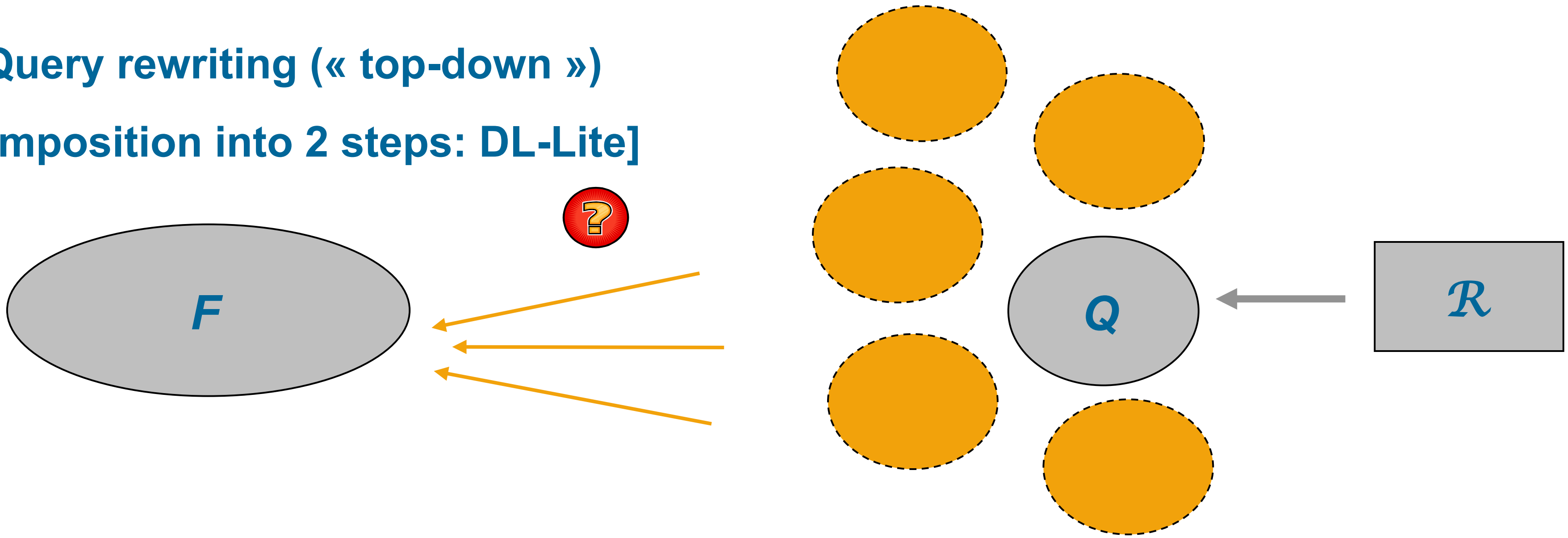
Forward vs Backward Chaining

FC Fact saturation (« chase », « bottom-up »)



BC Query rewriting (« top-down »)

[Decomposition into 2 steps: DL-Lite]



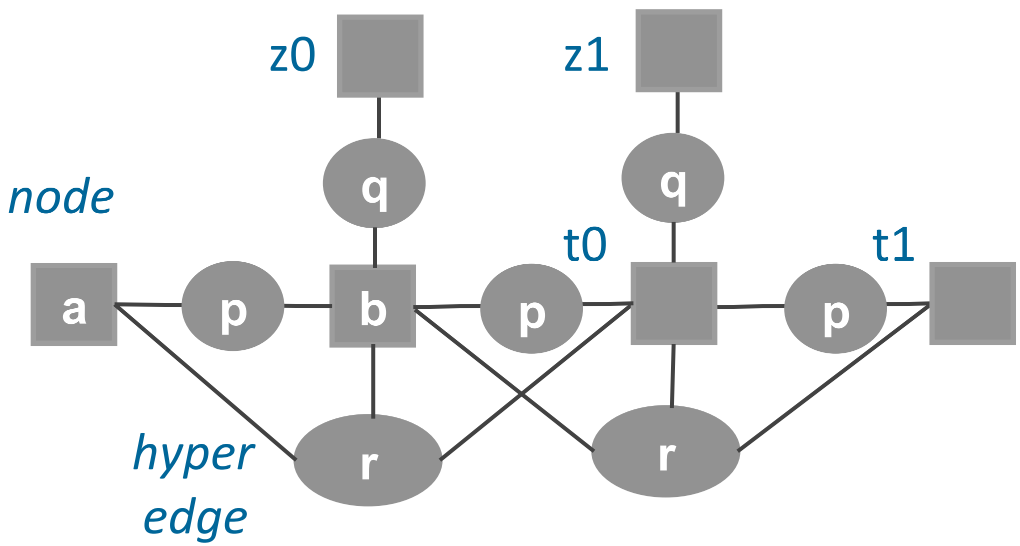
Decidability Issues

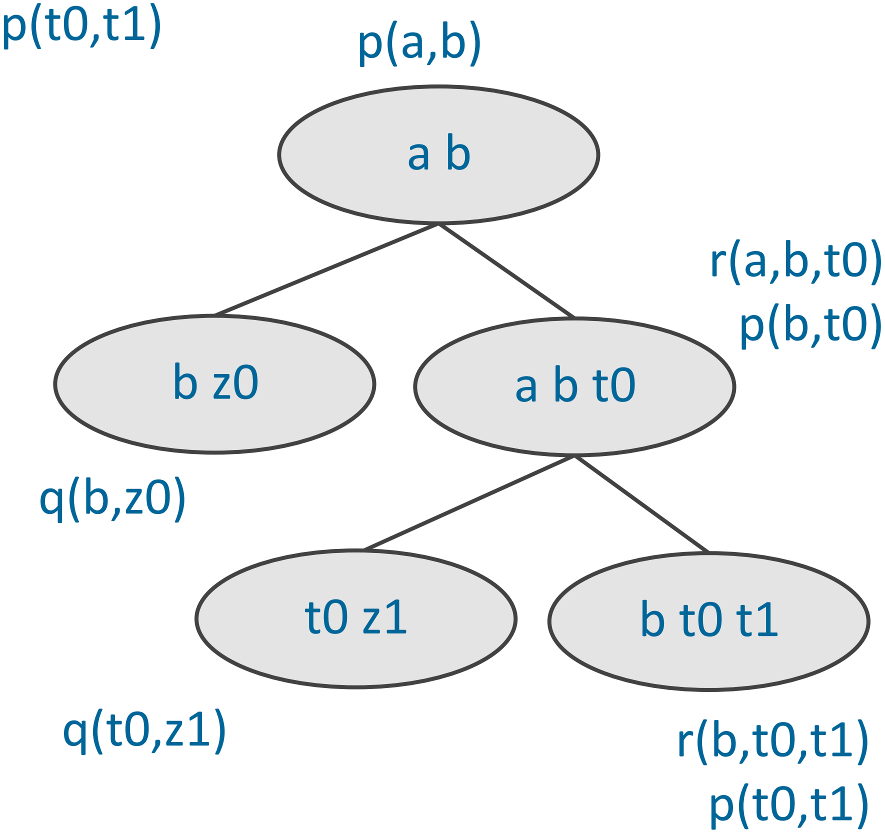
- Entailment is not decidable
- Many decidable classes exhibited in databases and KR
- Three generic kinds of properties ensuring decidability:
  - Saturation by Forward Chaining halts (« finite expansion set », *fes*)
  - Query rewriting by Backward Chaining halts (« finite unification set », *fus*)
  - Saturation by Forward Chaining may not halt *but* the generated facts have a tree-like structure (« bounded treewidth set », *bts*)

None of these properties is *recognizable* [Baget+ KR 10] but they provide *generic* algorithms

Decomposition Tree / Treewidth

p(a,b) q(b,z0) r(a,b,t0) p(b,t0) q(t0,z1) r(b,t0,t1) p(t0,t1)





Decomposition tree:

1) each node *(term)* appears in a bag

2) each hyperedge *(atom)* has all its nodes in a bag

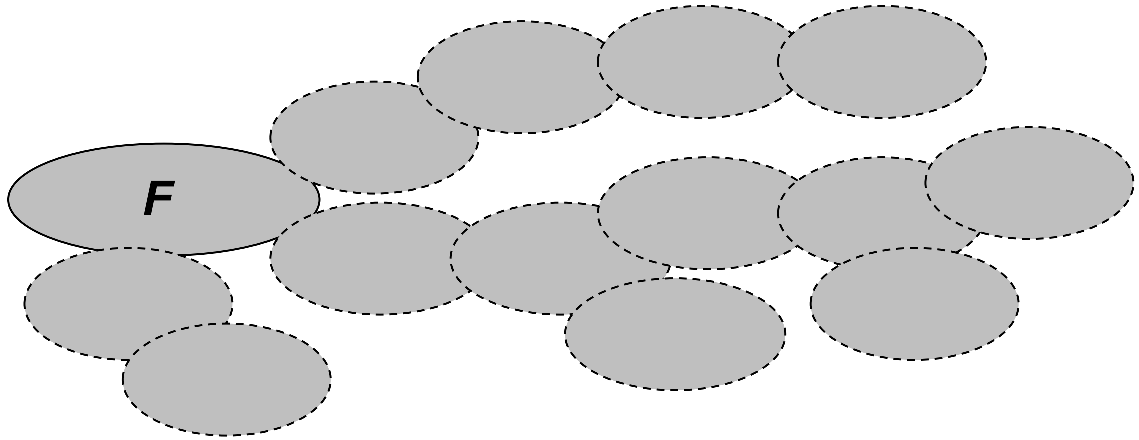
3) for each node *x*, the subgraph induced by the bags containing *x* is connected

Width of a tree decomposition = *max* number of nodes in a bag (minus 1)
Treewidth of a graph = *min* width over all decomposition trees of this graph

Bounded Treewidth of the Derived Facts *(bts)*

Essentially [Cali Gottlob Kifer KR'08]

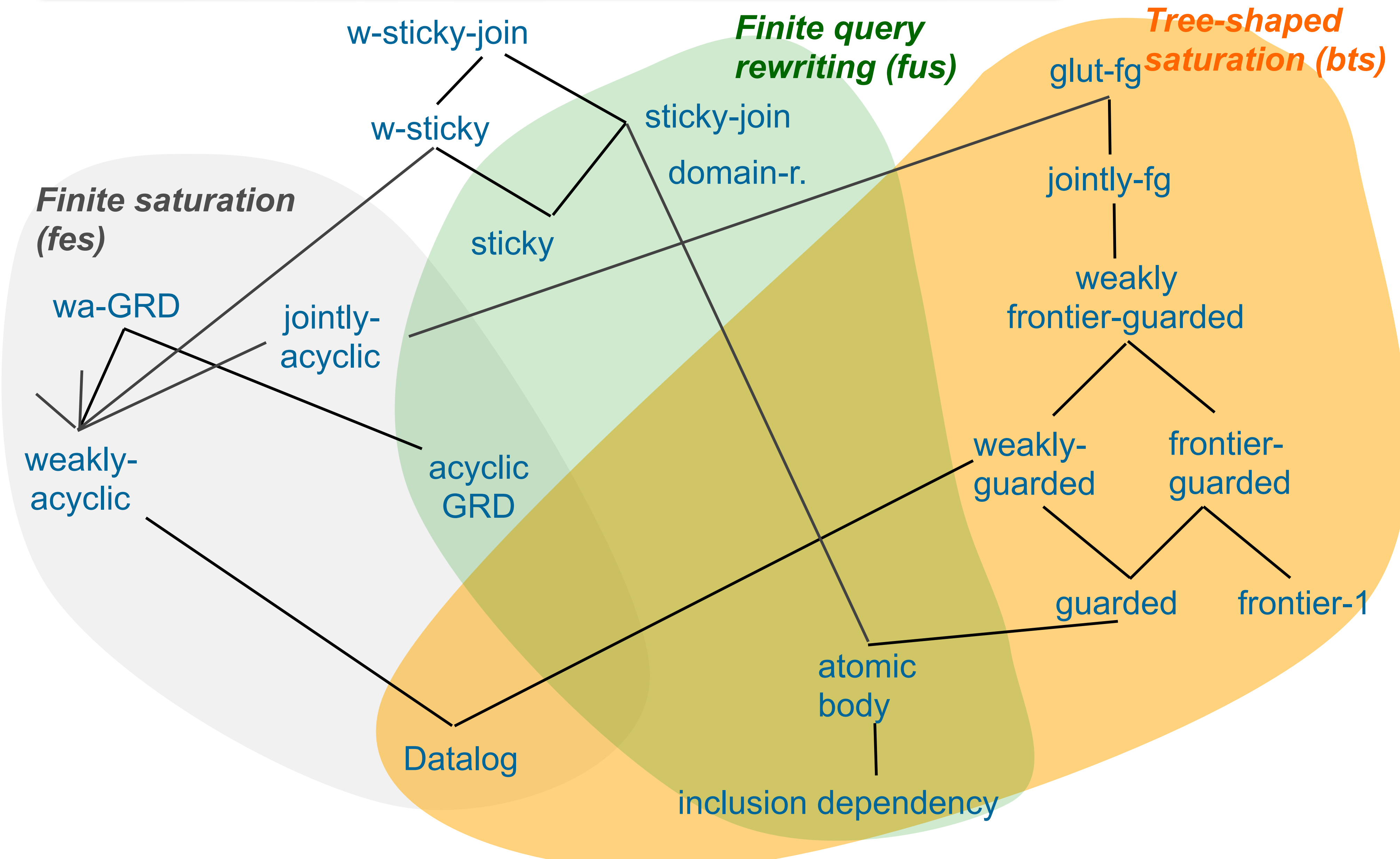
$\mathcal{R}$ is *bts* if FC with $\mathcal{R}$ generates facts with bounded treewidth
i.e., for any fact *F*, there is an integer *b* s.t.
any fact $\mathcal{R}$-derived from *F* has treewidth bounded by *b*



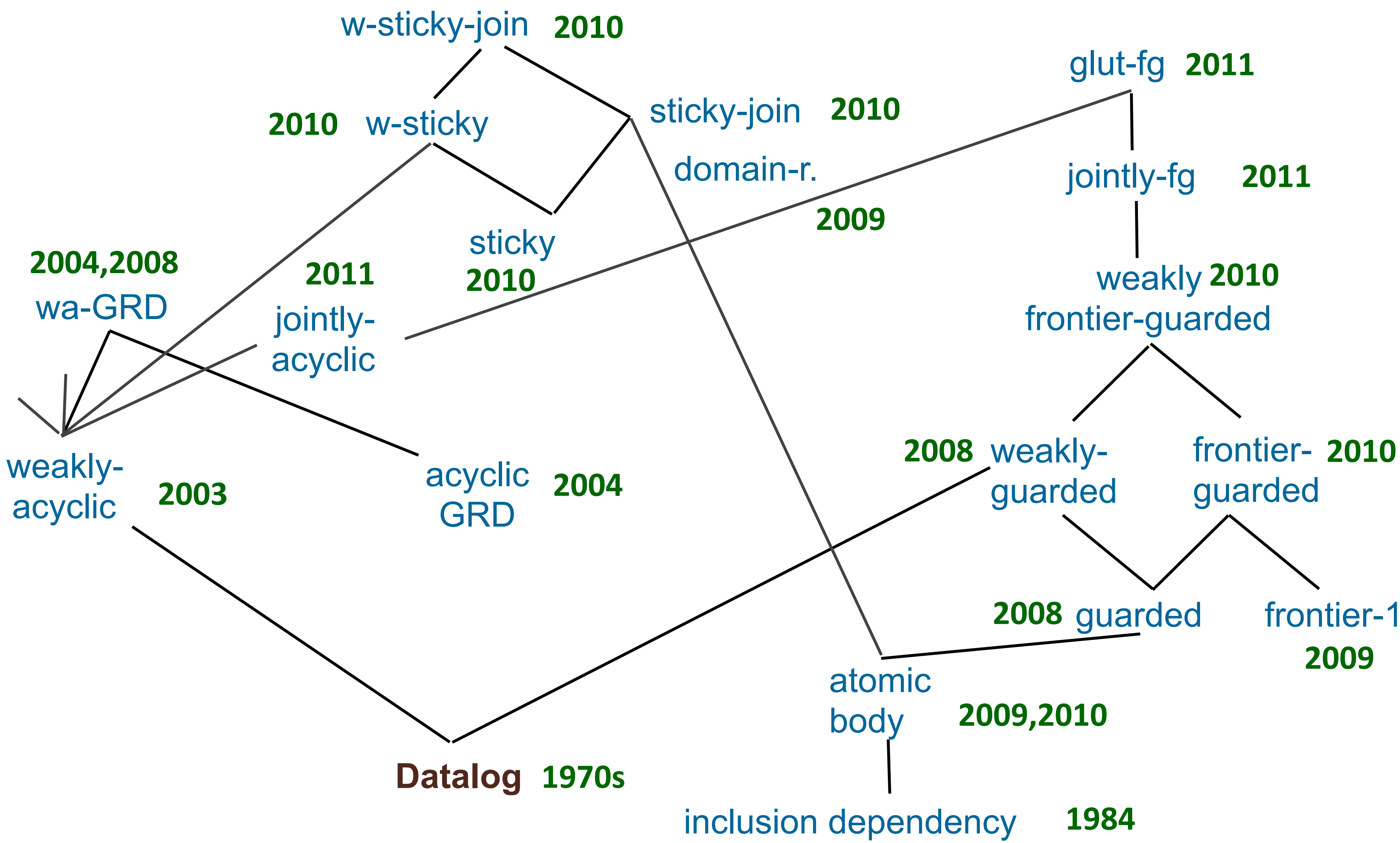
fes (finite saturation) is included in *bts*
(bound given by the number of terms in the finite « saturated fact »)

The decidability proof does not provide a halting algorithm
(relies on the bounded treewidth model property [Courcelle 90])

(Partial) Inclusion Map of Decidable Classes



(Partial) Inclusion Map of Decidable Classes



Some Recognizable bts (and not fes) Classes of Rules

Frontier: variables shared by the body and the head

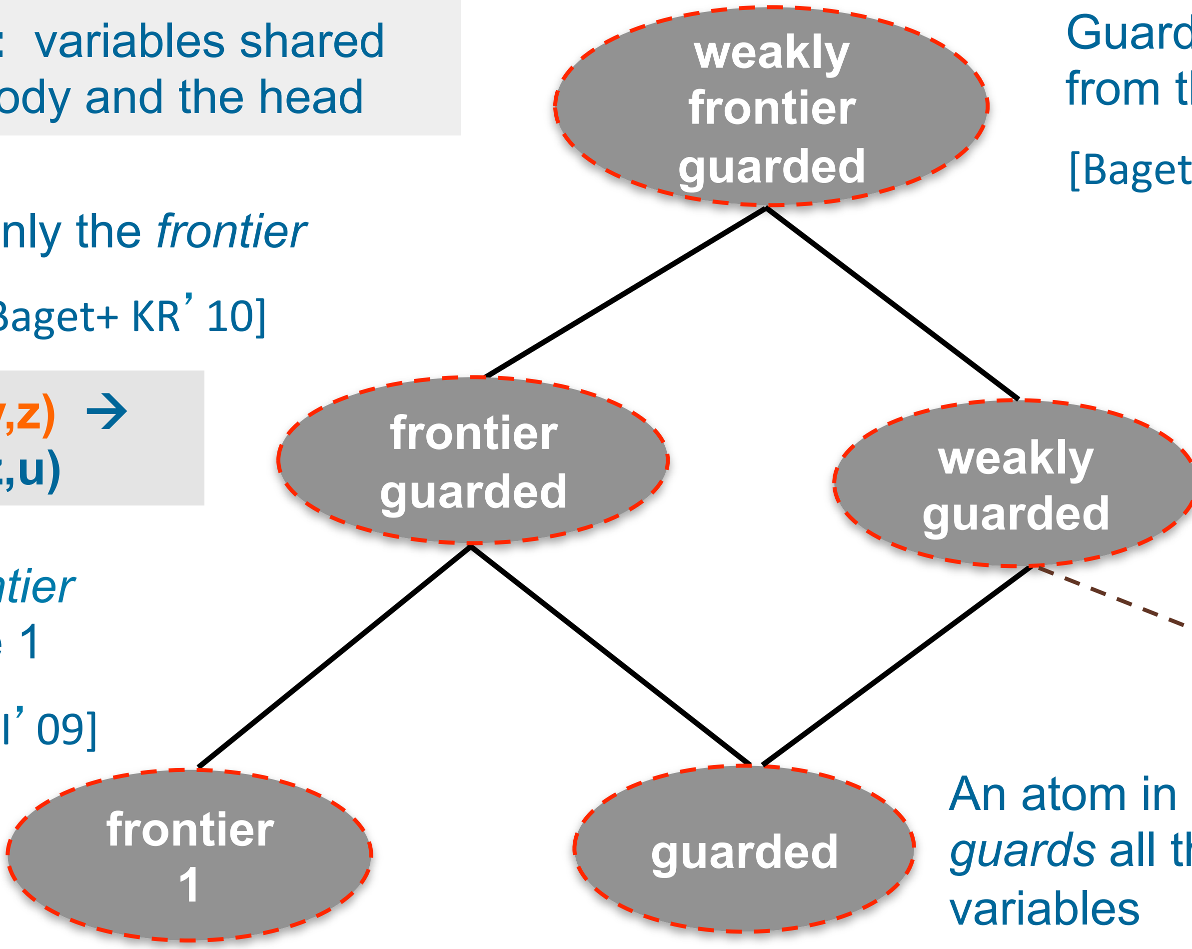
Guard only *affected* variables from the *frontier*

[Baget+ KR' 10]

Guard only the *frontier*

[Baget+ KR' 10]

r(x,y) ∧ r(y,z) → r(y,u) ∧ r(z,u)



Guard only *affected* variables (i.e.possibly mapped to new existentials)

[Cali+ KR' 08]

datalog

The *frontier* has size 1

[Baget+ IJCAI' 09]

An atom in the body *guards* all the body variables [Cali+ KR' 08]

r(x,y) ∧ r(y,z) ∧ r(x,z) → r(z,u)

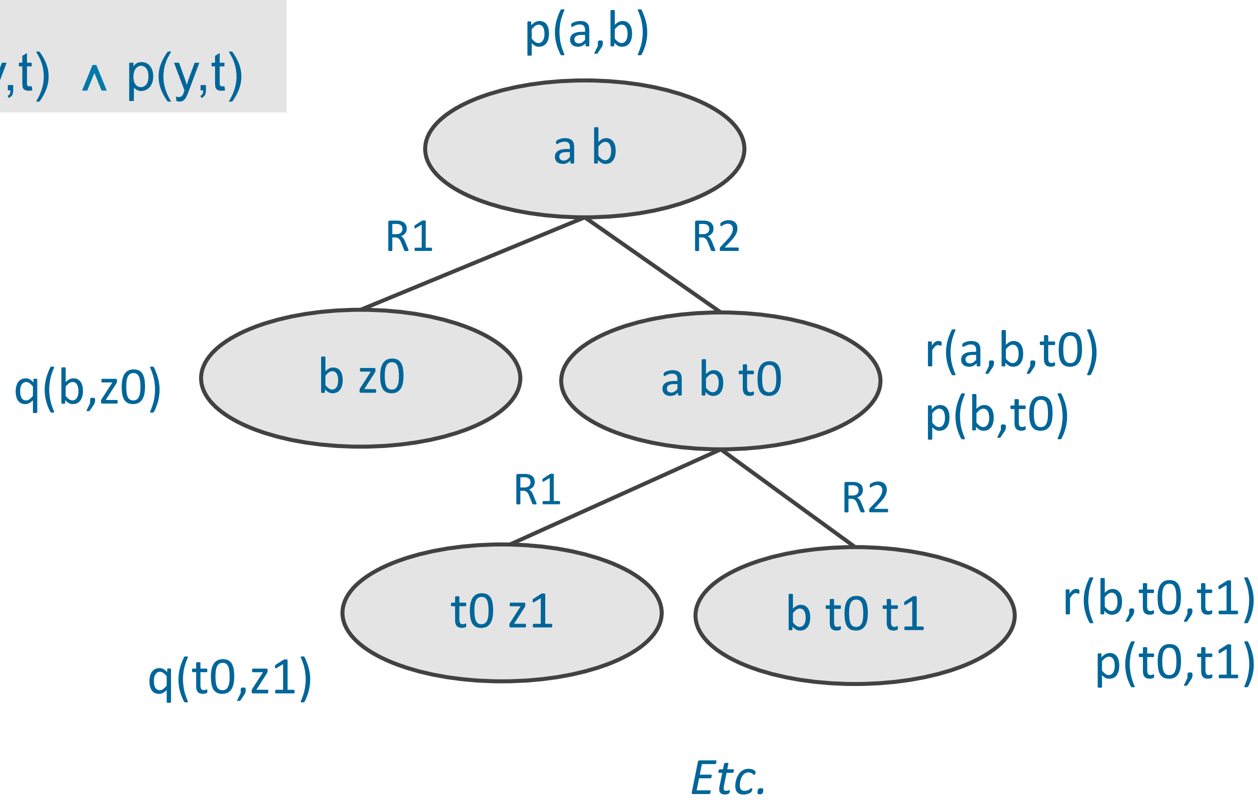
r(x,y) ∧ r(y,z) ∧ s(x,y,z) → r(y,u) ∧ r(z,u)

These classes are moreover « **greedy** bts » => a halting algorithm [Baget+ IJCAI' 11]

Greedy *bts*

R1 = p(x,y) → p(y,z)
R2 = p(x,y) ∧ q(x,z) → r(x,y,t) ∧ p(y,t)

F = p(a,b)

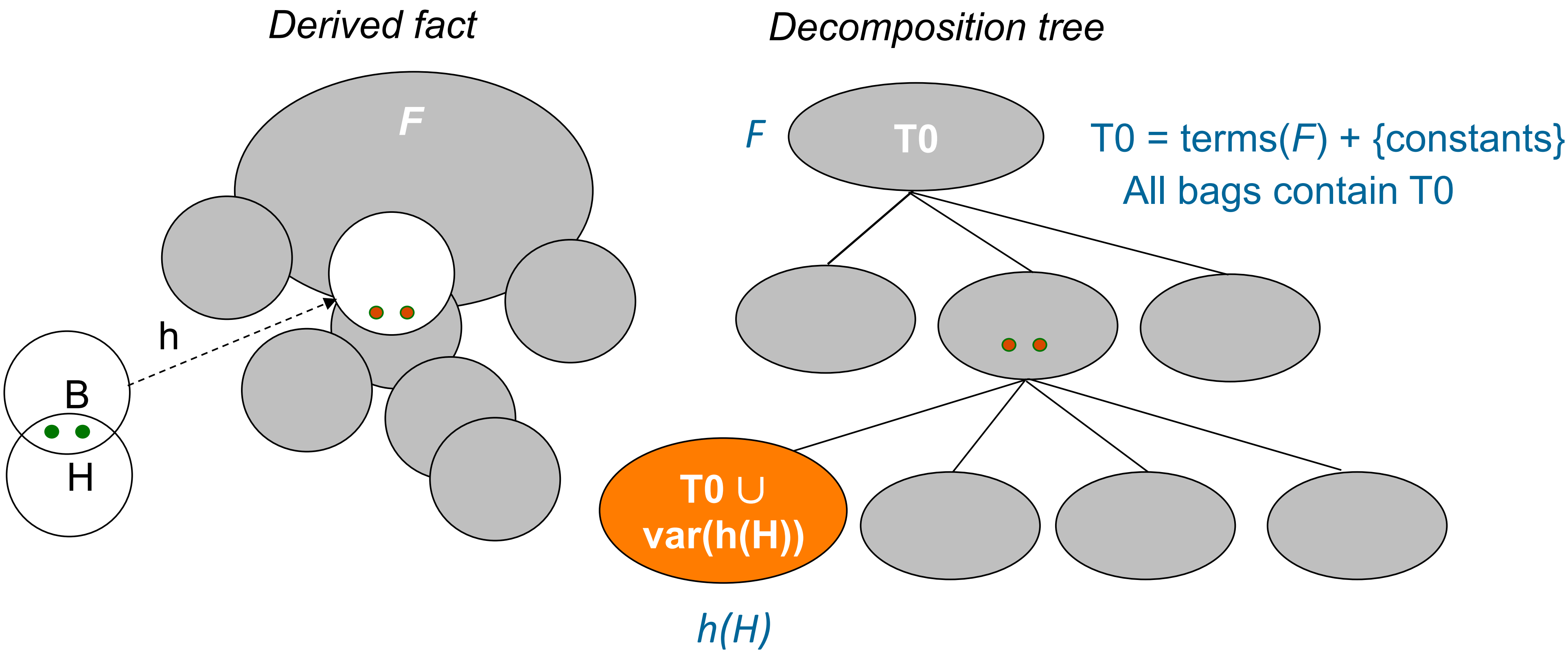


Greedy construction of a decomposition tree of the derived fact

with bounded width

The « Greedy bts » Property [Baget+ IJCAI' 11]

For any fact, for each rule application,
frontier variables not being mapped to initial terms are *jointly* mapped to variables occurring in atoms added by a single previous rule application



Main Ideas of the Algorithm for *gbts* (1)

Build a finite decomposition tree that encodes a potentially infinite fact

1. Bag pattern = { *homomorphisms* from *part of a rule body* to « current fact » that use some *terms of the bag* }

   → A rule is applicable to the current fact *iff* a bag pattern contains its body

   → FC can be performed on the decorated tree

2. Equivalence relation on bags

   Only one bag per equivalence class is developed
   The other nodes are *blocked*

Bounded number of equivalence classes → finite « full blocked tree » T*

Main Ideas of the Algorithm for *gbts* (2)

Query this finite decomposition tree

[Baget+ IJCAI 2011] Q seen as a rule « *Q → match* »

Q is entailed iff it occurs in a bag pat*tern*
i.e. Q maps by homomorphism *to atoms(T*)*

[Thomazo+ KR 2012] offline /online separation

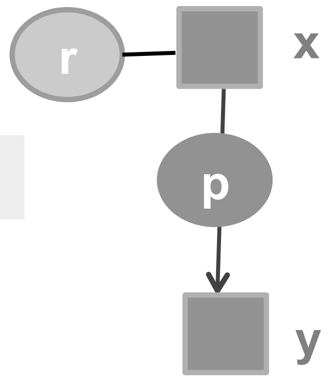
(1) compilation: tree T* built independently from *any* query

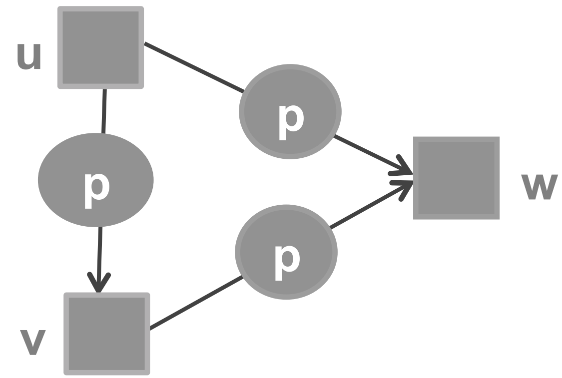
(2) querying: *any* Q is entailed iff it maps by **-homomorphism* to T*
*i.e. Q maps by homomorphism to a bounded « development » of T**

Backward Chaining: Unification Step

R = r(x) → p(x,y)

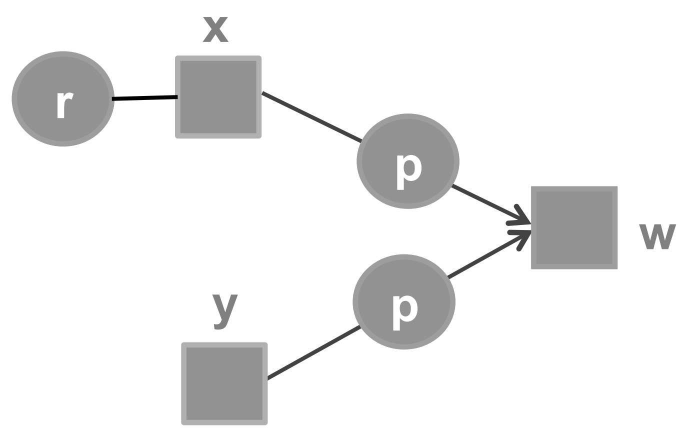
Q = p(u,v) ∧ p(u,w) ∧ p(v,w)





Atomic unification:
u → x   v → y

Q1 = r(x) ∧ p(x,w) ∧ p(y,w)



Soundness lost!

Indeed let F = Q1
saturation(F,R) ≅ F
Q does not map to F

Existentials in rule heads produce a structure that must be taken into account

Key Notion: « Piece »

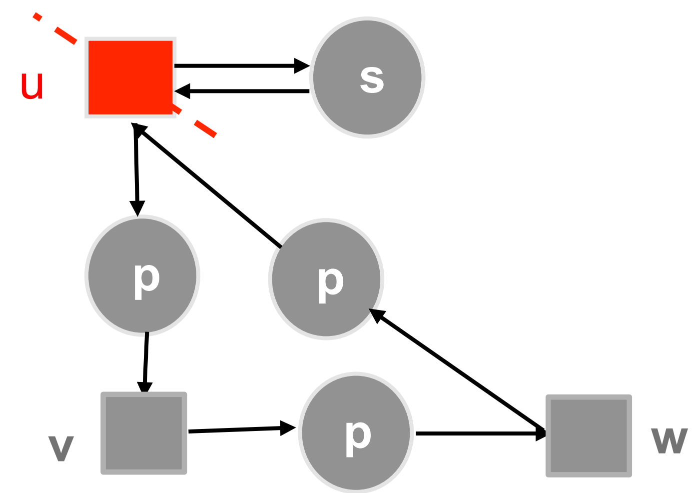
- Given a subset *T* of its variables, a set of atoms is partitioned into pieces.

A piece = all atoms linked by a « path » of variables not belonging to *T*

p(u,v) p(v,w) p(w,u) s(u,u)

T = { u }

Piece 1 = { p(u,v) p(v,w) p(w,u) }
Piece 2 = { s(u,u) }

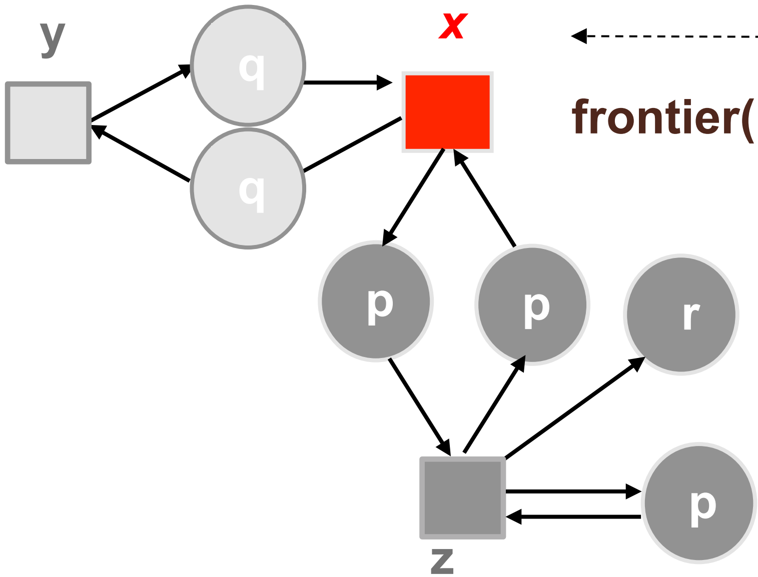


Basic notion for unification *in backward chaining,* dependency *between rules,* decomposition of *a rule into equivalent rules, …*

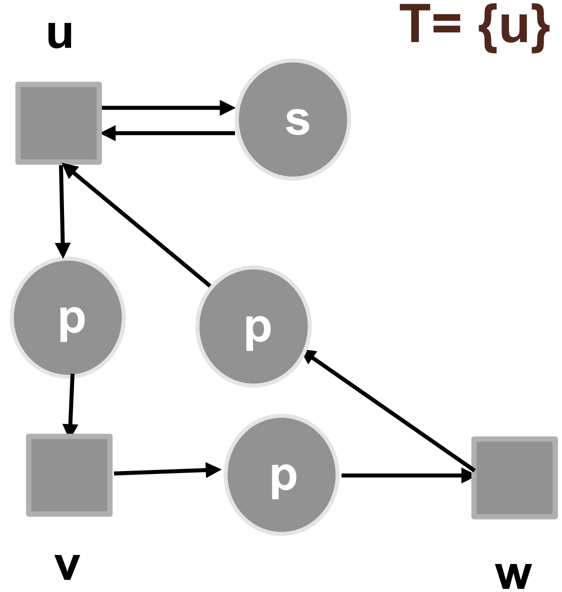
Piece-Unification (1)

R = q(x,y) ∧ q(y,x) → p(x,z) ∧ p(z,x) ∧ p(z,z) ∧ r(z)

Q = p(u,v) ∧ p(v,w) ∧ p(w,u) ∧ s(u,u)



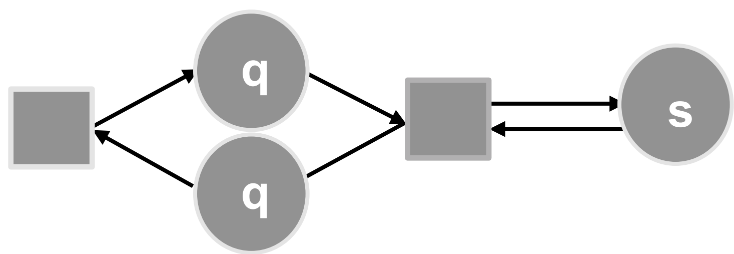
frontier(R) = {x}



T= {u}

A piece-unifier has to map at least one piece of the query to the rule head

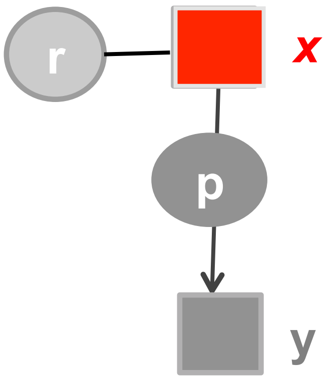
New query

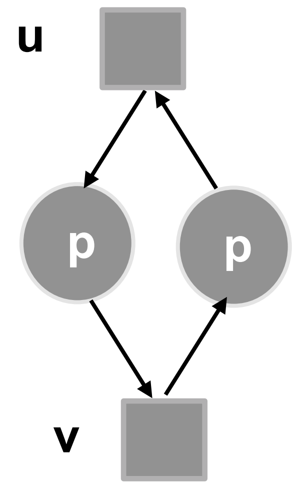


Piece-Unification (2)

R = r(x) → p(x,y)

Q = p(u,v) ∧ p(v,u)





A piece-unifier has to map at least one piece of the query to the rule head

→ *failure*

Piece-Unification (3)

Initially [Salvat M… ICCS 1996] on conceptual graphs

Piece-unifier of a query *Q* with a rule *R*:

- a substitution *s* of frontier(*R*) *by* frontier(*R*) + *constants(Q + head(R))*
- a homomorphism *h* from Q' $\subseteq$ Q to *s(head(R))*
  s.t. *Q'* is a set of *pieces* according to *s* and *h*

◆ [Salvat M… 1996]

F, $\mathcal{R}$ |= Q iff there is a sequence of piece-unifications that empties *Q*
(considering facts as rules with an empty body)

◆ [Baget+ IJCAI 2009] for *fus* existential rules

F, $\mathcal{R}$ |= Q iff one of the piece-based rewritings of Q maps to F

Outline

- Existential rules: a logic- and graph-based framework
- Decidability and algorithmic issues
  - Focus on:
    tree-shaped saturation in forward chaining
    piece-based unification in backward chaining
- A (graph) tool for combining decidable classes of rules

Union of Decidable Sets of Rules

- Next question:

  is the union of two decidable sets of rules still decidable ?

practically:

- can we safely merge several decidable ontologies ?
- can we build a decidable hybrid language from two languages whose semantics can be expressed by decidable subsets of rules ?

- Bad news:

  Almost all classes are pairwise incompatible

- Next question:

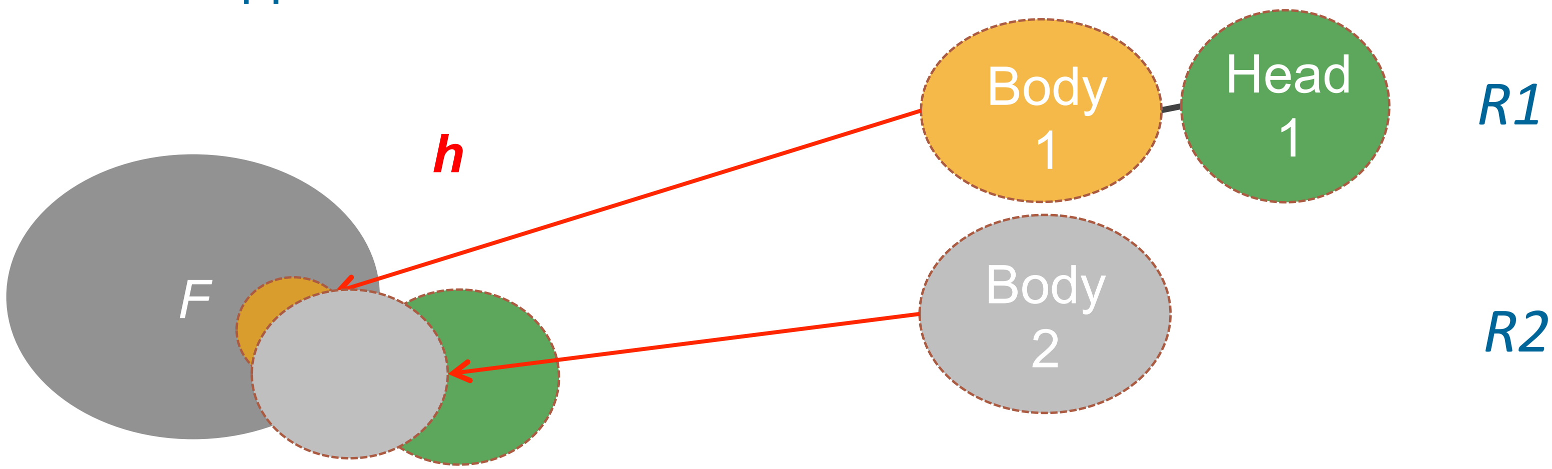
  which conditions on the interactions between rules ensure compatibility ?

A tool : the Graph of Rule Dependencies

[Baget KR 2004, Baget+ IJCAI 2009, AIJ 2011]

R2 depends on *R1* if applying *R1* may trigger a *new* application of *R2*

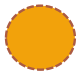
i.e., there exists a fact *F* s.t. *R1* is applicable to *F* but *R2* is not
and there is an application of *R1* to *F* leading to *F'*
s.t. *R2* is applicable to *F'*



Effective computation of dependencies with a piece-unification test:

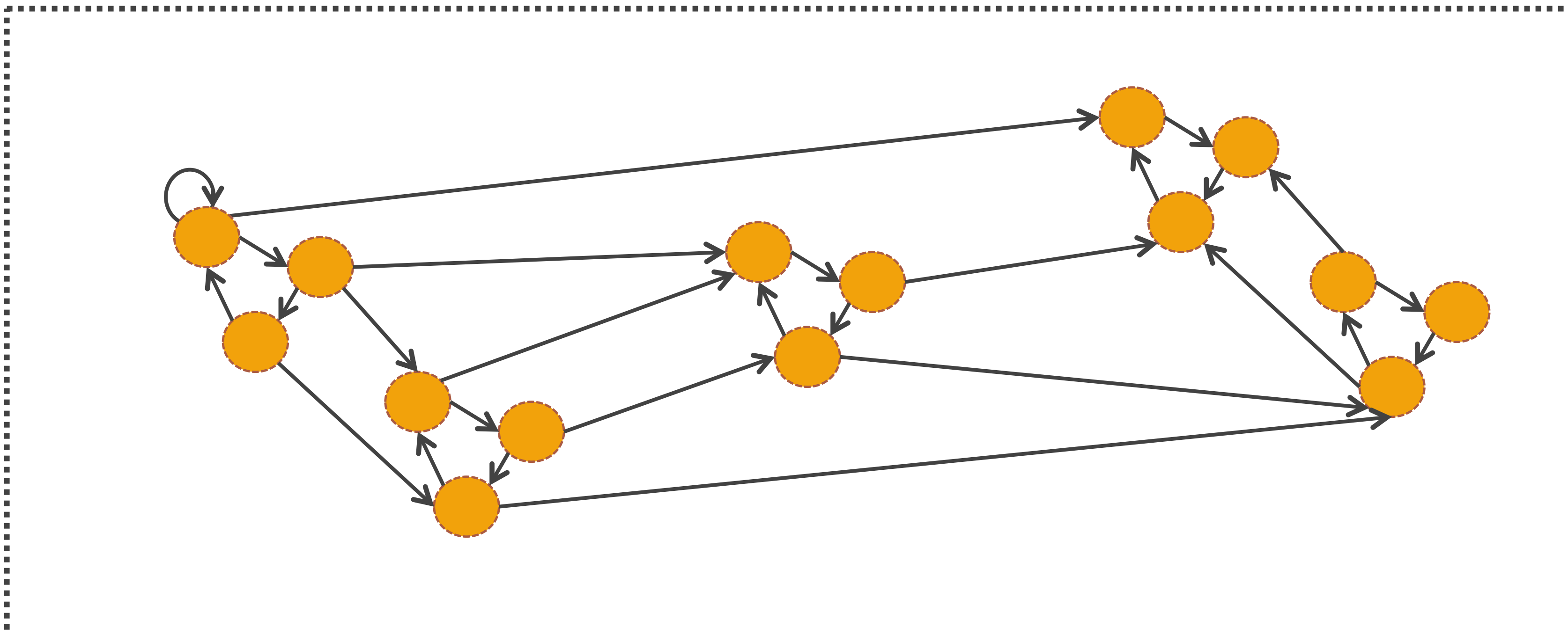
R2 depends on R1 iff there is a « piece-unifier » of body(R2) with head(R1)

Combining Decidable Classes with the Graph of Rule Dependencies



Rules

R1 ⟶ R2 : R1 « may trigger » R2 (R2 depends on R1)



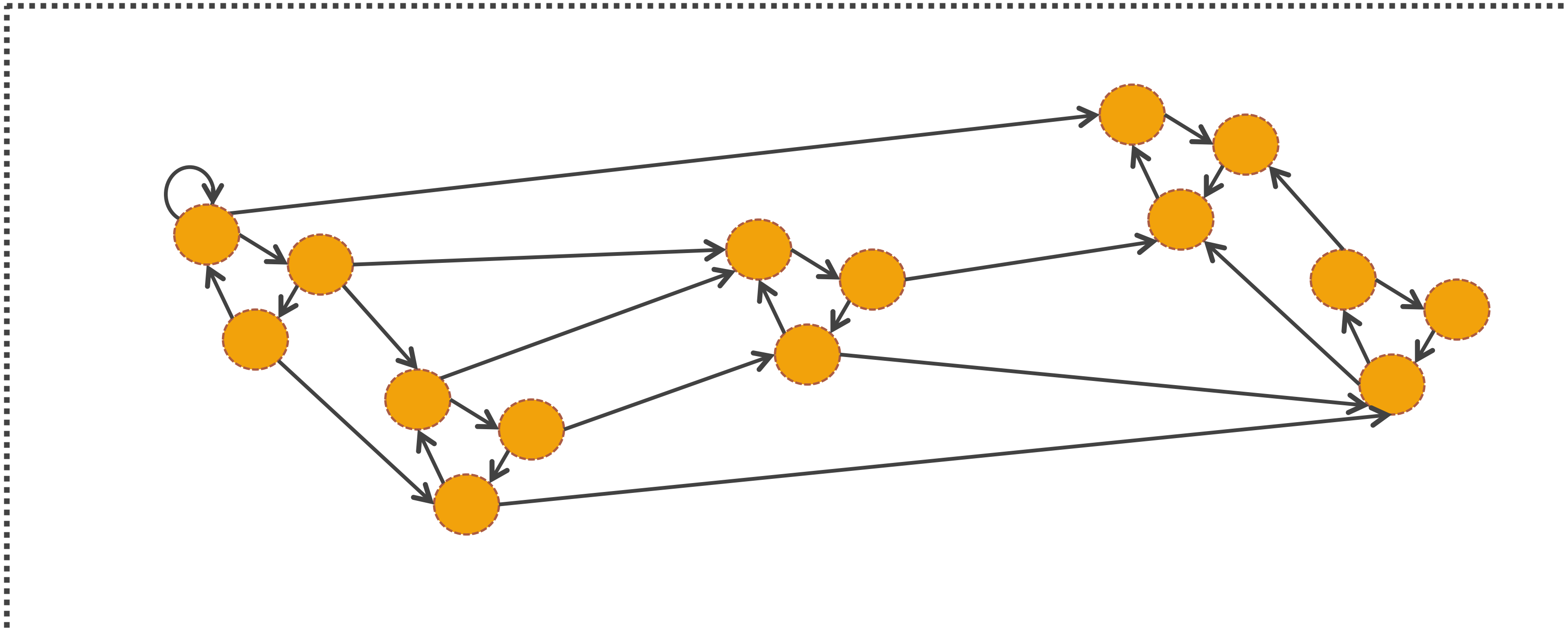
Combining Decidable Classes with the Graph of Rule Dependencies

If GRD($\mathcal{R}$) is without circuit then $\mathcal{R}$ is both *fes* (thus *bts*) and *fus*

fes = finite fact saturation

fus = finite query rewriting

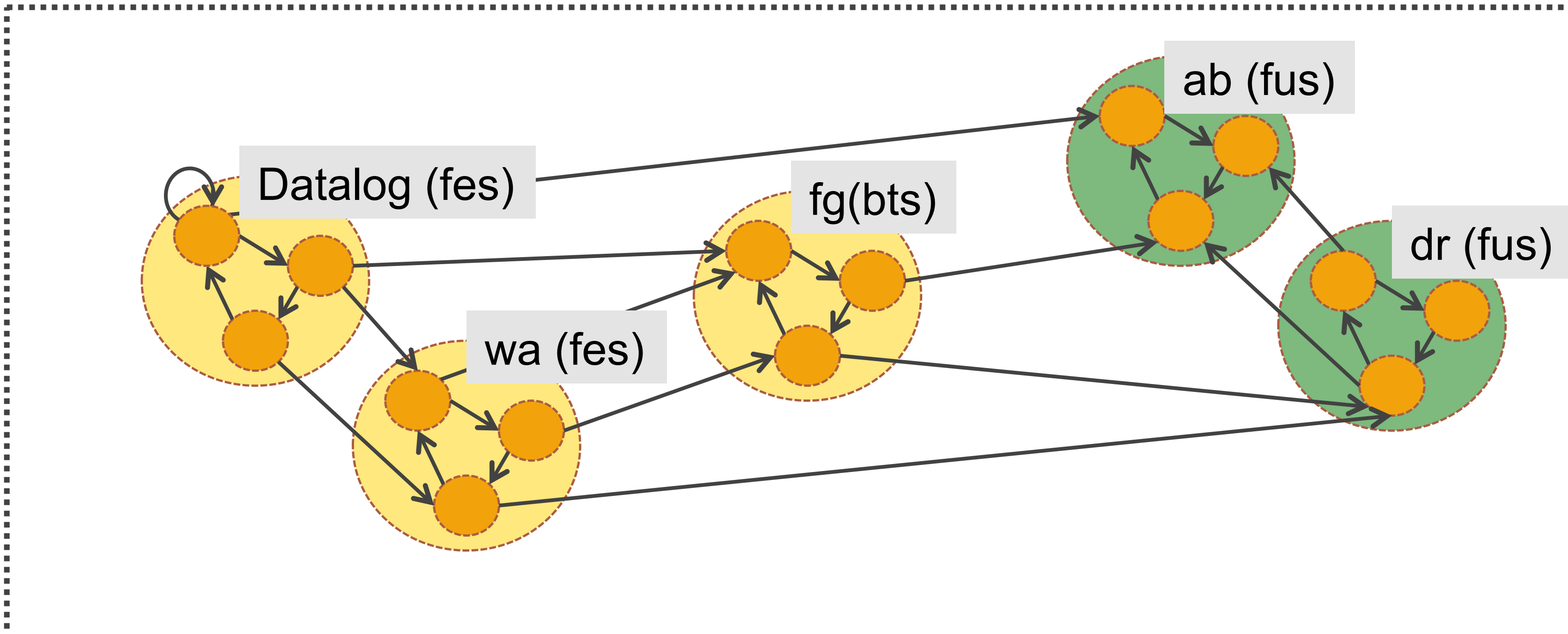
bts = (possibly infinite) tree-shaped saturation



Combining Decidable Classes with the Graph of Rule Dependencies

If all strongly connected components of GRD($\mathcal{R}$) are *fes*
then $\mathcal{R}$ is *fes* [Baget 2004]

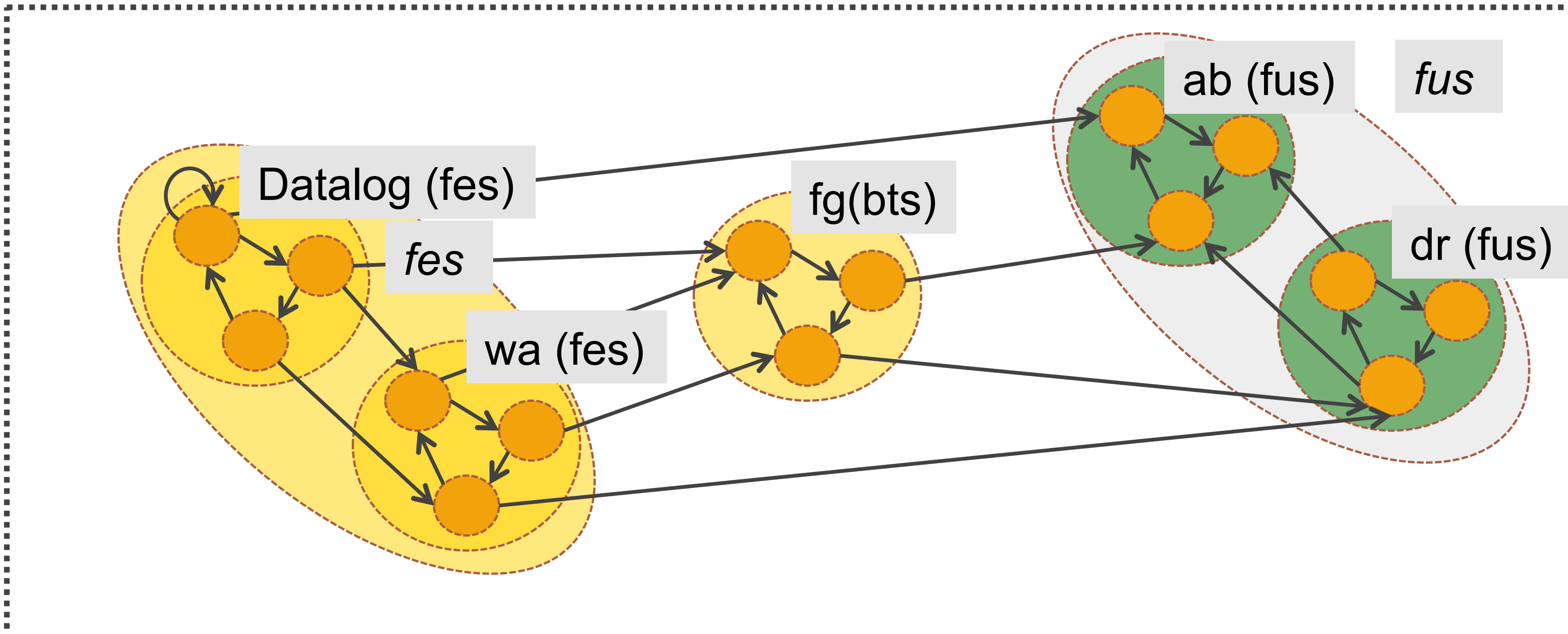
The same holds for *fus (but not for* bts*)*



Combining Decidable Classes with the Graph of Rule Dependencies

If all strongly connected components of GRD($\mathcal{R}$) are *fes*
then $\mathcal{R}$ is *fes* [Baget 2004]

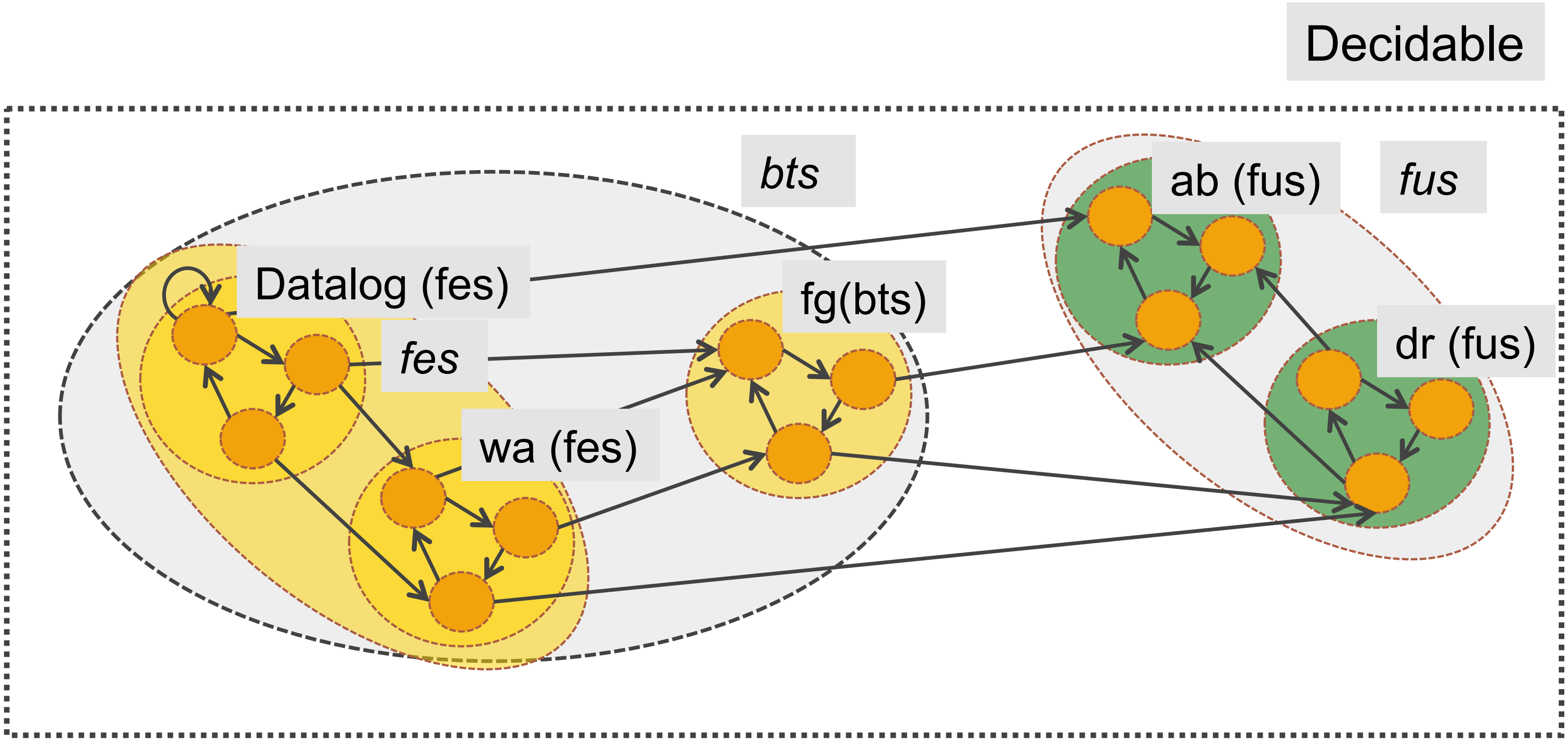
The same holds for *fus (but not for* bts)



Combining Decidable Classes with the Graph of Rule Dependencies

Let $\mathcal{R}_1 \rangle \mathcal{R}_2$ be a partition of $\mathcal{R}$ s.t. no rule of $\mathcal{R}_1$ depends on a rule of $\mathcal{R}_2$

- If $\mathcal{R}_1$ is *fes* and $\mathcal{R}_2$ is *bts*, then $\mathcal{R}$ is *bts*
- If $\mathcal{R}_1$ is *bts* and $\mathcal{R}_2$ is *fus*, then $\mathcal{R}$ is decidable



Conclusion

- An emerging rule-based framework for OBDA
  - simple
  - expressive
  - flexible
- Logic-based and Graph-based
- Currently:
  - A quite clear picture of decidable classes and their complexities
  - First implementations – often for very specific subclasses
- Next challenge: scalability