

Inference rules in DL and SWRL

G. Falquet
C. Métral

Expressivity of DL

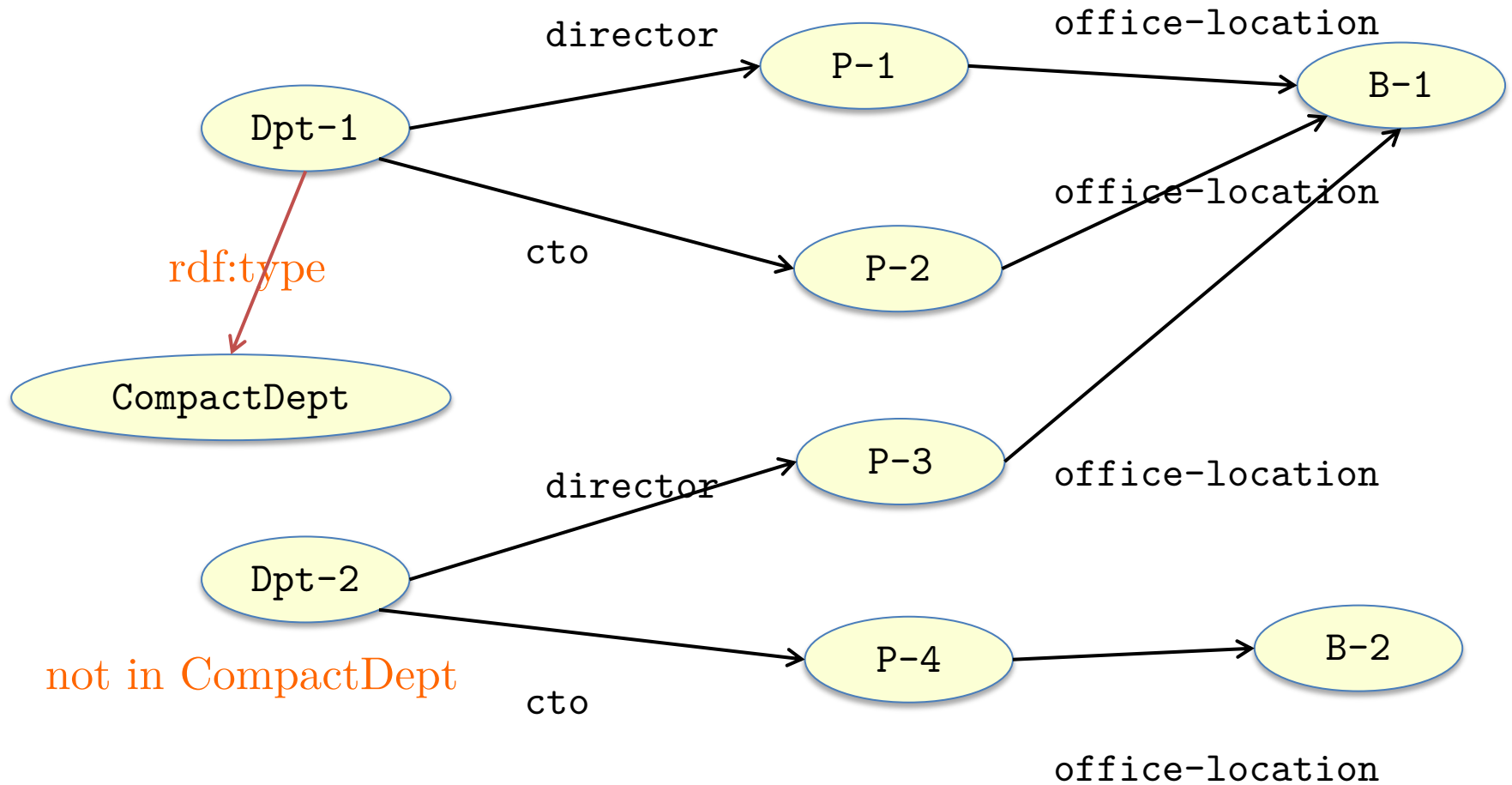
Vocabulary:

classes: Department, Employee, Building

properties: director, cto, office-location

How to define a class CompactDept to represent
departments that have their director and chief technology officer offices
located in the same building

CompactDept \equiv ???



In DL (OWL 2)

Impossible to define CompactDept in OWL-2

Many other examples cannot be defined in OWL-2

Theoretical reason: most DLs enjoy the **Tree Model Property**.

if a Tbox has a model

then it has a model that doesn't contain cycles

A fact is a consequence of a Tbox if it is true in **every** model of the Tbox

⇒ no "cyclic fact" is a consequence of a TBox.

⇒ Need for a another language to express these facts

Inference rules

Rules to produce

New **type** assertions

x is a member of class C

New **property** assertions

x is connected to y through property p

Inference rules

To produce **type** assertions

$$B_1 \wedge B_2 \wedge \cdots \wedge B_k \rightarrow C(x)$$

To produce new **property** assertions

$$B_1 \wedge B_2 \wedge \cdots \wedge B_k \rightarrow p(x, y)$$

B_i is either a class assertion $C(t)$ or a property assertion $p(u, v)$

t, u, v are either individual names or variables

$$Restaurant(x) \wedge hasMenu(x, m) \wedge contains(m, caviar) \rightarrow Expensive(x)$$

$$hasChild(x, y) \rightarrow hasParent(y, x)$$

SWRL Rules - syntax

`rule ::= antecedant -> consequent`

`antecedant ::= atom, atom, ...`

`consequent ::= atom, atom, ...`

`atom ::= description '(' i-object ')'`

`| dataRange '(' d-object ')'`

`| individualvaluedPropertyID '(' i-object i-object ')'`

`| datavaluedPropertyID '(' i-object d-object ')'`

`| sameAs '(' i-object i-object ')'`

`| differentFrom '(' i-object i-object ')'`

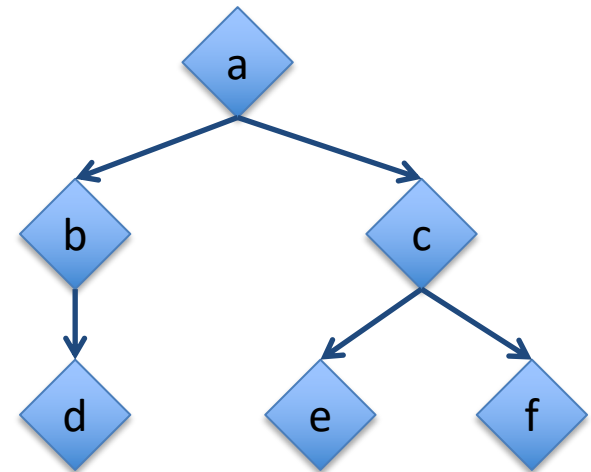
`| builtIn '(' builtinID { d-object } ')'`

`Person(?x), Person(?y), Person(?z), hasChild(?x, ?y), hasChild(?y, ?z) ->
hasGrandChild(?x, ?z)`

Interpretation

- Find all the variable bindings that satisfy the antecedent
- For each such binding the consequent must be satisfied

`hasChild(?x, ?y), hasChild(?y, ?z)`
→ `hasGrandChild(?x, ?z)`

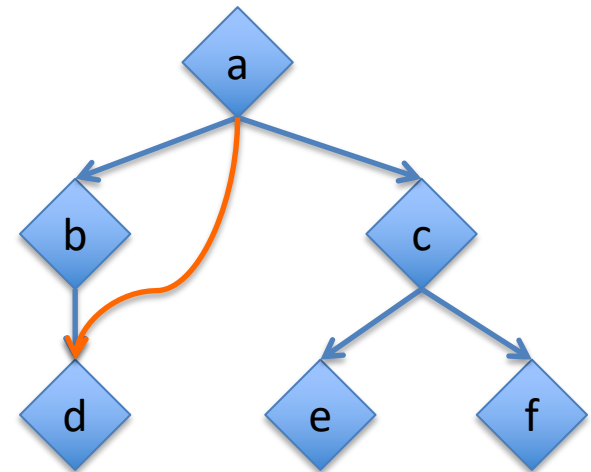


→ hasChild

Interpretation

`hasChild(?x, ?y), hasChild(?y, ?z)`
→ `hasGrandChild(?x, ?z)`

`x=a, y=b, z=d` → `hasGrandChild(a,d)`

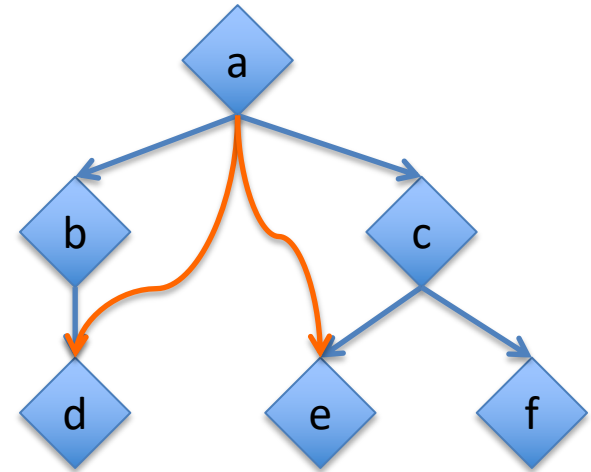


Interpretation

`hasChild(?x, ?y), hasChild(?y, ?z)`
→ `hasGrandChild(?x, ?z)`

`x=a, y=b, z=d` → `hasGrandChild(a,d)`

`x=a, y=c, z=e` → `hasGrandChild(a,e)`



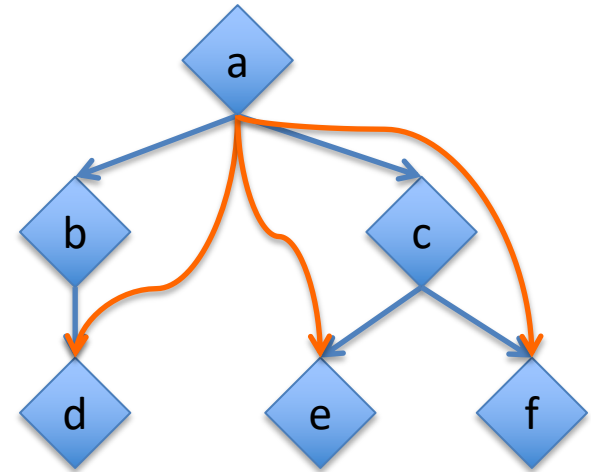
Interpretation

`hasChild(?x, ?y), hasChild(?y, ?z)`
→ `hasGrandChild(?x, ?z)`

`x=a, y=b, z=d` → `hasGrandChild(a,d)`

`x=a, y=c, z=e` → `hasGrandChild(a,e)`

`x=a, y=c, z=f` → `hasGrandChild(a,f)`



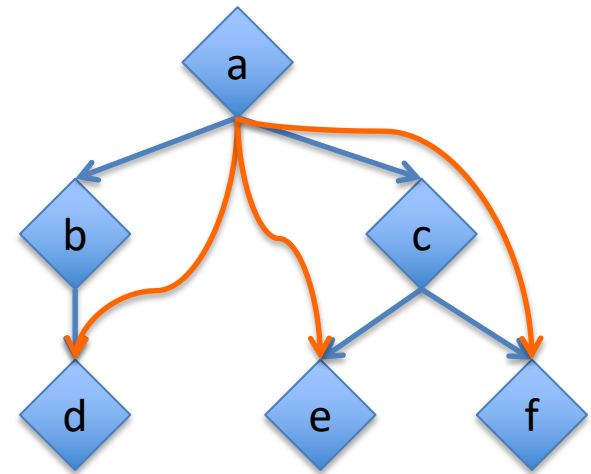
Interpretation

`hasChild(?x, ?y), hasChild(?y, ?z)`
→ `hasGrandChild(?x, ?z)`

`x=a, y=b, z=d` → `hasGrandChild(a,d)`

`x=a, y=c, z=e` → `hasGrandChild(a,e)`

`x=a, y=c, z=f` → `hasGrandChild(a,f)`



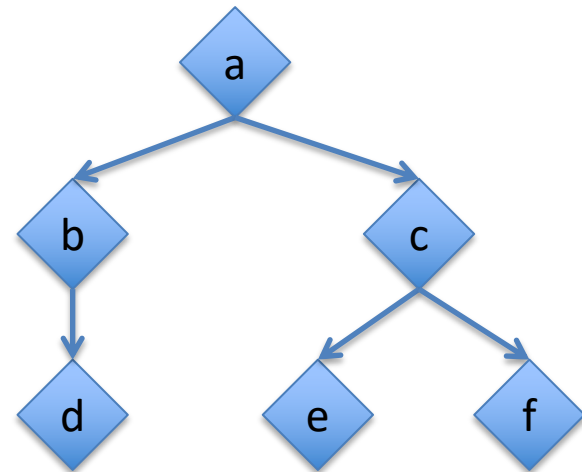
an interpretation that
satisfies the rule

DifferentFrom

Variables with different names may represent the same individual !

`hasChild(?x, ?y), hasChild(?x, ?z)`

`-> hasSibling(?y, ?z)`

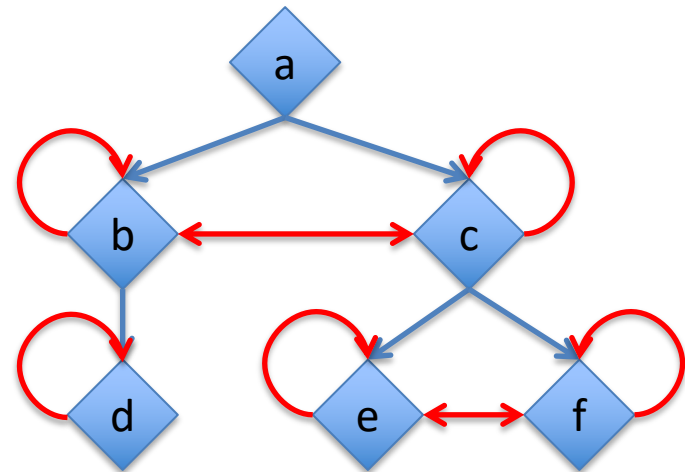


DifferentFrom

Variables with different names may represent the same individual !

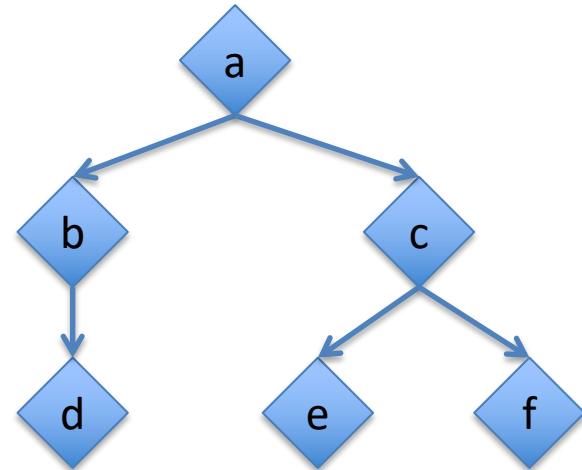
`hasChild(?x, ?y), hasChild(?x, ?z)`

`-> hasSibling(?y, ?z)`



DifferentFrom

`hasChild(?x, ?y), hasChild(?x, ?z), DifferentFrom (?y, ?z)`
`-> hasSibling(?y, ?z)`



DifferentFrom

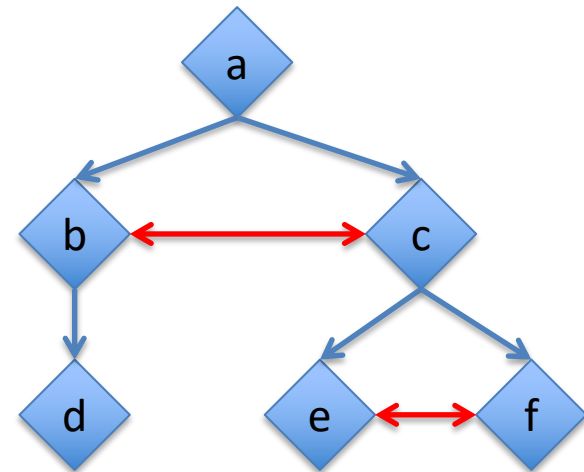
`hasChild(?x, ?y), hasChild(?x, ?z), DifferentFrom (?y, ?z)`
`-> hasSibling(?y, ?z)`



works only if

`DifferentIndividual(b,c)`

`DifferentIndividual(e,f)`



Example

`hasChild(?x, ?y) -> hasDescendant(?x, ?y)`

`hasChild(?x, ?y), hasDescendant(?y, ?z) -> hasDescendant(?x, ?z)`

The screenshot displays a Semantic Web editor interface with several panels:

- Class hierarchy (inferred):** Shows a hierarchy starting with `owl:Thing` and a subclass `Person`.
- Individual:** Lists individuals `a`, `b`, `c`, `d`, `e`, and `f`.
- Annotations:** A tab for adding annotations to the selected individual `a`.
- Rules:** A list of SWRL rules:
 - `hasChild(?x, ?y), hasChild(?y, ?z) -> hasGrandChild(?x, ?z)`
 - `hasChild(?x, ?y) -> hasDescendant(?x, ?y)`
 - `hasChild(?x, ?y), hasDescendant(?y, ?z) -> hasDescendant(?x, ?z)`
 - `hasChild(?x, ?y), hasChild(?x, ?z), DifferentFrom(?y, ?z) -> hasSibling(?y, ?z)`
- Description: a:** A panel for defining the description of individual `a`, showing the type `Person` and a list of individuals `b, c, d, e, f`.
- Property assertions: a:** A panel for defining property assertions for individual `a`, showing a list of assertions such as `hasChild c`, `hasChild b`, `hasDescendant d`, `hasDescendant e`, `hasDescendant f`, `hasDescendant b`, `hasDescendant c`, `hasGrandChild d`, `hasGrandChild e`, and `hasGrandChild f`.

DL-safe rules

Query answering for DL-axioms + rules is **undecidable**

It is **decidable** if rules are DL-safe

A rule r is called DL-safe if each variable in r occurs in a non-DL-atom in the rule body.

Practically: the variables in rules can only be bound to known individuals

Axioms:

TBox: $\text{Parent} \equiv \text{hasChild } \text{some } \text{Person}$

ABox: $\text{Parent}(a), \text{Parent}(b), \text{Parent}(c), \text{Person}(d), \text{hasChild}(a,d)$

Rule:

$\text{hasChild}(?x, ?y) \rightarrow \text{PersonWithChild}(?x)$

consequence:

$\text{PersonWithChild}(a)$

without the DL-safe restriction:

$\text{PersonWithChild}(a), \text{PersonWithChild}(b), \text{PersonWithChild}(c)$

Builtin predicates

To deal with numbers, strings, etc.

```
Rectangle(?x), hasWidthInMetres(?x, ?w), greaterThan(?w, 10)  
-> WideRectangle(?x)
```

```
Rectangle(?x), hasHeightInMetres(?x, ?h), hasWidthInMetres(?x,  
?w), greaterThan(?a, 100), multiply(?a, ?w, ?h)  
-> LargeRectangle(?x)
```

swrlb:equal
swrlb:notEqual
swrlb:lessThan
swrlb:lessThanOrEqual
swrlb:greaterThan
swrlb:greaterThanOrEqual

swrlb:add
swrlb:subtract
swrlb:multiply
swrlb:divide
swrlb:integerDivide
swrlb:mod
swrlb:pow
swrlb:unaryPlus
swrlb:unaryMinus
swrlb:abs
swrlb:ceiling
swrlb:floor
swrlb:round
swrlb:roundHalfToEven
swrlb:sin
swrlb:cos
swrlb:tan

swrlb:stringEqualIgnoreCase
swrlb:stringConcat
swrlb:substring
swrlb:stringLength
swrlb:normalizeSpace
swrlb:upperCase
swrlb:lowerCase
swrlb:translate
swrlb:contains
swrlb:containsIgnoreCase
swrlb:startsWith
swrlb:endsWith
swrlb:substringBefore
swrlb:substringAfter
swrlb:matches
swrlb:replace
swrlb:tokenize

When you don't need SWRL: DL rules

Some SWRL rules can be encoded in OWL expressions

Example

$\text{Man}(\text{?x}) \wedge \text{hasBrother}(\text{?x}, \text{?y}) \wedge \text{hasChild}(\text{?y}, \text{?z}) \rightarrow \text{Uncle}(\text{?x})$

becomes

$\text{Man} \sqcap \exists \text{hasBrother} . \exists \text{hasChild} . \top \sqsubseteq \text{Uncle}$

it's sometimes tricky ...

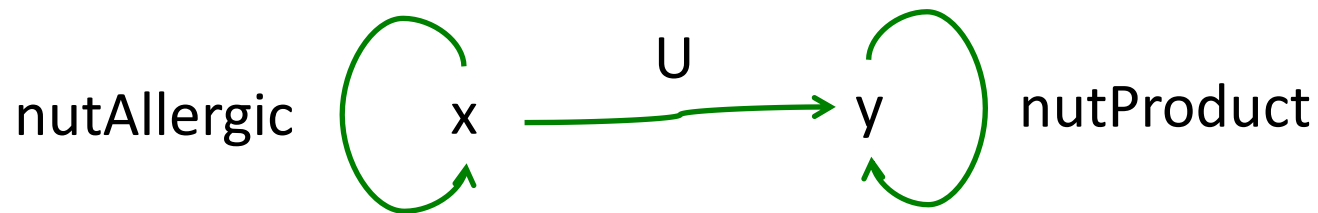
$\text{NutAllergic}(x) \wedge \text{NutProduct}(y) \rightarrow \text{dislikes}(x,y)$

$\text{NutAllergic} \equiv \exists \text{nutAllergic}.\text{Self}$

$\text{NutProduct} \equiv \exists \text{nutProduct}.\text{Self}$

$\text{nutAllergic} \circ \mathbf{U} \circ \text{nutProduct} \sqsubseteq \text{dislikes}$

\mathbf{U} = universal property ($x \mathbf{U} y$ is always true)

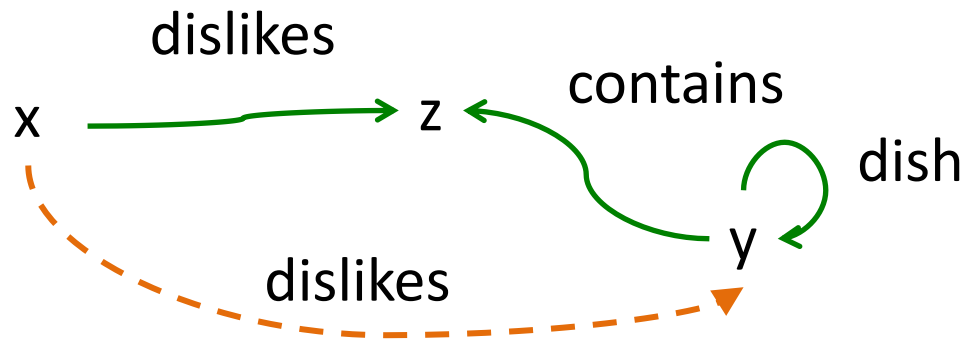


... more

$\text{dislikes}(x,z) \wedge \text{Dish}(y) \wedge \text{contains}(y,z) \rightarrow \text{dislikes}(x,y)$

becomes

- $\text{Dish} \equiv \exists \text{dish}.\text{Self}$
- $\text{dislikes} \circ \text{contains}^- \circ \text{dish} \sqsubseteq \text{dislikes}$



Rules vs. SPARQL queries

- Rules are “executed” globally
 - all rules must be satisfied simultaneously
- Rules may have interactions
 - the outcome of a rule may trigger another one
- SPARQL queries are executed independently

Simulating rules with queries

define a 'construct' query for each rule

repeat

- execute each query
- add the results to the RDF graph

until nothing new is created

$\text{parent}(\text{?x}, \text{?y}) \wedge \text{ancestor}(\text{?y}, \text{?z}) \rightarrow \text{ancestor}(\text{?x}, \text{?z})$

construct {?x ancestor ?z.}

where {?x parent ?y. ?y ancestor ?z.}

Simulating rules with queries

$\text{parent}(\text{?x}, \text{?y}) \wedge \text{ancestor}(\text{?y}, \text{?z}) \rightarrow \text{ancestor}(\text{?x}, \text{?z})$

repeat

construct {?x ancestor ?z.}

where {?x parent ?y. ?y ancestor ?z.}

until nothing new

Can be extremely inefficient

SWRL and Protégé

There is a “rule” view in Protégé

to activate it:

- menu Window -> Views -> Ontology Views -> Rules
- (a black dot appears)
- click the Class Annotation | Class Usage pane

The syntax uses ',' for the logical **and** (not '^')

$C(?x), p(?x, ?y) \rightarrow D(?y)$

SameAs and **DifferentFrom** must start with an uppercase letter.

SWRL inference in Protégé

- Pellet and HermiT support SWRL inference
 - simply run the reasoner to perform swrl inference
 - menu Reasoner -> Start Reasoner or Classify or Synchronize
- HermiT does not support the builtin atoms: add, multiply, lessThan, ... (=> hardly usable)
- Reasoners make the rules DL-safe by binding the variable only to explicitly asserted individuals
 - => reasoning is not complete with respect to the axioms

SWRL and Protégé bugs

Protégé 4.0

- the rule editor does not accept builtin predicates (add, multiply, lessThan, ...)

Protégé 4.1 – 4.3

- does not show the inferred data properties (the inferences actually takes place but the interface does not show them)