

# Capturing Quantified Constraints in FOL, through Interaction with a Relationship Graph

Prof. Peter M.D. Gray

*Dept. Computing Science*

*Univ of Aberdeen, SCOTLAND*

Dr. Graham J.L. Kemp

*Dept. Computer Science and Eng.,*

*Chalmers Univ. of Technology,*

*Göteborg, SWEDEN*

# Themes:

1. Relating Constraints to an Entity-Relationship Diagram with SubClasses (like UML Class Diag.)
2. Extending Constraints via Derived Relationships
3. Capturing Implications via Nested Loops
4. Capturing Existential Quantifiers as well
5. Testing and Revising Constraints against Remote Data

# Relating Constraints to an Entity-Relationship Diagram

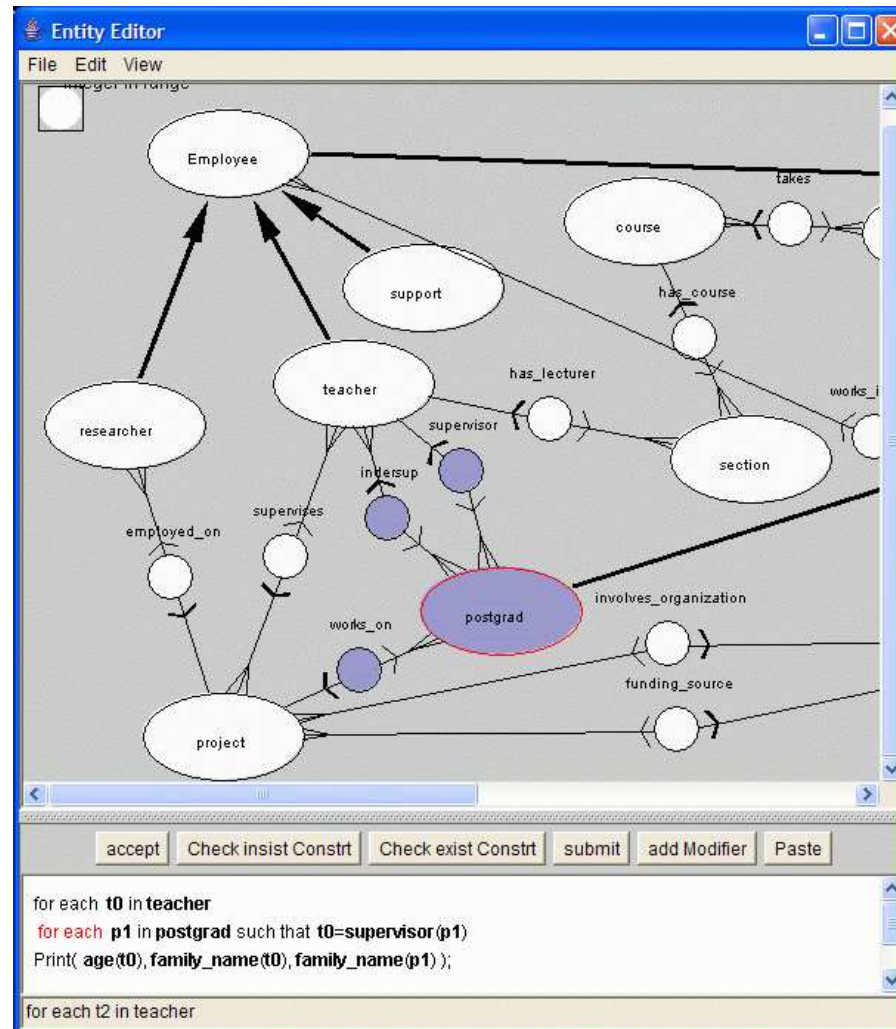
- We **START** from an ER DIAGRAM showing:  
*Entity Classes:* postgrad teacher  
*Relationship:* A supervises B  $\iff$  B pupilof A  
*Quantifiers:* Universal :-  $(\forall p:\text{postgrad})$
- We **GENERATE** a well-formed readable **CONSTRAINT** by point and click: constrain  
*each* p in postgrad so that *each* t in  
supervisor(p) *has* age(t) > 21;
- *N.B. Functional Syntax:* TREAT supervisor(p) AS  
p.supervisor in Java or SQL

# Relating Constraints to an Entity-Relationship Diagram

- We **CAPTURE** this in Predicate Logic (encoded in XML-RDFS):

$$(\forall p) \text{ postgrad}(p) \Rightarrow ((\forall a,t) \text{ supervises}(t,p) \wedge \text{age}(t,a) \Rightarrow a > 21)$$

# A Daplex query



# Why not learn the Constraints from Data?

- This is very hard for complex constraints with mixed quantifiers and several relationships.
- Scientists often have rich background knowledge about their data, and its experimental conditions, that may not be illustrated in the sample data
- We can't just parse Natural Language statements – often ambiguous or unclear
- Instead we make the scientists structure how they express the constraint by looking at the diagram and clicking on **relationships** in it.

# Adding Derived Relationships

- Scientists sometimes store relationships in non-standard fashion.
- e.g. no explicit relationship between *BIND\_Pathway* and *BIND\_Interaction*.
- Consider the *pathway* attribute for the “epidermal growth factor”:  
{116, 118, 145, 148, 167, 1444, 1448, 1451}
- these integers are identifiers (iid), of instances of *BIND\_Interaction*.
- We derive the relationship as a *Formula*.

# Defining Derived Relationships

We define a function `pathway_interactions` relating pathway objects `p` to sets of interaction objects `i`:

```
define pathway_interactions
  (p in BIND_Pathway) ->>
    BIND_Interaction
```

```
  i in BIND_Interaction such that
    iid(i) in pathway(p);
```

The relationship is shown as a *labelled arc* in the ER diagram, just like other stored relns. It will be clicked on to generate constraints

# Capturing Implications via Nested Loops

```
for each B0 in BIND_Pathway
  for each B1 in BIND_Interaction
    such that
      B1 in pathway_interactions(B0)
    print(pid(B0), iid(B1));
```

- Each variable (B0,B1,...)in the constraint generates a Quantified Var in the FOL version.
- The variable ranges over objects in an Entity Class or Subclass, or over an Integer subrange.
- The variable is governed by a *for each* construct, filtered by a *such that* condition.

# Capturing Implications via Nested Loops

- Variables come into scope on successive lines, forming recognisable Nested Loops.
- A *such that* condition like  $V1 \text{ in } \text{rela}(V0)$  serves to restrict variable  $V1$  to values related to  $V0$  (in an outer loop) through relationship *rela* (arc in ER diagram).
- If we put a *print* command in innermost loop, it will print associated values satisfying join conditions, as in an SQL Query.

# Representing Implication by "Insist"

```
for each B0 in BIND_Pathway
  for each B1 in BIND_Interaction
    such that
      B1 in pathway_interactions(B0)
      insist pid(B0) > iid(B1)
      print(pid(B0), iid(B1));
```

- Instead of the Declarative "Implies" we use the Operational verb "Insist" to test values in innermost loop.
- If the Implication is logically true, then the Insist test will succeed (i.e. fail to find counterexample values).

# Representing Implication by "Insist"

- The print statement is there to print values of attributes in any counterexamples.
- Scientists and Engineers find nested loops more intuitive than FOL notation, without losing FOL precision.
- Following "insist" comes a small empty clickable box, which expands into a boolean expression builder. This expression is held invariantly True.
- If we put a *print* command in innermost loop, it will print associated values satisfying join conditions, as in an SQL Query.

# Capturing Existential Quantifiers as well

*Each Molecular Complex must be related through a BIND\_object to interactions on its interaction list.*

$$\begin{aligned} & (\forall B0) \text{Molecular\_Complex}(B0) \Rightarrow (\forall B1) \\ & \text{complex\_objects}(B0,B1) \Rightarrow \\ & ((\exists B2,I,L) \text{object\_interactions}(B1,B2) \wedge \\ & \text{iid}(B2,I) \wedge \text{interaction\_list}(B0,L) \\ & \wedge I \text{ in } L) \end{aligned}$$

- Observation shows that most Existential Quantifiers in constraints come innermost, dependent on outer  $\forall$  ones.

# Capturing Existential Quantifiers as well

- Unlike "Insist", they introduce an extra variable, through a relationship, which must also satisfy an invariant.
- Thus we use a Radio Button "Insist Exist" as alternative to "Insist". It is only enabled when clicking on an ER relationship.
- We believe bracketed Existentials used elsewhere are too easily misconstrued!

# Insist Exist Example

```
for each B0 in BIND_Molecular_Complex
  for each B1 in BIND_object
    such that B1 in complex_objects(B0)
      insist exist B2 in BIND_Interaction
        such that B2 in object_interactions(B1)
          and iid(B2) in interaction_list(B0);
```

This is of the form:

**for each** ...

**for each** e1 in entity such that ... and ...

**insist exist** e2 in entityclass2

such that e2 in rel(e1) and (predicate...);

# Query for Counterexamples

We easily generate a query for counterexamples:  
for each ...

for each e1 in entity such that ... and ...  
and

```
(no e2 in entityclass2 such that  
e2 in rel(e1) and (<predicate>) exists)  
print(...);
```

# Testing Constraints vs Remote Data

- Queries for counterexamples can be sent to remote databases via *Mediator*.
- *print* command in inner loop prints a *row* of values from any objects which (together) *violate* the constraint; rows form a table.
- Table values are *mouse active* (Java Swing).
- We click on these values in a row to *generate* a well-formed condition which *excludes* such values from the constraint.
- This is a rudimentary form of *case-based adaptation*

# Counterexamples in table

Database Response

Retrieved : 91Rows

pid(B0)	description(B0)	short_label(B1)	user_id(B1)
13042	The activation and deactivation the heterotri...	G_beta_gamma_e	0
13042	The activation and deactivation the heterotri...	G_alpha_q/GTP	0
13038	Epidermal growth factor (EGF) pathway	EGF	0
13038	Epidermal growth factor (EGF) pathway	EGFR	0
13038	Epidermal growth factor (EGF) pathway	EGF_EGFR	0
13038	Epidermal growth factor (EGF) pathway	(EGF_EGFR) dimer complex	0
13038	Epidermal growth factor (EGF) pathway	ATP	0
13038	Epidermal growth factor (EGF) pathway	(EGF_EGFR) dimer_ATP	0

```
for each B0 in BIND_Pathway
  for each B1 in BIND_object such that B1 in pathway_objects(B0)
    insist user_id(B1)>0
  Print(pid(B0), description(B0), short_label(B1), user_id(B1));
```

# Example Constraint Modifier

- e.g. if *pid* column contains value 13042 and *short\_label* value in same row is "GTP" we wish to exclude such cases.
- this generates the negated conjunction:

```
for each B0 in BIND_Pathway
  for each B1 in BIND_object such that
```

```
...
```

```
FILTER ADDED
```

```
and not( (pid(B0)=13042) and
          (short_label(B1)="GTP"
```

# Modifying Constraints with Generated Filters

- Scientists needn't struggle with boolean expression syntax for filters, or where to add them.
- Clicking on several values in a *row* will *and* the conditions in a Filter.
- Clicking on values in a *column* will *or* the conditions - i.e. accept any one of them.
- values needn't be adjacent.

# Conclusions

- Knowledge Capture should pay more attention to *Relationships* in a Data Model – ER Diagrams help!
- Clicking on Relationships to generate *Nested Loops* helps people build up Constraints visually and systematically.
- We must be able to *Derive extra relationships* by Formulae, to use in Constraints.
- We should capture Constraints in *Predicate Logic*, independent of *Storage* :Database Tables, Arrays etc...
- FOL RuleML and Protégé PAL allow such Constraints in FOL to be transformed and interchanged over the Semantic Web. W3C RIF group carries this on.