

Comprehensions for Integrity Constraints

An Integrity constraint can be considered as a **query that is always true (Invariant)**.

Here is a constraint in P/FDM syntax:

```
constrain each t in seniortutor
  so that each s in advisees(t)
    has grade(s) > 60;
```

Here is a Comprehension to compute the **set of exceptions** - hopefully the Empty Set.

```
[ t | t <- seniortutor; s <- advisees(t) ;
  not (grade(s) > 60) ]
```

Finally, here is the Constraint in Logic Form:

$$(\forall t) \text{ seniortutor}(t) \Rightarrow ((\forall s, g) \text{ advisee}(t, s) \wedge \text{grade}(s, g) \Rightarrow g > 60)$$

Existential Query as Comprehension

Daplex P/FDM(Functional Data Model):

```
for each u in undergrad such that
    some c in takes(u) has level(c) > 3
print(name(u));
```

ZF-expression (Set-expression, Formal):

```
[ name(u) | u <- undergrad;
    Exists [ c | c <- takes(u);
            level(c) > 3 ] ]
```

AMOSQL (Object-Relational):

```
SELECT name(u)
FROM    undergrad u
WHERE   some(
    SELECT c
    FROM    course c
    WHERE   c = takes(u)
            AND level(c) > 3
    );
```

The query returns the name of all undergrads taking at least one course with level above 3.

Comprehensions in Existential Constraints

An existential constraint in Colan syntax:

```
constrain each t in seniortutor
  so that some s in advisees(t)
  has grade(s) > 60;
```

Here is a Comprehension to compute the **set of exceptions** - hopefully the Empty Set.

```
[ t | t <- seniortutor;
  [s <- advisees(t) ;(grade(s) > 60)] = []]
```

Finally, here is the Constraint in Logic Form:

$$(\forall t) \text{seniortutor}(t) \Rightarrow ((\exists s, g) \text{advisee}(t, s) \wedge \text{grade}(s, g) \wedge g > 60)$$

Comprehension Equivalences

DAPLEX:

```
constrain each x1 in obj1 such that pred1(x1)
      each x2 in f(x1) such that pred2(x2) and
      pred12(x1,x2)
to have cpred(x1,x2);
```

MIRANDA (using LISP-style function application):

```
(foldr and True
  [ (cpred x1 x2) | x1 <- obj1; (pred1 x1);
    x2 <- (f x1); (pred2 x2);
    (pred12 x1 x2) ] ) = True
```

Where

```
(foldr and True [a, b, ... z]) =
  a and b and ... z and True
```

Existential Comprehensions

We can extend this to cover existential quantifiers, for example, where they appear on the right of the `to have` construct:

```
constrain each x1 in obj1 such that pred1(x1)
          each x2 in f(x1) such that pred2(x2) and
                                     pred12(x1,x2)
          to have some x3 in f2(x2) has cpred3(x1,x2,x3);
```

We just replace `(cpred x1 x2)` in the original Miranda formulation by:

```
(foldr or False [(cpred3 x1 x2 x3) | x3 <- (f2 x2)])
```

If we replace **some** `x3` in `f2(x2)` by **no** `x3` in `f2(x2)` we then replace `(cpred x1 x2)` by:

```
(foldr or False
 [(cpred2 x1 x2 x3) | x3 <- (f2 x2)]) = False
```

```
or: [x3 | x3 <- (f2 x2) ; (cpred3 x1 x2 x3)] = []
```

Colan and E-R Diagrams

An ER diagram (or UML class diagram) is commonly thought of as just a *visualisation* of the types of entities and relationships in a database (or an O-O application) but we are using it as an *operational basis* for:

- Building queries by *point and click* on the diagram to create a *well-formed* Daplex query *incrementally*.
- *Clicking* on an entity name in the partly-formed *query* highlights it in the *diagram*, together with possible *navigation steps* from it.
- *Submitting* queries to local or remote databases, and using data values from the *result tables* to *copy and paste* in extra conditions to focus the query.

- Using a novel **insist** construct to *build Constraints interactively*, like queries.
- Interactively testing Constraints by *generating queries to search for violations*. Saving valid constraints. Thus constraints are *machine verifiable*.

Colan and Predicate Logic

Colan is as expressive as the subset of first-order logic that is useful for expressing integrity constraints: namely, *range-restricted* constraints.

This class includes all FOL expressions in which each variable is constrained to be a member of some *finite set of values*. Its use avoids the classic problem of *safety* of certain expressions.

Thus we have a formalism which gives us a *precise denotation* for constraints but it does not force us to *evaluate them* as integrity checks!

Constraints as Specifications

The constraint expresses a formula of logic which is true when applied to instances in a database. It can also apply to instances in an *empty solution database*, yet to be populated with solutions constructed by a *solver process* (KRAFT project). Here it is behaving more like a *specification* than as an integrity check.

Thus constraints can be passed as a form of *mobile knowledge* between agents and processes; they are *no longer stuck* inside a piece of database software!

Functional Programming freed Functions to move around *inside an executing program*. This development frees Constraints to move around the Semantic Web, into *different execution environments*.

Colan and Semantic Web

Colan is ideal for Semantic Web applications, since it is *not dependent on human inspection* — it can be

- *proof-checked* mechanically,
- *transformed* by symbol manipulation,
- *sent* to a *remote constraint solver*.

Given a standardised *interchange format* such as XML or RDF, data and attached constraints can be:

- *gathered together* from multiple sources,
- *checked* for compatibility,
- used to *derive or infer* new information.

RDFS constraint interchange format

Because P/FDM is an extended ER model, it maps easily onto the *RDF Schema* specification, so we have developed an interchange format *independent* of both Platform and DataBase Format!. It depends solely on **Predicate Logic** and **Functional Data Model**.

Constraint's Role in Semantic Data Models

The original motivation for introducing Constraints into P/FDM was to enrich the capability of the data model in *capturing semantics*, just like the Semantic Web vision of making information self-describing for *maximal machine processing*.

"Data models are more than type systems (which they resemble syntactically) because they also represent constraints on the data" – *Buneman*. Thus, although the original FDM was somewhat lacking in semantics, we have made up for that by introducing constraints.

In doing this, we have capitalised on a cardinal virtue of the FDM, that it enables one to make *well-formed mathematically precise computations over data* related by an ER diagram. Thus we based Colan on these computations, extending it to use the *expression syntax* of P/FDM's Daplex query language.

Crucially, because the expressions are *fully quantified* and *referentially transparent*, it is easy to *move them into other contexts* and transform them in ways which preserve their semantics. This would not be so if our expression of data semantics had been confined to a *diagrammatic notation* used by many others.