Inverse relationships and “rewriting”

Daplex:

```
for each s in section
    for each c in has_course(s) such that level(c) = 4
print(section_name(s), code(c));
```

List comprehension 1:

```
[ section_name(s), code(c) |    s ← section;
                                c ← has_course(s);
                                level(c) = 4 ]
```

List comprehension 2:

```
[ section_name(s), code(c) |    c ← course;
                                level(c) = 4 ;
                                s ← has_course_inv(c) ]
```

Keys and optimisation

Daplex:

for each c in course such that code(c) = "C_331"
print(units(c));

List comprehension 1:

[units(c) | c ← course; code(c) = "C_331"]

Prolog 1:

```
getentity(course,C), getfnval(code,[C],'C_331')
```

List comprehension 2:

[units(c) | c ← course with key = ["C_331"]]

Prolog 2:

```
getentity(course,['C_331'],C)
```

Optimisation example

Daplex:

```
for each u in undergrad such that year(u) = 3
    for each c in takes(u) such that code(c) = "C_331"
print(surname(u));
```

List comprehension 1:

```
[ surname(u) |     u ← undergrad; year(u) = 3;
                   c ← takes(u); code(c) = "C_331" ]
```

List comprehension 2:

```
[ surname(u) |     c ← course with key = ["C_331"];
                   u ← takes_inv(c); year(u) = 3 ]
```

Factoring out common sub-expressions

Daplex:

```
for each s in section
print(section_name(s),
   forename(has_lecturer(s)), surname(has_lecturer(s)));
```

Prolog 1:

```
getentity(section,S), getfnval(section_name,[S],SName),
getfnval(has_lecturer,[S],L1), getfnval(forename,[L1],Forename),
getfnval(has_lecturer,[S],L2), getfnval(surname,[L2],Surname)
```

Prolog 2:

```
getentity(section,S), getfnval(section_name,[S],SName),
getfnval(has_lecturer,[S],L),
getfnval(forename,[L],Forename), getfnval(surname,[L],Surname)
```

Semantic optimisation — ship example (1)

```
declare ship ->> entity
  declare name(ship) -> string
  declare size(ship) -> integer
  key_of ship is name

declare aircraft_carrier ->> ship
  declare num_aircraft(aircraft_carrier) -> integer;
```

```
with common n in integer
rewrite     s in ship such that size(s) > n
into        if n>10000
                then a in aircraft_carrier such that (size(a)>n) as ship
                else s1 in ship such that size(s1) > n;
```

```
for each s in ship such that size(s) > 12000
print(name(s));
```

Semantic optimisation — ship example (2)

```
run :-
        (   (   12000>10000 ->
                getentity(aircraft_carrier, A),
                B=A,
                relative(ship, B, C),
                D=C,
                getfnval(size, [A], E),
                E>12000
            ;   getentity(ship, F),
                G=F,
                D=G,
                getfnval(size, [F], H),
                H>12000
            ),
            getfnval(name, [D], I),
            write_result_row([I]),
            fail
        ;   true
        ).
```

Semantic optimisation — ship example (3)

```
[generate(ship,A),
 restrict(size,[ship],[A],B),
 expression([],[B,C],expr(>,B,C))],

[generate_subquery(ship,A,
  if([expression([],[C],expr(>,C,10000))]),
    [generate(aircraft_carrier,D),
     expression(E,[D],expr(=,E,D)),
     restrict(size,[aircraft_carrier],[E],F),
     expression([],[F,C],expr(>,F,C)),
     expression(G,[E],expr(=,G,E)),
     expression(H,[G],relative(ship,G,H)),
     expression(A,[H],expr(=,A,H))],
    [generate(ship,I),
     expression(J,[I],expr(=,J,I)),
     restrict(size,[ship],[J],K),
     expression([],[K,C],expr(>,K,C)),
     expression(L,[J],expr(=,L,J)),
     expression(A,[L],expr(=,A,L))])]
```

Semantic optimisation — ship example (4)

```
[generate(ship,A),
 restrict(size,[ship],[A],B),
 expression([],[B,C],expr(<,C,B))],

[generate_subquery(ship,A,
  if([expression([],[C],expr(>,C,10000))]),
    [generate(aircraft_carrier,D),
     expression(E,[D],expr(=,E,D)),
     restrict(size,[aircraft_carrier],[E],F),
     expression([],[F,C],expr(>,F,C)),
     expression(G,[E],expr(=,G,E)),
     expression(H,[G],relative(ship,G,H)),
     expression(A,[H],expr(=,A,H))],
    [generate(ship,I),
     expression(J,[I],expr(=,J,I)),
     restrict(size,[ship],[J],K),
     expression([],[K,C],expr(>,K,C)),
     expression(L,[J],expr(=,L,J)),
     expression(A,[L],expr(=,A,L))])])).
```