MINT Group Seminar

12th May 1998

Andrew Broad

broada@cs.man.ac.uk

http://www.cs.man.ac.uk/~broada/

MPhil Project

The Application of Case-Based Reasoning to Transforming Constraints for Schema-to-Schema Mapping

Transforming Constraints for Schema-to-Schema Mapping

- We want to translate data records from one data repository to another.

- But the data are in two different formats!

- The format of data in each repository is specified by an EXPRESS model.

- The two models may differ *structurally*, which requires records from the source repository to be transformed into the format of the target repository.

- The two models may have different *constraints* on them, which affects what records can be transferred.

Example of the Constraints Problem

- Consider the following two EXPRESS schemata:

```
SCHEMA person_source;
   ENTITY person;
      name : STRING;
      age : INTEGER;
   END_ENTITY;
END_SCHEMA;
```

```
SCHEMA person_target;
   ENTITY person;
      name : STRING;
      age : INTEGER;
   UNIQUE
      name;
   WHERE
      {0 <= age < 130};
   END_ENTITY;
END_SCHEMA;
```

- Before we map a record from source to target repository, we had better check that it isn't going to violate the constraints on the target model!

  e.g.

```
#1 = PERSON('John Blaylock', 30);
#2 = PERSON('John Blaylock', 330);
```

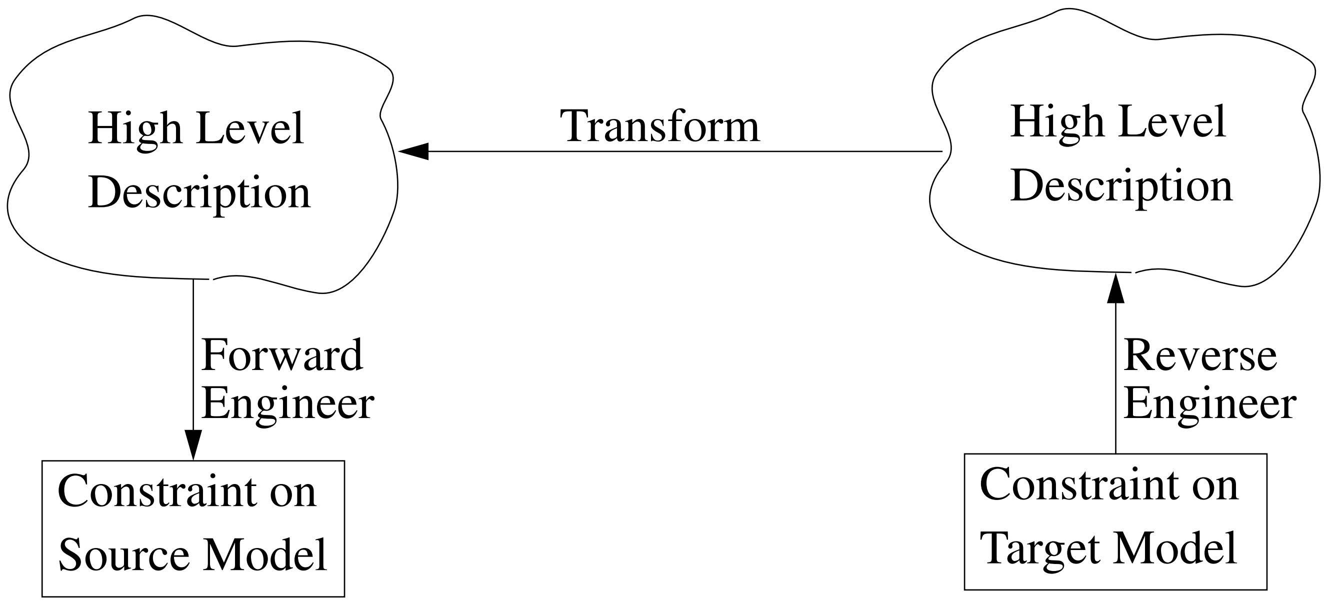
Project Aims

- It's an *automatic programming* problem: we want a system that will take two EXPRESS models as input and generate a program to map records from one conformant repository to another.

- The novel contribution will be for the system to plant code to check the target model's constraints on source records *before* they are mapped.

- This involves transforming the target model's constraints onto the source model so we can check that the records in the source repository respect those constraints.

- We take a 'reengineering' approach rather than hacking the EXPRESS code directly!



- It may be necessary to transform constraints due to differences in the two models (rather than just copying the code across unedited).

e.g.

```
SCHEMA temperature_in_fahrenheit;
  TYPE fahrenheit = REAL;
  END_TYPE;

  ENTITY water;
    temperature : fahrenheit;
  END_ENTITY;
END_SCHEMA;


SCHEMA temperature_in_celsius;
  TYPE celsius = REAL;
  END_TYPE;

  ENTITY water;
    temperature : celsius;
  WHERE
    {0 < temperature < 100};
  END_ENTITY;
END_SCHEMA;
```

- Here is the transformed source model (with the constraint from the target model in bold italics):

```
SCHEMA temperature_in_fahrenheit;
  TYPE fahrenheit = REAL;
  END_TYPE;

  ENTITY water;
    temperature : fahrenheit;
  WHERE
    {32 < temperature < 212};
  END_ENTITY;
END_SCHEMA;
```

- Transforming this constraint requires knowing $F = 32 + {}^{9}/_{5}\,C$.

- It is not necessary to transfer a constraint from the target model to the source model if it already holds on the source model!

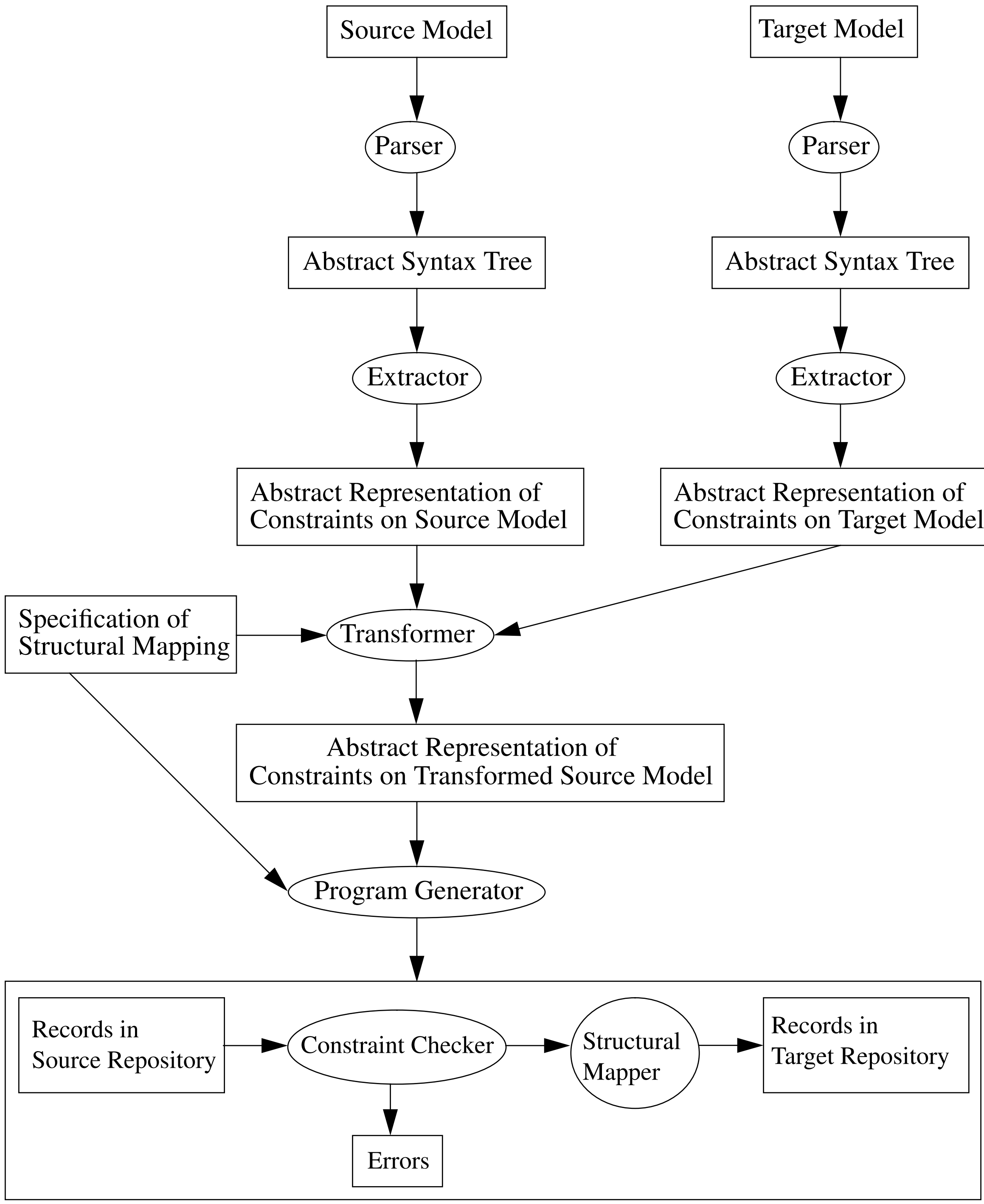
e.g. Consider mapping *from* the following model to the `person_target` model on Slide 3:

```
SCHEMA teenagers;
   ENTITY person;
      name : STRING;
      age : INTEGER;
   WHERE
     {13 <= age <= 19};
   END_ENTITY;
END_SCHEMA;
```

- It is not necessary to transfer the WHERE clause `{0 <= age < 130}` from `person_target` to the `teenagers` model, because it is already implied by `{13 <= age <= 19}` (whereas if it was the other way round, we would).

- To avoid transferring unnecessary constraints requires inferring semantic equivalences between models.

Architecture of the Constraint Transformation System



Higher Level Constraints

- Judging the equivalence of constraints on different models requires understanding those constraints at a more abstract level.

- The approach taken is to extract *higher level constraints* off of the models themselves, and use those to infer the extent to which the constraints on the models are semantically equivalent.

- A higher level constraint frame represents deeper knowledge of a constraint on a model.

  e.g.

```
WHERE ((age >= 0) AND (age < 130))
```

```
inst = NUMERIC_RANGE_CONSTRAINT
thing_constrained = age
lower_bound =   inst = BOUND
                limit = 0
                inclusive = true

upper_bound =   inst = BOUND
                limit = 130
                inclusive = false
```

Example of using Higher Level Constraints to infer equivalence

```
SCHEMA gender_by_inheritance;
  ENTITY person ABSTRACT SUPERTYPE OF
              (ONEOF (man,woman));
    name : STRING;
    age : INTEGER;
  END_ENTITY;

  ENTITY man SUBTYPE OF (person);
  END_ENTITY;

  ENTITY woman SUBTYPE OF (person);
  END_ENTITY;
END_SCHEMA;
```

```
inst = EXACTLY_ONE
choice_set = {man,woman}
```

```
SCHEMA gender_by_enumeration;
  TYPE gender = ENUMERATION OF (male,female);
  END_TYPE;

  ENTITY person;
    name : STRING;
    age : INTEGER;
    sex : gender;
  END_ENTITY;
END_SCHEMA;
```

```
inst = EXACTLY_ONE
choice_set = {male,female}
```

Case-Based Reasoning

- Case-Based Reasoning is the enterprise of solving new problems by analogy with old ones rather than from first principles.

- Problems and their solutions are stored in memory as *cases*.

- A Case-Based Reasoner solves a new problem by *retrieving* a similar case and *adapting* its solution to fit the new situation (it may then store the problem it's just solved as a new case).

- CBR is brilliant because:

- it is faster to solve complex problems by starting from old solutions rather than from scratch!

- cases can suggest solutions to problems that could not be solved in any other way!

- a CBR system can learn from experience by acquiring new cases, thereby improving its efficiency and its capabilities over time, and even avoiding previous mistakes!

- CBR systems are easier to build and maintain than the equivalent non-CBR systems, which is good for prototyping!

Case-Based Reasoning in Constraint Transformation

- There are several tasks for CBR in the constraint transformation domain:

- **Extraction**:
  - Cases can suggest what HLCs to extract;
  - Cases can be used to verify hypotheses as to what HLCs to look for.

- **Transformation**:
  - Cases can suggest how to realise HLCs that are needed on the source model as EXPRESS code.

- **Program Generation**:
  - Cases could suggest how to write the code which checks the constraints on the transformed source model.
  - Cases could suggest how to write the code which implements the structural mapping.

An example of Extraction using Case-Based Reasoning

```
SCHEMA manic_miner_1;
   ENTITY item;
      position_x : INTEGER(2);
      position_y : INTEGER(2);
      ink_colour : INTEGER;
   WHERE
      {3 <= ink_colour <= 6};
   END_ENTITY;
END_SCHEMA;
```

```
SCHEMA manic_miner_2;
   TYPE colour = ENUMERATION OF
                 (magenta,green,cyan,yellow);
   END_TYPE;

   ENTITY item;
      position_x : INTEGER(2);
      position_y : INTEGER(2);
      ink_colour : colour;
   END_ENTITY;
END_SCHEMA;
```

- This example shows how extraction should be driven by the needs of transformation.

• First, a ***NUMERIC_RANGE_CONSTRAINT*** is extracted from `manic_miner_1`, this being a very *explicit* constraint...

```
inst = NUMERIC_RANGE_CONSTRAINT
thing_constrained = ink_colour
lower_bound = | inst = BOUND
              | limit = 3
              | inclusive = true
upper_bound = | inst = BOUND
              | limit = 6
              | inclusive = true
```

• ... and an ***EXACTLY_ONE*** from `manic_miner_2`:

```
inst = EXACTLY_ONE
choice_set = {magenta,green,cyan,yellow}
```

• Then a more implicit ***EXACTLY_ONE*** is extracted from `manic_miner_1`:

```
inst = EXACTLY_ONE
choice_set = {3,4,5,6}
```