UKCBR4 Presentation

15th September 1999

Applying Case-Based Reasoning to Code Understanding and Generation

Andrew Broad

and

Nick Filer

Department of Computer Science
University of Manchester
Manchester, England

`{broada/nfiler}@cs.man.ac.uk`

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

THE UNIVERSITY of MANCHESTER

Overview

The paper addresses several themes:

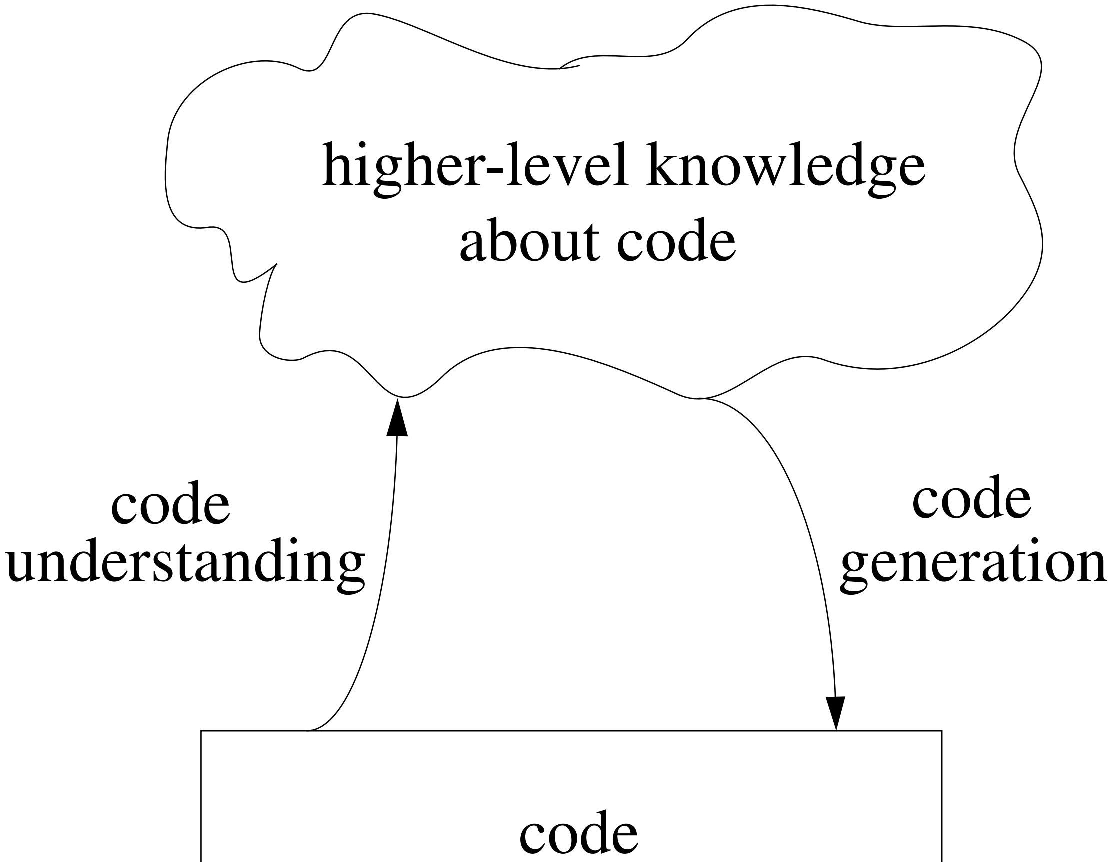
- Why should CBR be applied to code understanding and code generation?

- How has CBR already been applied to code understanding and code generation?
  - *explicit* CBR
  - *covert* CBR

- Future directions...

Code Understanding and Code Generation

- What do we mean by *code*?
  - program code (e.g. Java)
  - formal specifications (e.g. VDM, Z)
  - information models (e.g. EXPRESS, UML)
  - *any* code written in a formal language!



THE UNIVERSITY of MANCHESTER

Examples of Code (1)

- Some program code (written in Java):

```
int index= 0;

while ((index < length)
     && (array[index] != goal))
{
   index= index + 1;
}

if (index < length) {
   System.out.println(goal
   + " found at array[" + index + "]");
} else {
   System.out.println(goal + " not found");
}
```

Examples of Code (2)

- Model-oriented formal specification:

```
push (s : stack, x : int)
   pre true
   post s' = x::s

pop (s : stack)
   pre length(s) > 0
   post s' = tail(s)
```

- Algebraic formal specification:

```
type Stack
operations
   Create  : ()            -> Stack
   Push    : (Stack,Int)   -> Stack
   Pop     : (Stack)       -> Stack
axioms
   Pop(Create) = Create
   Pop(Push(s,i)) = s
end
```

Examples of Code (3)

- An information model (written in EXPRESS):

```
SCHEMA simple_people_model;
   ENTITY person;
      name : STRING;
      age : INTEGER;
   WHERE
      age_must_be_sensible:
         (age >= 0) AND (age < 130);
   END_ENTITY;
END_SCHEMA;
```

Why is CBR Applicable to Code *Generation*?

- The ultimate goal of *automatic programming* has three features:
  - takes as input a brief, informal specification
  - fully automated
  - general-purpose

- Why could CBR help us to move towards this goal?
  - Automatic programming is theory-weak, but there are plenty of cases!
  - Cases could suggest which transformations to apply, rather than the user.
  - Analogical reasoning could help to achieve more general-purpose automatic programming.

Why is CBR Applicable to Code *Understanding*? (1)

- If CBR can be applied to code generation, why not apply it to the inverse task?

- *Bidirectional* CBR: perhaps the code generation cases could also be used for code understanding.

- Either way, a case consists of a ‘specification’ and an ‘implementation’.
  Match one side and extract the other.

Why is CBR Applicable to Code *Understanding*? (2)

Code understanding has several difficulties:

- **Code understanding is NP-complete.**

  CBR could alleviate the combinatorial explosion by matching and extracting larger chunks at once.

- **Plan libraries can never be complete.**

  CBR can handle exceptions to the norm by acquiring new cases.

- **Some information cannot be recovered from the source code.**

  The reverse-engineering process would have to take other artifacts into account.

THE UNIVERSITY of MANCHESTER

Explicit CBR Approaches

- Openly acknowledged as being CBR approaches.

- Much of the explicit CBR concerns *software reuse* – retrieval and adaptation in reuse correspond to retrieval and adaptation in CBR.

- Déjà Vu applies CBR to automatic programming in a restricted domain.
  – *hierarchical CBR*
  – *adaptation-guided retrieval*

- *Derivational replay* approach to algorithm design (Mostow & Fisher)

- Application of CBR to software verification, using *internal analogy* (Melis & Schairer)

Covert CBR Approaches (1)

- In the spirit of CBR, but not claimed to be CBR.

- An important class of code understanding techniques is based on *clichés* (commonly-used computational structures – data or algorithmic).

- The *cliché library* contains *overlays*.

  Each overlay associates an *implementation cliché* with a *specification cliché* (to extract when the implementation cliché is recognised in the code).

- Overlays are analogous to cases, but:
  - – are small chunks of knowledge;
  - – no partial matching or adaptation.

  So it's less flexible and efficient than true CBR.

- Clichés can be used for code generation too, as in the Programmer's Apprentice.

Covert CBR Approaches (2)

- Unsurprisingly, there is a lot of covert CBR in the field of software reuse, as well as explicit CBR.

- *Design patterns* are based on the same underlying philosophy as CBR – solutions to commonly-recurring software problems. But they are intended for human – rather than automatic – programmers.

- EVA uses derivational replay to modify a program when its specification changes. It does adaptation without retrieval – *evolution* as opposed to *reuse*.

- *Analogical reasoning* approach to specification reuse (Maiden & Sutcliffe).

  Analogical reasoning is like CBR, but source case and target case can be from different domains – *analogical mapping* is needed to see the analogy.

Future Directions (1)

- Our current research focuses on understanding constraints in EXPRESS information models – a particular form of code understanding.

- We take a CBR approach to extracting higher-level knowledge from (EXPRESS) code.

- The constraint-understanding system is being extended to languages other than EXPRESS (c.f. Slide 2) – Java in particular.

- In the future, we aim to achieve *comparative code understanding* – being able to compare the semantics of two pieces of code.

- Another research area is the *discriminate* extraction of knowledge from code – only extracting that knowledge which is useful to the task at hand.

  This entails *meta-level reasoning*.

Future Directions (2)

- *Case-based automatic programming* is the ultimate dream!

- The code generation side would be like a CBR software reuse system, but more fully automated than those of today.
  Analogical reasoning could help to make it more general-purpose, and CBR could also help to elaborate the ‘brief, informal requirements’.

- The code understanding side could be integrated into the same system, by taking a bidirectional CBR approach.