Pattern Matching and Recognition

689 - Special Topics on Advanced Compiler Technologies

Alin Jula
March, 19th 2003

Motivation - Why do we care?

- Abstract concepts are simpler to deal with and closer to the human language
- Enables implementation substitution
  - e.g. recurrences can be replaced
- Eases maintenance process
  - e.g. Polaris - 600,000 lines of code

General Issues

- Pattern matching and recognition = Semantic compiler

<table>
<thead>
<tr>
<th>Highest Level</th>
<th>Human languages: English, Romanian, Spanish, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher - Level codes</td>
<td>STAPL, CASE tools</td>
</tr>
<tr>
<td>Low - Level codes</td>
<td>Human C, assembly</td>
</tr>
</tbody>
</table>

Example

- “Given a set of uranium isotopes with random energies and velocities, calculate their coordinates (energy, velocities, space coordinates, etc) after 3 seconds of interaction”
- Equivalent with thousands or maybe hundred of thousands of lines of code

Pattern Matching and Recognition

- Program Understanding
  - (Low to High Level)
- Code Generation and Replacement
  - (High to Low Level)
Quilici - Overview

- Translates C into C++
  - Input: C code in Abstract Syntax Tree form
  - Output: Programming plans (then C++)
- Analysis
  - Bottom-Up on the Code
  - Top-Down on the Programming Plans

Quilici - Example

Equivalent C++ code

Quilici - Plan Library

- A Library of patterns (programming plans)
- Extends an existing Library (from Andersen Consulting for understanding Cobol).

Quilici - Programming Plan

- Programming Plan
  - definition: list of attributes
  - recognition rule(s)
    - components of the plan
    - constraints on the components

Quilici - Library Organization

- Plan indices
  - Connects plans hierarchically
  - Objective: narrows the search
- Specialization Constraints
  - Stores only the Δ constraints
  - Objective: eliminates commonalities
- Implied Plans
  - Objects recognized form a particular plan
  - Objective: avoids matching them with the code, their existence is implied from the recognized plans
Quilici - The Algorithm

1. Extract the first component C from L.
2. Specialize C as far as possible.
3. Do each of its specializations in C.
4. If none succeeds, retry the specialization process.
5. Repeat until all possible specializations are considered.

Quilici Example

BH - Overview

- Parallelizes FORTRAN code using pattern matching
  - Input: (FORTRAN code - abstract syntax tree)
  - Output: (parallel FORTRAN code)
- Analysis
  - Patterns are found in the code
  - User replaces these patterns with an already parallelized implementation

BH - Pattern Language

- Describes
  - programming concepts
  - domain concepts
- The patterns are written in the pattern language before pattern recognition process
BH - Pattern Language Syntax

- Pattern rule: (SYNTAX) pattern-name (edge/pattern-body) (constraint) (pattern-body) = (PARTIAL (pattern-body)) = (ORIF (pattern-body)) = (텀마스 (constraint) (pattern-body))
- Constraint = (LABEL) (가시) = (LABEL) (가시)

BH - Patterns Examples

<table>
<thead>
<tr>
<th>PART</th>
<th>Rule</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>(HEXPY N, N, N)</td>
<td>x \oplus y \oplus z</td>
</tr>
<tr>
<td>1.2</td>
<td>(HEXPY N, N, N, N)</td>
<td>x \oplus y \oplus z \oplus w</td>
</tr>
<tr>
<td>1.3</td>
<td>(HEXPY N, N, N, N, N)</td>
<td>x \oplus y \oplus z \oplus w \oplus v</td>
</tr>
<tr>
<td>1.4</td>
<td>(HEXPY N, N, N, N, N, N)</td>
<td>x \oplus y \oplus z \oplus w \oplus v \oplus u</td>
</tr>
</tbody>
</table>

BH - Pattern Classification

- Hierarchical approach
- Classification of patterns
  - base-concept (atomic) - e.g. swap
  - intermediate-concept - e.g. pivoting
  - domain-concept - e.g. Gaussian elimination
- Dependencies between the patterns are stored in the library

BH - Pattern Recognition

- The user is asked to specify the domain concepts as narrow as possible
- The AST is then compared against the Pattern Library

BH - Domain Patterns
**BH - Algorithm**

- Prune the space of patterns to search
- Base concepts are identified
- Intermediate patterns are built up from the base ones
- Domain concepts are built up from intermediate ones
- One Pass Search!
  - Fast
  - Not effective

**BH - Algorithm (cont.)**

![Diagram](before_search_after_intermediate_concept_pattern_search_after_domain_concept_search)

**K - Overview**

- Parallelizes FORTRAN 77 and C code using pattern matching
  - Input: (FORTRAN 77 and C code - abstract syntax tree)
  - Output (parallel C code)
- Analysis
  - Graph based
  - Hierarchical approach

**K-Library**

- Collected around 150 patterns for scientific numeric codes
- Hierarchical approach
  - e.g. matrix-multiplication = loop over a dot product
- Patterns characterized based on the depth of the loop nest
  - e.g. scalar = 0. Dot product = 1, matrix multiplication = 2, etc
K - Library

K- Library Organization

- Organized as a Pattern Hierarchy Graph (PHG)
- \( G(V,E) \) - \( V \) is a set of patterns, \( E \) set of dependencies between patterns
- \( m \) dependent of the trigger patterns \( m_i \), there is an edge between \((m,m)\)
- Pattern recognition becomes a path finding program in PHG

K - Library Organization

For \( (m_1) \):
\[
\text{for } (m_1) \text{; } \text{for } (m_2) \text{; } (m_1, m_2)
\]
\[
\begin{align*}
\text{if } & (m_1) \text{; } (m_2) \text{; } (m_1, m_2) \\
\text{if } & (m_1, m_2) \text{; } (m_1, m_2) \\
\text{if } & (m_1) \text{; } (m_1, m_2) \\
\text{if } & (m_2) \text{; } (m_1, m_2) \\
\text{if } & (m_1, m_2) \text{; } (m_1, m_2)
\end{align*}
\]

K - Algorithm

K - Paramat

- Paramat = PARallelize Automatically by pattern MATCHing
- Tool for pattern recognition and algorithm substitution
- Automatic Parallelization Tool
**K-Paramat**
- Algorithm replacement (recurrences)
- Data distribution
- Run time prediction

**W & Y - Overview**
- Program Understanding as Constraint Satisfaction Problem
  - a set of variables \( X_i \)
  - a set of values for each \( X_i \), \( \text{Dom}(X_i) \)
  - set of constraints - permissible subsets to variable
- Constraints implemented in Prolog
- Searching algorithm - backtracking

**W & Y - Algorithm**
Generic CSP Solve:
- \( V \): variables in a CSP, \( \text{Dom}(X) \): the domain values of \( Y \).
1. [Initialization]: for each variable \( X_i \in V \), find the set of domain values for \( X_i \).
2. [Initial Constraint Propagation]: before \( \text{Dom}(X) \) by constraint propagation.
3. [Select Value]: \( \text{Dom}(X) \) from \( V \).
4. [Value Selection]: Select and remove a value \( Y \) from \( \text{Dom}(X) \).
5. [Backtrack Point Selection]: Backtrack if any \( \text{Dom}(X) \) is \( V \) becomes empty.
6. [Solution Evaluation]: If \( V \) is empty, exit with solution (if solution, continue)
   - go to Step 1.
PAP - Overview
- Parallelizable Algorithmic Patterns
- Graphical tool - permits visualization of the recognized concepts, together with the implementation within the program
- Integrated into Vienna Fortran Compilation System

PAP - Recognizer
- Performs a hierarchical parsing driven by concept recognition rules (implemented in Prolog)
- Concept
  - its compositional hierarchy (set of composing subcomponents)
  - Relationships and constraints among composing components

PAP - APR
- PAP builds an Abstract Program Representation (APR) from a Basic Program Representation (BPR) (initial code)
- Base level of representation - Program Dependence Graph (PDG)
  - nodes - statements
  - edges
    - control flow (labeled with T/F)
    - data dependence (labeled with variable identifier)
- The APR is a hierarchical PDG

PAP - Traits
- PAP deals with
  - Program variation - various implementations for the same algorithm
  - Delocalization - implementation of a concept throughout the code
  - Overlapping implementations - implementations of two or more concepts are merged
PAP - Output example

BFV - Overview
- Code Generator
- Graphical Tool
- Input
  - Design Patterns
- Output
  - C++ code.

BFV
**M & Y - Algorithmic equivalence**

- Let P and P' be two subprograms. Define P and P' to be algorithmically equivalent iff one can be obtained from the other one through following transformations:
  - Rename variables (bijective function-mapping)
  - Reorder the statements without violating the semantics
  - Permute the operands of any commutative operator

**References**

- [Quilici] "An Opportunistic, Memory-Based Approach to Recognizing Programms Plans", Alex Quilici
- [K] "Pattern-Driven Automatic Parallelization", Christoph Kessler
- [W&Y] "Program Understanding as Constraint Satisfaction Representation and Reasoning Techniques", S. Woods, Q. Yang
- [M&Y] "Automatic Algorithm Recognition and Replacement", Robert Metzger and Zhaofang Wen