NUS

National University of Singapore

School of Computing

CS3203 Software Engineering Project

AY22/23 Semester 1

Project Report – System Overview

Team 11

Team Members	Student No.	Email
AMADEUS ARISTO WINARTO	A0221733N	e0559265@u.nus.edu
BERNARDUS KRISHNA	A0196717N	e0389203@u.nus.edu
CHAN CHOON YONG	A0222743L	e0560275@u.nus.edu
CHEN HSIAO TING	A0222182R	e0559714@u.nus.edu
NG JUN WEI, TIMOTHY	A0217635A	e0543671@u.nus.edu
SIMON JULIAN LAUW	A0196678A	e0389164@u.nus.edu

Consultation Hours: Tuesday, 1 PM - 2 PM

Tutor: Sumanth

Table of Content

Appendix

1. Extension Proposal

1.1 Definition of the extension

We propose an extension to SPA by introducing a new SIMPLE construct called `match-case` statement.

1.1.1 Extension to SIMPLE Grammar Rules

Concrete Syntax Grammar (CSG)

```
stmt: read | print | call | while | if | assign | match_case

match_case: 'match' '(' var_name ')' '{' caseLst '}'
caseLst: case+
case: 'case' '(' expr ')' '{' stmtLst '}' | 'case' '(' '_' ')' '{' stmtLst '}'
```

Abstract Syntax Grammar (ASG)

```
stmt: read | print | call | while | if | assign | match_case

match_case: var_name caseLst
caseLst: case+
case: expr stmtLst | stmtLst
```

Attributes and Value Types:

```
match.varName: NAME
match.stmt#, case.stmt#: INTEGER
```

Other Rules:

1. **Statement Number**

Each `match` and `case` line receives an index and statement number.

2. **No Fall-through Rule**

Formally, given any 2 cases `c1` and `c2`, If

- `Parent(c1, s1)` // any statement in case 1
- `Parent(c2, s2)` // any statement in case 2
- `Parent(m, c1)` and `Parent(m, c2)` // same match block

then, `Next(s1, s2)` does not hold

Note: multiple `case` can evaluate to the same expression, and only the matching `case` will be evaluated.

2.1.2 Extension to PQL Grammar Rules

Lexical Tokens

```
design-entity : 'stmt' | 'read' | 'print' | 'while' | 'if' | 'assign' | 'variable'
| 'constant' | 'procedure' | 'match' | 'case'
```

Grammar Rules

```
pattern : assign | while | if | match | case
match: syn-match '(' entRef ')'
case: syn-case '(' expression-spec ',' '_' ')'
syn-match : IDENT
```

```
syn-case : IDENT
```

Other Rules:

1. **Match Pattern:** The first argument in the match pattern can only be a variable synonym, wildcard, or variable name in quotes.

Refer to the appendix for examples of a SIMPLE program with match-case construct, the CFG, and PQL queries with new pattern clauses.

1.2. Changes to Existing System

1.2.1. Existing Design Abstractions

Design Abstraction	Changes
Parent/Parent*	match and case statements are container statements
Uses/Modifies	Container statement s includes match and case statements

1.2.2. SP

Tokeniser:

- Refactorisation of QPS' WildCardTokenizer class by relocating this class to Common/Tokenizer, accessible by SP. This refactor supports the '_' symbol within a case statement.

Parser:

- Add MATCH and CASE statement keywords to the StatementKeywordConst header file.
- Add MATCH and CASE if-else conditional statements to parseStmtPrime to invoke parseMatchStmt and parseCaseStmt functions respectively.
- Addition of parseMatchStmt method within the StatementParser class.
- Addition of parseCaseStmt method within the StatementParser class.

Traversers: No changes required. Logic for match_case statements is located within MatchNode and CaseNode.

AST:

- Addition of new class WildcardNode, to represent '_', which extends from ExprNode.
- Addition of new classes MatchNode and CaseNode, which extends from StatementMixin, ModifiesMixin, ParentMixin and UsesMixin.

CFG:

- Modification of OutNeighbours data type to accept >= 2 out-neighbour CFG nodes, since the use of match-case should support an arbitrary number of case statement lists.

1.2.3. PKB

- Add Match and Case in existing StatementType class
- Add Match and Case statements in StatementStore matching the additions to StatementType
- Addition of MatchVarStore implementing ManyToMany<Variable,StatementNumber>
- Addition of CaseStore with similar pattern matching capabilities (without the notion of lhs and rhs) as AssignmentStore
- DirectParentStore/ParentStarStore store the parent relationship of the new match-case pattern
- StatementUsesStore should store uses of variables of the new match-case pattern
- StatementModifiesStore should store modification of variables of the new match-case pattern
- Addition of new APIs in ReadFacade and WriteFacade

1.2.4. QPS

- Addition of the strategies PatternMatchStrategy and PatternCaseStrategy
- Addition of the syntactic pattern analysers PatternMatchAnalyser and PatternCaseAnalyser
- Addition of the syntactic patterns PatternMatch and PatternCase
- Addition of the syntactic pattern evaluators PatternMatchEvaluator and PatternCaseEvaluator

1.3. Implementation Details

1.3.1. SP

Parser:

- Declare MATCH=*"match"*; and CASE=*"case"*; keywords with 'static constexpr string'.
- In parseStmtPrime, if the previous token is MATCH, parseStmtPrime will call parseMatchStmt.Else if the previous token is CASE, parseStmtPrime will call parseCaseStmt.
- Algorithm for parseMatchStmt method.
  a. Parse Variable Name after MATCH, and create new VarNode.
  b. While Next Token is CASE, call parseCaseStmt and store each CaseNode into a Vector.
  c. Return MatchNode which stores VarNode from (a), and the CaseNode vector from (b).
- Algorithm for parseCaseStmt method.
  a. Parse the expression or wildcard after CASE, and create a new ExprNode.
  b. Parse the nested statement list using the existing StatementListParser.parse method, which returns StatementListNode.
  c. Returns CaseNode which stores the ExprNode from (a), and the StatementListNode from (b).

Abstract Syntax Tree (AST)

- WildcardNode is similar to NullNode, but the node name is "WildcardNode".
- Algorithm for all populate_pkb_* methods for MatchNode class
  a. Traverse the VarNode.
  b. Traverse each of the CaseNodes.
- Algorithm for all populate_pkb_* methods for CaseNode class
  a. Traverse the ExprNode.
  b. Traverse each StatementNode in the StatementListNode .

Control Flow Graph (CFG):

- OutNeighbours changes data type from Pair of Strings to a Vector of Strings.

1.3.2. PKB

- Addition of APIs in ReadFacade depending on QPS needs, see appendix for details
- Addition of APIs in WriteFacade, see appendix for details
- Storing of parent relationship in DirectParentStore does not involve changes to existing PKB, because SP would populate as per necessary
- Computation of ParentStarStore does not change. The parent relation graph remains a DAG, so the Parent* relations would still be populated correctly
- Storing of uses of variables in StatementUsesStore and modification of variables in StatementModifiesStore do not change existing PKB, because SP would populate as necessary
- Create new MatchVarStore implementing ManyToMany<Variable, StatementNumber>
- Create new CaseStore with similar pattern matching capabilities (without the notion of lhs and rhs) as AssignmentStore. I.e., given stored_case and queried_case:
  - Exact match → stored_case == queried_case
  - Partial match → stored_case.find(queried_case) != std::string::npos
  - Wildcard → no filter

1.3.3. QPS

- For the syntax validation process, we implement new `PatternMatchStrategy` and `PatternCaseStrategy` which defines the new PQL grammar rules
- For semantic validation, we implement new `PatternMatchAnalyser` and `PatternCaseAnalyser` for the new untyped pattern clauses. They would return a `PatternMatch` and `PatternCase` respectively.
- For the evaluation process, we would add new `PatternMatchEvaluator` and `PatternCaseEvaluator` for `PatternMatch` and `PatternCase` syntactic patterns respectively.
- To support the new PQL syntax, we can update TypeList `DefaultSupportedPatternStrategies` to include the new strategies, or let users decide which strategies they would like to use/compile.

1.4. Possible Challenges to Implementation and Testing, and Mitigation Plans

1.4.1. SP

- The new `WildcardNode` must be thoroughly tested. Mitigation: We adopt `WildcardTokenizer` test cases from QPS.
- `match_case` statements must be stress tested (e.g. >50 `match_case` statements). Mitigation: We use AI to generate such large test cases.
- CFG with `match` and `case` statements must be carefully verified, especially the dummy nodes. Mitigation: overload the `<<` operator of `MatchNode` and `CaseNode` to visualise the CFG during testing.

Control Flow Graph (CFG):

- `OutNeighbours` data type must be ordered so that CFG traversal logically corresponds to the SIMPLE source program execution. The first `case` conditional expression will be executed, followed by subsequent `case` expressions in sequence.

1.4.2. PKB

- Additional unit, integration and system testing are needed to ensure the new match-case pattern works well on top of what is already implemented.

1.4.3. QPS

- Additional unit, integration and system testing are needed to ensure the new match-case pattern works well on top of what is already implemented.

1.5. Benefits to SPA

- Allows Multiple Conditions / Greater Flexibility: match-case structure offers multiple case conditions, unlike the current if-else, which only offers two.
- Enhanced Readability: match-case structure offers a cleaner way to represent complex conditional logic compared to deeply nested if-else blocks.
- Improved Performance: if we can optimize match-case structure more efficiently than if-else blocks, it could lead to performance and efficiency improvements.

2. Plan for Milestone 3

Gantt chart key

🔴 QPS
🟠 PKB
🔵 SP
🟡 Common

	W10 Tue	W10 Fri	W11 Tue	W11 Fri	W12 Tue	W12 Fri	W13 Tue	W13 Fri
Affects Implementation	Affects							
Not Implementation		Not						
QPS Optimization				UFDS, Hash Join, Heuristic, etc.				
Next* Optimization	Tarjan's Algo, etc.							
PKB Optimization				Caching, etc.				
System Testing + Bug Fixing				Testing and Bug Fixing				
Integrate Tracing Tool	Tracy							
Integrate Google Benchmarking	Google Benchmarking							
Milestone 3 Submission + Presentation							Freedom	

3. Testing Progress

Type of Test	Quantity
Unit-test coverage	2640 assertions in 110 test cases
Integration-test coverage	280 assertions in 18 test cases
System-test coverage	666 test cases in 32 test files

Appendix A: Example of SIMPLE Program with Match-Case Construct

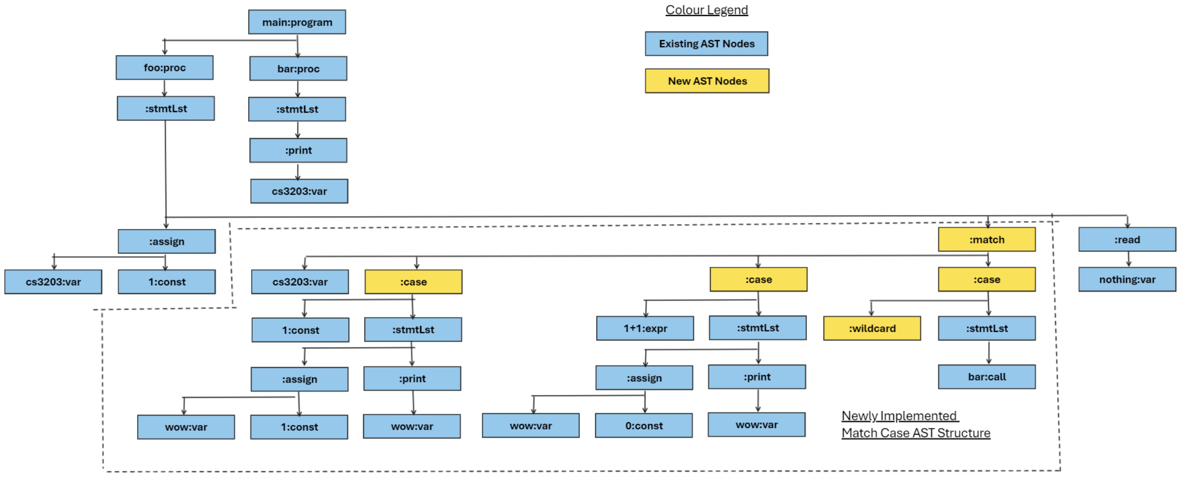
```
procedure foo {
  cs3203 = 1;         // 1
  match (cs3203) {    // 2
    case (1) {        // 3`
      wow = 1;        // 4
      print wow;      // 5
    }
    case (1 + 1) {    // 6
      wow = 0;        // 7
      print wow;      // 8
    }
    case (_) {        // 9
      call bar;       // 10
    }
  }
  read nothing;       // 11
}

procedure bar {
  print cs3203;       // 12
}
```

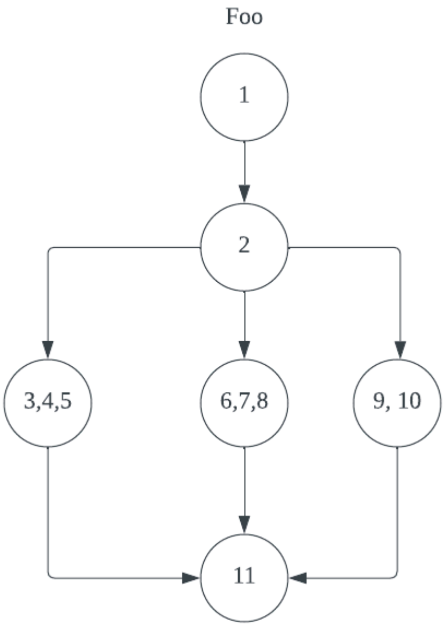
Appendix B: Example of PQL Query with Pattern Match and Case

```
   1) match m; case c; variable v; Select BOOLEAN pattern m(v) pattern c('1 + 1',
      _)
   2) match m; case c; Select m pattern m('cs3203') pattern c(_, _)
   3) match m; case c; Select <m, c>
   4) match m; case c; Select <m, c> pattern m(_) pattern c(_, _) such that
      Parent(m, c)
```

Appendix C: AST Diagram of Appendix A



Appendix D: CFG of Foo from Appendix A



Appendix E: New APIs for PKB ReadFacade

Match pattern-related Read Operations

std::unordered_set<std::string> get_match_stmts_with_var() const;
std::unordered_set<std::string> get_match_stmts_with_var(const std::string& variable) const;
std::unordered_set<std::string> get_vars_in_any_match() const;
std::unordered_set<std::string> get_vars_in_match(const std::string& if_stmt) const;

Case pattern-related Read Operations
// wildcard

```
std::unordered_set<std::string> get_all_case();

// exact match
std::unordered_set<std::string> get_all_case_exact(const std::string& expr);

// partial match
std::unordered_set<std::string> get_all_case_partial(const std::string& expr);
```

Appendix F: New APIs for PKB WriteFacade

Match pattern-related Write Operations
void add_match_var(const std::string& statement_number, const std::string& variable);
Case pattern-related Write Operations
void add_case_expr(const std::string& statement_number, const std::string& expr);