Matching Logic

Andrei Ștefănescu Chucky Ellison Grigore Roșu

University of Illinois at Urbana-Champaign

September 17, 2010

Matching Logic

Matching logic is a framework for defining axiomatic semantics for programing languages

- Inspired from operational semantics
- Program configurations play an important role
- Specifications: special FOL= formulae, patterns
- Configurations match patterns
- Patterns can be used to:
  - Give an axiomatic semantics to a language, so that we can reason about programs
  - Define and reason about patterns of interest in program configurations

Program Configurations (no heap)

- Simple configuration using a computation and an environment

$$\langle \langle \ldots \rangle_k \ \langle \ldots \rangle_{env} \rangle_{config}$$

- Example

$\langle \langle x = 1; y = 2; \rangle_k$
$\langle x \mapsto [3 : int],\ y \mapsto [3 : int],\ z \mapsto [5 : int] \rangle_{env} \rangle_{config}$

Program Configurations (add heap)

- Add a heap to the configuration structure

$$\langle \langle \ldots \rangle_k \ \langle \ldots \rangle_{env} \ \langle \ldots \rangle_{mem} \rangle_{config}$$

- Example

$\langle \langle *x = 5; y = *x; \rangle_k \ \langle x \mapsto [2 : int*], \ y \mapsto [3 : int] \rangle_{env}$
$\langle 1 \mapsto 7, \ 2 \mapsto 3 \rangle_{mem} \rangle_{config}$

Pattern Matching

- Configurations match ($\models$) patterns iff they match the structure and satisfy the constraints
- Example

$$\langle\langle *x = 5; y = *x; \rangle_k \langle x \mapsto [2 : int*],\ y \mapsto [3 : int]\rangle_{env}$$
$$\langle 1 \mapsto 7,\ 2 \mapsto 3\rangle_{mem}\rangle_{config}$$
$$\models$$
$$\langle\langle *x = 5; y = *x; \rangle_k \langle x \mapsto [?x : int*],\ y \mapsto [?y : int],\ ?\rho\rangle_{env}$$
$$\langle ?x \mapsto ?a,\ ?\sigma\rangle_{mem}\ \langle ?a \geq 0\rangle_{form}\rangle_{config}$$

Symbolic Execution

- Assumes the pre-condition, symbolically execute the program, assert the post-condition
- The symbolic execution semantics are identical to the dynamic semantics for most program constructs
- The semantics are different for
  - If
  - While
  - Call/Return

MatchC

- Implemented on top of Maude rewrite engine
- Provides simplification rules for program configuration
- Reason simultaneously about heap patterns and value properties
- Uses the SMT-LIB standard to query SMT solvers for integer reasoning

List Reverse

- C program that reverses a list in place

```
struct list_node* reverse(struct list_node *x) {
  struct list_node *p;
  p = 0 ;
  while(x != 0) {
    struct list_node *y;
    y = x->next;
    x->next = p;
    p = x;
    x = y;
  }
  return p;
}
```

List Reverse(2)

- List data structure

```
struct list_node {
  int value;
  struct list_node *next;
};
```

- Reverse pre/post conditions

struct list_node* reverse(**struct** list_node *x)

pre:
$\langle\langle x \mapsto [?x : \text{struct list\_node}*] \rangle_{env} \langle list(?x, \alpha) \rangle_{mem} \langle true \rangle_{form} \rangle_{config}$
post:
$\langle\langle .Map \rangle_{env} \langle list(?p, ?\alpha) \rangle_{mem} \langle returns\ ?p \wedge ?\alpha = rev(\alpha) \rangle_{form} \rangle_{config}$

List Reverse(3)

- Loop invariant invariant:
  $\langle\langle x \mapsto [?x : \text{struct list\_node}*],\ p \mapsto [?p : \text{struct list\_node}*]\rangle_{env}$
  $\langle list(?p,\ ?\beta),\ list(?x,\ ?\gamma)\rangle_{mem}$
  $\langle rev(\alpha) = rev(?\gamma)@?\beta\rangle_{form}\rangle_{config}$

```
while(x != 0) {
  struct list_node *y;
  y = x->next;
  x->next = p;
  p = x;
  x = y;
}
```

Axioms

- List heap pattern
  $\langle\langle list(p,\ \alpha), \sigma\rangle_{mem}\ \langle\phi\rangle_{form}\ C\rangle_{config}$
  $\Leftrightarrow \langle\langle\sigma\rangle_{mem}\ \langle p = 0 \wedge \alpha = \epsilon \wedge \phi\rangle_{form}\ C\rangle_{config}$
  $\vee\langle\langle p \mapsto [?a,\ ?q],\ list(?q,\ ?\alpha),\ \sigma\rangle_{mem}\ \langle\alpha = ?a : ?\alpha \wedge \phi\rangle_{form}\ C\rangle_{config}$
- Sequence reverse
  $rev(\alpha @ \beta) = rev(\beta) @ rev(\alpha)$
  $rev(a) = a$

Examples

- MatchC is available online `http://fsl.cs.uiuc.edu/ml`
- Annotated programs are proven correct completely automatically
- List: append, reverse, copy
- Queue: enqueue, dequeue, transfer
- Tree: mirror, flattening
- Sorting: bubble sort, insertion sort, quick sort, merge sort