

Simple Loop Patterns and Rich Loop Invariants

Marc Sango

AdaCore – CNAM

05-10-2011

Contents

- 1 Motivation
- 2 Principle
- 3 Collection of Patterns
- 4 Statistics in SPARK Projects

Introduction

- Simple loop invariants without quantifiers, concern typically polynomial relation between scalar variables.
- Best invariants with quantifiers and disjunctions by using some predefined templates or in some hint formulas.
- Rich loop invariants with quantifiers and disjunctions from simple patterns for programs with scalar and array variables.

Specification with a Strong Postcondition

package Search **is**

— *Types declaration*

subtype Position **is** Integer **range** 1 .. 10;

type A **is** **array** (Position) **of** Integer;

— *Expression function*

function No_V_In_Range

(T : A; V : Integer; Low,Up : Position) **return** Boolean

is (for all Pos in Low .. Up => T(Pos) /= V);

— *Function declaration*

function Search (T : A; V : Integer) **return** Integer

with Post =>

((Search' Result = 0 **and** **then**

No_V_In_Range(T,V,T' First ,T' Last))

or else

(T(Search' Result) = V **and** **then**

No_V_In_Range(T,V,T' First ,Search' Result - 1))

);

end Search;

Implementation

```
package body Search is
  function Search (T : A; V : Integer) return Integer is
    Pos : Integer := 0;
  begin
    for I in T'Range loop
      — Search Pattern's Loop Invariant.
      pragma Assert
        ( for all J in Position range T'First .. I-1
          => (T(J) /= V) );
      if T (I) = V then
        Pos := I;
        exit;
      end if;
    end loop;
    return Pos;
  end Search;
end Search;
```

Principle

- Translate the loop to intermediate form of loop;
- Identify the patterns in the loop body;
- Propose a precise invariant for each of the patterns;
- Prove that each invariant "captures" its pattern.

Search Pattern

Definition: There are no assignments in the body and the exit condition g_{exit} is a value-preserving expression.

$$g_{exit} = g^{(i)} \quad (\text{PAT1})$$

Loop invariant:

$$Inv(i) \stackrel{def}{=} (\forall j)(seen(j, i) \Rightarrow \neg g_0^{(j)}) \quad (\text{INV1})$$

Update Patterns: Known Single Update

Definition: A scalar variable is only modified in a single assignment, whose guard is a value-preserving expression, and it receives a value-preserving expression.

$$g^{(i)} \rightarrow a := e^{(i)} \quad (\text{PAT2})$$

Loop invariant:

$$\text{Inv}(i) \stackrel{\text{def}}{=} \left(a = a_0 \wedge (\forall j)(\text{seen}(j, i) \Rightarrow \neg g_0^{(j)}) \right. \\ \left. \vee (\exists j) \left(\begin{array}{l} \text{seen}(j, i) \wedge g_0^{(j)} \wedge a = e_0^{(j)} \\ \wedge (\forall k)(\text{between}(j, k, i) \Rightarrow \neg g_0^{(k)}) \end{array} \right) \right) \quad (\text{INV2})$$

Integrate Patterns: Unknown Single Add-Up

Definition: A scalar variable a is modified at most in a single assignment, whose guard is unknown, and it receives its own previous value plus a constant.

$$g^? \rightarrow a := a + K \quad (\text{PAT3})$$

Loop invariant:

$$\text{Inv}(i) \stackrel{\text{def}}{=} \begin{array}{c} (a = a_0) \\ \vee \\ (\exists j)(\text{seen}(j, i) \wedge (\exists k)(1 \leq k \leq \text{tcount}(j) \wedge a = a_0 + K * k)) \end{array} \quad (\text{INV3})$$

Map Patterns: Filter Pattern

Definition: An array variable is only modified in a single assignment, whose guard is a value-preserving expression, at index v , and it receives in each a value-preserving expression. Scalar variable v is incremented in a single assignment, whose guard is the same as the one above.

$$g^{(i)} \rightarrow A[v] := e^{(i)}$$

$$g^{(i)} \rightarrow v := v + 1$$

Exchange Patterns: Known Exchange

Definition: An array variable is only modified in a single assignment, whose guard is a value-preserving expression, at some unknown index, and it receives a value-preserving expression.

$$g^{(i)} \rightarrow A[e^?] := e^{(i)}$$

Note: This pattern includes the case where the index of A is value-preserving but different from i (the loop index).

Summary of Patterns

Categories of patterns	Number of patterns in a category
Search Pattern	1
Update Patterns	6
Integrate Patterns	8
Map Patterns	7
Exchange Patterns	4

- More details in the paper (writing state) explaining our approach.

Statistics in projects written with SPARK(1)

- Tokeneer : Highly secure biometric system (44 loops, 9905 code lines)

Categories of Patterns	Patterns in the category	Number
Search	search	14
Update	Known Single Update	1
Integrate	Known Single Add-Up	1
Map	Certain Single Map	6
	Certain Shifted Single Map	1
	Known Single Map	1
Mixing patterns	Search + Update	6
	Map + Integrate	2
	Update + Integrate	1
	Search + Update + Integrate	1
Not in Database	Loops that are not in our patterns	1
Invariants Free	Loops that don't need Invariants	9

Statistics in projects written with SPARK(2)

- Missile guidance simulator (32 loops, 9892 code lines)

Categories of Patterns	Patterns in the category	Number
Search	search	0
Update	Certain Single Update	4
Integrate	Certain Single Add-Up	7
	Known Multiple Update	1
Map	Certain Multiple Map	1
Exchange	Known Exchange	1
Mixing Patterns	Search + Update	1
	Map + Integrate	2
	Map + Update	1
Not in Database	Loops that are not in our patterns	10
Invariants Free	Loops that don't need Invariants	4

Statistics in projects written with SPARK(3)

- Sparkskein : Cryptographic hash function "Skein" (18 loops, 1599 code lines)

Categories of Patterns	Patterns in the category	Number
Search	search	0
Update	Single Min/Max	1
	Certain/known Single Update	1
Integrate	Integrate	0
Map	Certain Single Map	7
	Certain Shifted Single Map	1
Exchange	Exchange Patterns	0
Mixing	Search + Update + Integrate	1
	Search + Map + Integrate	2
Not in Database	Loops that are not in our patterns	2
Invariants Free	Loops that don't need Invariants	3

Statistics in projects written with SPARK(4)

- Examiner (+ 1000 loops)
- Most loops in this code are while loops and infinite loops using complex data structures
- This is not embedded code, does not correspond to our primary focus.

Conclusion

- Work done : Collections of patterns and their invariants.
- Work doing : Proof that each suggested invariant "captures" its pattern.
- Work to do : Translation into the intermediate semantic parallel loops form.