Program Verification
COSE419, Spring 2024

Hakjoo Oh

Due: 5/30 23:59

Problem 1 (Partial Correctness Verifier) *주어진 프로그램이 명세를 만족하는지 확인하는 함수* `verify_partial_correctness`*를 구현해보자.*

```
verify_partial_correctness : pgm -> bool
```

프로그램은 명세와 함께 하나의 `pgm`*의 요약된 문법 구조(Abstract syntax)는* `syntax.ml`*에 정의되어 있다. 입력 프로그램에는 런타임 오류(Runtime error)가 없다고 가정한다. 텍스트 형식의 프로그램을* `pgm` *타입의 데이터로 변환해주는 파서가 제공된다.*

Examples *아래 예제들은 명세와 구현이 모두 올바른 경우이다. 검증에 성공해야 한다.*

- *Abs:*

```
@pre true
@post 0 <= rv && (rv == x || rv == -x)
int Abs(int x)
{
  if (x < 0) {
    return -x;
  } else {
    return x;
  }
}
```

- *AbsArray:*

```
@pre 0 <= |a0|
@post forall k. (0 <= k && k < |rv| -> rv[k] >= 0)
int[] AbsArray(int[] a0) {
  int[] a;
  int i;
  a := a0;
  for
    @L: 0 <= i && i <= |a| &&
        forall j. (0 <= j && j < i -> a[j] >= 0)
    (i := 0; i < |a|; i := i + 1) {
    if (a[i] < 0) {
      a[i] := -a[i];
    }
  }
  return a;
}
```

- *LinearSearch:*

```
@pre 0 <= l && u < |a|
@post rv <-> exists j. (l <= j && j <= u && a[j] == e)
bool LinearSearch(int[] a, int l, int u, int e) {
  int i;
  for
    @L: l <= i && forall j. (l <= j && j < i -> a[j] != e)
    (i := l; i <= u; i := i + 1)
  {
    if (a[i] == e)
      return true;
  }
  return false;
}
```

- *BubbleSort:*

```
@pre |arr0| >= 0
@post sorted(rv, 0, |rv| - 1)
int[] BubbleSort(int[] arr0) {
  int[] arr;
  int i;
  int j;
  int temp;
  arr := arr0;
  for
    @L1: -1 <= i && i < |arr| && partitioned(arr, 0, i, i+1, |arr|-1) &&
        sorted(arr, i, |arr| - 1)
    (i := |arr| - 1; i > 0; i := i - 1)
  {
    for
      @L2: 1 <= i && i < |arr| && 0 <= j && j <= i
        && partitioned(arr, 0, i, i + 1, |arr| - 1)
        && partitioned(arr, 0, j - 1, j, j)
        && sorted(arr, i, |arr| - 1)
      (j := 0; j < i; j := j + 1)
    {
      if (arr[j] > arr[j + 1]) {
        temp := arr[j];
        arr[j] := arr[j + 1];
        arr[j + 1] := temp;
      }
    }
  }
  return arr;
}
```

- *FindMax:*

```
@pre |a| >= 1
@post (forall k. 0 <= k && k < |a| -> a[k] <= rv) &&
      (exists k. 0 <= k && k < |a| && a[k] == rv)
int FindMax (int[] a)
```

```
{
  int i;
  int m;
  i := 0;
  m := a[0];
  while
    @L: 0 <= i && i <= |a|
        && (forall k. 0 <= k && k < i -> a[k] <= m)
        && |a| >= 1 && (a[0] == m || (exists k. 0 <= k && k < i && a[k] == m))
  (i < |a|)
  {
    if (a[i] > m) { m := a[i]; }
    i := i + 1;
  }
  return m;
}
```

- *BinarySearchRec:*

```
@pre 0 <= l && u < |a| && sorted(a, l, u)
@post rv <-> exists i. (l <= i && i <= u && a[i] == e)
bool BinarySearch(int[] a, int l, int u, int e) {
  int m;
  bool res;
  if (l > u) return false;
  else {
    m := (l + u) / 2;
    if (a[m] == e) return true;
    else if (a[m] < e) {
      call res := BinarySearch(a, m + 1, u, e);
      return res;
    }
    else {
      call res := BinarySearch(a, l, m - 1, e);
      return res;
    }
  }
}
```

명세가 잘못되었거나 프로그램에 오류가 있는 경우에는 검증에 실패해야 한다.

- *구현에 오류가 있는 경우:*

```
@pre |a| >= 1
@post (forall k. 0 <= k && k < |a| -> a[k] <= rv) &&
      (exists k. 0 <= k && k < |a| && a[k] == rv)
int FindMax (int[] a)
{
  int i;
  int m;
  i := 2;  // bug
  m := a[0];
  while
    @L: 0 <= i && i <= |a|
```

```
        && (forall k. 0 <= k && k < i -> a[k] <= m)
        && |a| >= 1
        && (a[0] == m || (exists k. 0 <= k && k < i && a[k] == m))
  (i < |a|)
  {
    if (a[i] > m) { m := a[i]; }
    i := i + 1;
  }
  return m;
}
```

- *명세에 오류가 있는 경우:*

```
@pre |a| >= 1
@post (forall k. 0 <= k && k < |a| -> a[k] <= rv) &&
      (exists k. 0 < k && k < |a| && a[k] == rv) // bug
int FindMax (int[] a)
{
  int i;
  int m;
  i := 0;
  m := a[0];
  while
    @L: 0 <= i && i <= |a|
        && (forall k. 0 <= k && k < i -> a[k] <= m)
        && |a| >= 1
        && (a[0] == m || (exists k. 0 <= k && k < i && a[k] == m))
  (i < |a|)
  {
    if (a[i] > m) { m := a[i]; }
    i := i + 1;
  }
  return m;
}
```

Problem 2 (Termination Prover) *주어진 프로그램이 모든 입력에 대해 항상 종료하는지 여부를 확인하는 함수* `verify_termination`*을 구현해보자.*

```
verify_termination : pgm -> bool
```

Examples *검증에 성공하는 예는 다음과 같다.*

- *LinearSearch:*

```
@pre 0 <= l && u < |a|
@post true
bool LinearSearch(int[] a, int l, int u, int e) {
  int i;
  for
    @L: u < |a|
    # (|a| - i)
    (i := l; i <= u; i := i + 1)
  {
    if (a[i] == e)
      return true;
```

```
  }
  return false;
}
```

- *BinarySearch:*

```
@pre u - l + 1 >= 0
@post true
# (u - l + 1)
bool BinarySearch(int[] a, int l, int u, int e) {
  int m;
  bool res;
  if (l > u) return false;
  else {
    m := (l + u) / 2;
    if (a[m] == e) return true;
    else if (a[m] < e) {
      call res := BinarySearch(a, m + 1, u, e);
      return res;
    }
    else {
      call res := BinarySearch(a, l, m - 1, e);
      return res;
    }
  }
}
```

아래는 검증에 실패하는 경우들이다.

- *LinearSearch:*

```
@pre 0 <= l && u < |a|
@post true
bool LinearSearch(int[] a, int l, int u, int e) {
  int i;
  for
    @L:  u < |a|
    # (|a| - i)
    (i := l; i <= i; i := i + 1)  // bug
  {
    if (a[i] == e)
      return true;
  }
  return false;
}
```

- *BinarySearch:*

```
@pre u - l + 1 >= 0
@post true
# (u - l + 1)
bool BinarySearch(int[] a, int l, int u, int e) {
  int m;
  bool res;
```

```
  if (l > u) return false;
  else {
    m := (l + u) / 2;
    if (a[m] == e) return true;
    else if (a[m] < e) {
      call res := BinarySearch(a, m, u, e); // bug
      return res;
    }
    else {
      call res := BinarySearch(a, l, m - 1, e);
      return res;
    }
  }
}
```