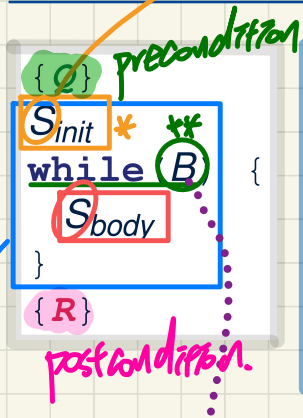
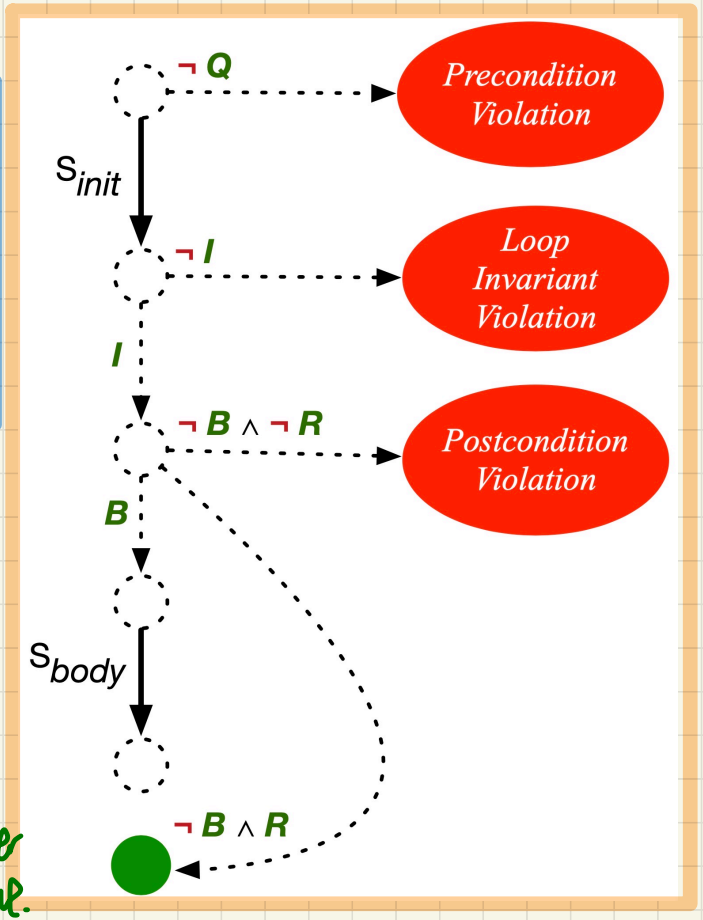
Correctness of Loops: Syntax

programming statements.




```
void myAlgorithm() {
  assert Q; /* Precondition */
  Sinit
  assert I; /* Is LI established? */
  while( B ) {
    Sbody
    assert I; /* Is LI preserved? */
  }
  assert R; /* Postcondition */
}
```



Iterative Algorithm

B: stay condition

¬B: exit condition.

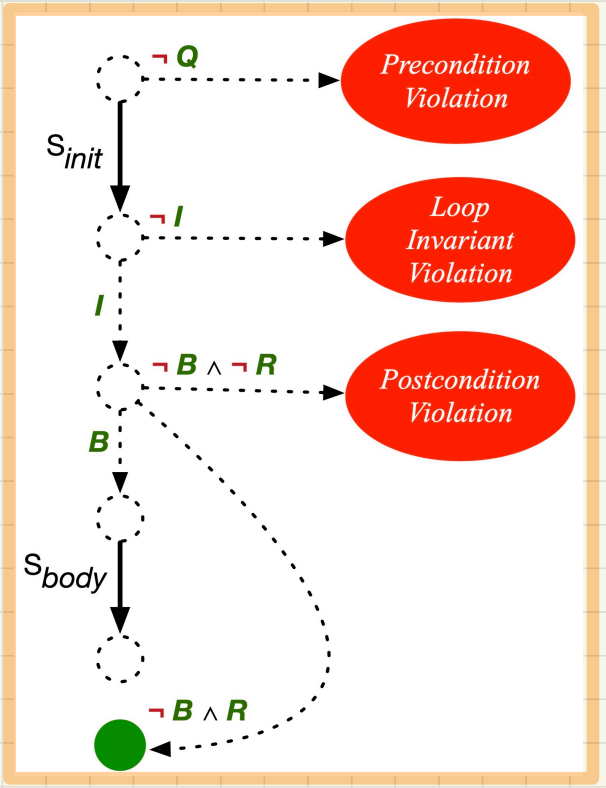
* Initialization/Preparation for iterations.

** As long as B is true, execute Sbody another time.

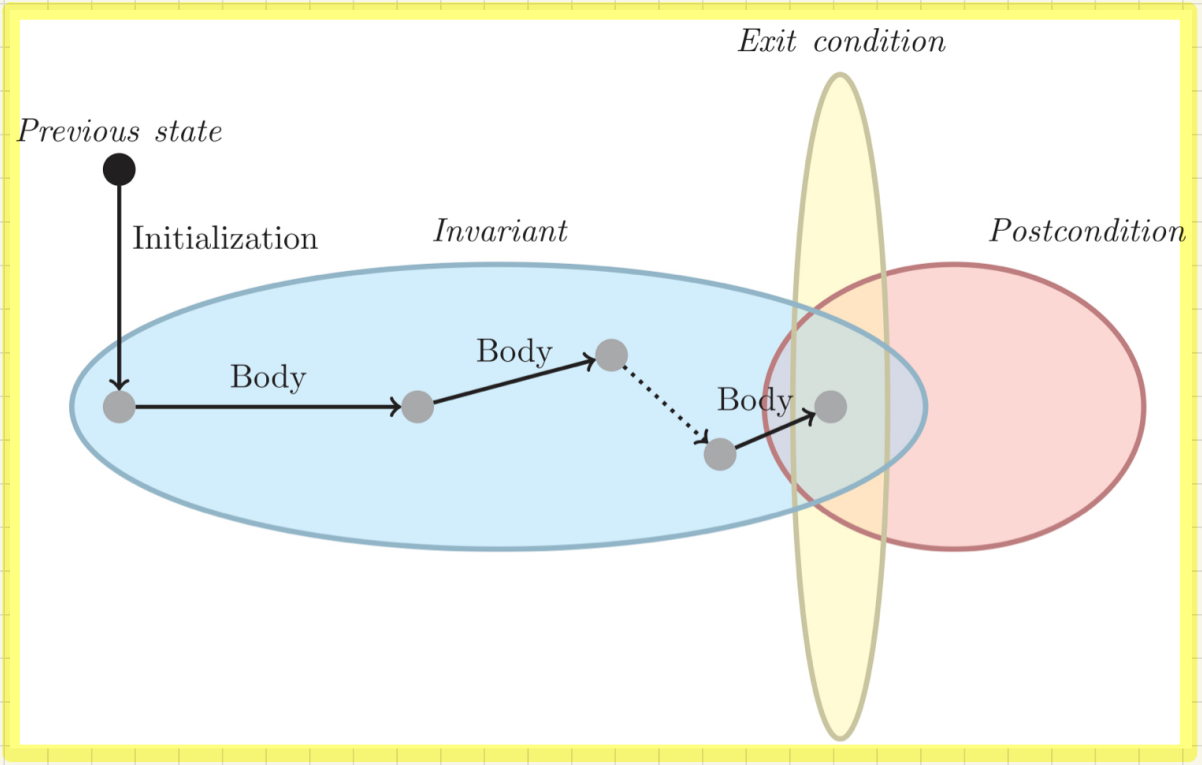
As soon as B is false, exit from the loop.

Correctness of Loops: Example

```
1  void testLI() { /* Assume: integer attribute i */
2    assert i == 1; /* Precondition */
3    assert (1 <= i) && (i <= 6); /* Is LI established? */
4    while (i <= 5) {
5      i = i + 1;
6      assert (1 <= i) && (i <= 6); /* Is LI maintained? */
7    }
8    assert i == 6; /* Postcondition */
9  }
```



Contracts of Loops: Visualization



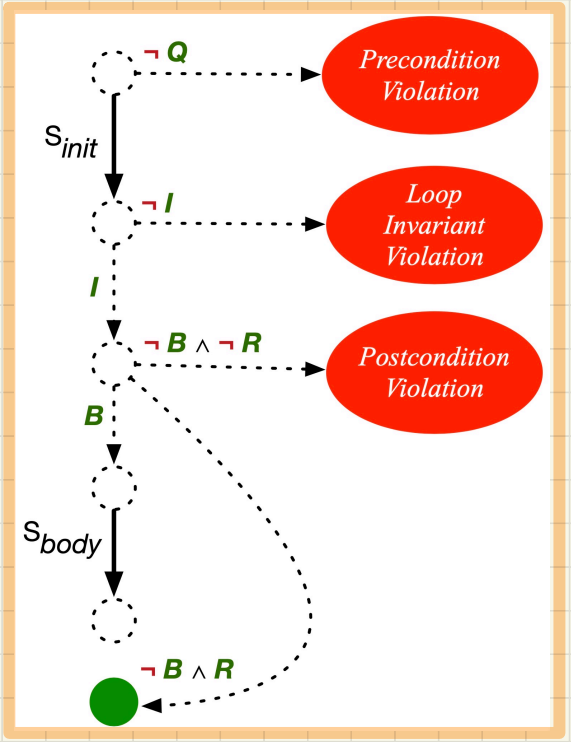
Correctness of Loops: Dijkstra's Shortest-Path Algorithm

Recall: A ***loop invariant*** (***LI***) is a Boolean condition.

- ***LI*** is establisehd before the 1st iteration.
- ***LI*** is preserved **at the end of** each subsequent iteration.

The (iterative) Dijkstra's algorithm has ***LI***:

> For every vertext u that has already been removed from the priority queue Q (i.e., u is considered visited), $D(u)$ equals the ***true*** shortest-path distance from source s to u.



Dijkstra's Shortest Path Algorithm: Negative Weights

The (iterative) Dijkstra's algorithm has ***LI***:

> For every vertext u that has already been removed from the priority queue Q (i.e., u is considered visited), $D(u)$ equals the ***true*** shortest-path distance from source s to u.