Invariant Based Programming

Ralph-Johan Back

Abo Akademi University, Dept. of Information Technologies

Third Workshop on Formal and Automated Theorem Proving and Applications January 29-30, 2010, Belgrade, Serbia

January 29, 2010

Teaching programming

- Teaching programming is known to be very difficult
- Studies show that students have big difficulties in
  - understanding how a program works
  - designing a program
  - checking whether the program works correctly,
  - etc
- Introductory programming courses usually focus on teaching programming through a programming language
  - teach syntax of a standard programming language like Java, C, to Python,
  - show how to implement some simple algorithms in the chosen programming language
  - teach how to run and test these algorithms, and how to debug them

Problems with this approach

- Most of the time is spent on learning the special features of syntax
    - less time for learning how to design algorithms
    - and implement them correctly as executable programs
- The basic idea of how an algorithm / program works, and why, often remains unclear
    - the execution model can be quite complex, e.g., object-oriented systems
    - the code - test - debug cycle gives little insight into the overall working of the program
- Students are taught from the very beginning that software bugs are unavoidable
    - guess and test approach to solving programming problems
    - low quality software is acceptable
    - no tools are given for producing higher quality software

Programming as mathematics

- Our purpose here is to show how to teach programming as a mathematics course
    - without a specific programming language
    - using a graphical presentation of programs
    - focusing on building programs that are proved correct mathematically
- Place in curriculum
    - Not necessarily the first course on programming (which could be, e.g., Python), but maybe the second course
    - Could be taught both in introductory CS courses and in high school
    - Teaches students to understand how programs work, how to design programs, and how to analyze their correctness
- Need not be more difficult than any ordinary high school math course (but maybe not much easier either)

Goal here

- Presentation next is inteded to show how a short course on formal methods in programming could be given to
  - high school students (as a mathematics course).
  - first year CS students (as a course on formal methods in programming)
- The lecture is here very compressed, in a real course the material would be spread over a number of lectures, with a lot of examples and class excercises
- Have taught this material to math teachers in Austria (Graz) last year.

Main point and a caveat

- Main point 1: We teach formal methods using the *invariant based programming* approach
- Main point 2: this is more or less **ALL** the theory that is needed for teaching formal methods in programming
- Caveat: but we assume that the students have a basic familiarity with
  - logic (propositional calculus and predicate calculus basics)
  - using predicate calculus to express mathematical properties
  - reasoning about logical properties
- We teach this in a preceeding course, called *structured derivations* (essentially, how to use practical logic in mathematics).

Outline

1. Programming as mathematics

2. Mathematics of programming

   Situations

   Programs

   Correctness

   Invariant diagrams

   Consistency

   Termination and liveness

3. Invariant based programming

4. Case study

Situations

A situation describes a specific set of circumstances of a system. A situation

- has a name,
- a list of attributes that are used to observe the system state,
- a list of constraints that restrict the possible values that the attributes can take,
- concrete examples of the situation (a figure, a graph, a table, some text, etc.),
- a list of properties that hold in this situation,
- proofs of the properties stated in the situation.

Situations

A situation describes a specific set of circumstances of a system. A situation

- has a name,
- a list of attributes that are used to observe the system state,
- a list of constraints that restrict the possible values that the attributes can take,
- concrete examples of the situation (a figure, a graph, a table, some text, etc.),
- a list of properties that hold in this situation,
- proofs of the properties stated in the situation.

Situations

A situation describes a specific set of circumstances of a system. A situation

- has a name,
- a list of attributes that are used to observe the system state,
- a list of constraints that restrict the possible values that the attributes can take,
- concrete examples of the situation (a figure, a graph, a table, some text, etc.),
- a list of properties that hold in this situation,
- proofs of the properties stated in the situation.

Situations

A situation describes a specific set of circumstances of a system. A situation

- has a name,
- a list of attributes that are used to observe the system state,
- a list of constraints that restrict the possible values that the attributes can take,
- concrete examples of the situation (a figure, a graph, a table, some text, etc.),
- a list of properties that hold in this situation,
- proofs of the properties stated in the situation.

Situations

A situation describes a specific set of circumstances of a system. A situation

- has a name,
- a list of attributes that are used to observe the system state,
- a list of constraints that restrict the possible values that the attributes can take,
- concrete examples of the situation (a figure, a graph, a table, some text, etc.),
- a list of properties that hold in this situation,
- proofs of the properties stated in the situation.

Situations

A situation describes a specific set of circumstances of a system. A situation

- has a name,
- a list of attributes that are used to observe the system state,
- a list of constraints that restrict the possible values that the attributes can take,
- concrete examples of the situation (a figure, a graph, a table, some text, etc.),
- a list of properties that hold in this situation,
- proofs of the properties stated in the situation.

Example situation

Right triangle

var a, b, c, h : *real*

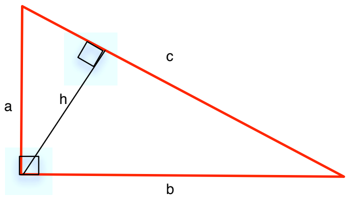
- the triangle is right, with hypotenuse c and catheters a and b,
- h is the height of the triangle on the hypotenuse
- h divides the hypotenuse in the proportion 3:7

● $a^2 + b^2 = c^2$

⊩ {Pythagoras' theorem}

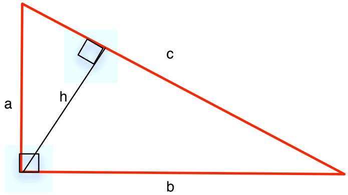
● $\frac{a}{b} = \frac{\sqrt{3}}{\sqrt{7}}$

⊩ . . . proof . . .



Example situation

Right triangle



var a, b, c, h : *real*

- the triangle is right, with hypotenuse c and catheters a and b,

- h is the height of the triangle on the hypotenuse

- h divides the hypotenuse in the proportion 3:7

● $a^2 + b^2 = c^2$

⊩ {Pythagoras' theorem}

● $\frac{a}{b} = \frac{\sqrt{3}}{\sqrt{7}}$

⊩ … proof …

Example situation

Right triangle

var a, b , c , h : *real*

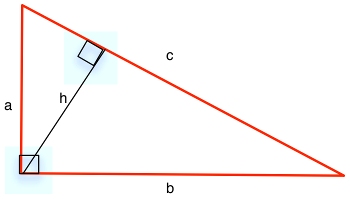
- the triangle is right, with hypotenuse c and catheters a and b,
- h is the height of the triangle on the hypotenuse
- h divides the hypotenuse in the proportion 3:7

• $a^2 + b^2 = c^2$

⊩ {Pythagoras' theorem}

• $\frac{a}{b} = \frac{\sqrt{3}}{\sqrt{7}}$

⊩ . . . proof . . .



Example situation

Right triangle

var a, b, c, h : *real*

- the triangle is right, with hypotenuse c and catheters a and b,

- h is the height of the triangle on the hypotenuse

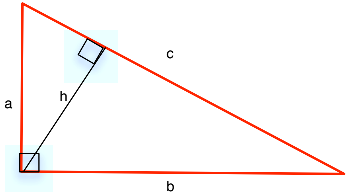
- h divides the hypotenuse in the proportion 3:7

• $a^2 + b^2 = c^2$

⊩ {Pythagoras' theorem}

• $\frac{a}{b} = \frac{\sqrt{3}}{\sqrt{7}}$

⊩ . . . proof . . .



Example situation

Right triangle

var a, b, c, h : *real*

- the triangle is right, with hypotenuse c and catheters a and b,

- h is the height of the triangle on the hypotenuse

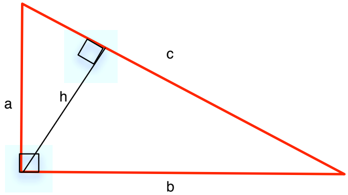
- h divides the hypotenuse in the proportion 3:7

• $a^2 + b^2 = c^2$

$\Vdash$ {Pythagoras' theorem}

• $\frac{a}{b} = \frac{\sqrt{3}}{\sqrt{7}}$

$\Vdash$ . . . proof . . .



Example situation

Right triangle

var *a*, *b* , *c* , *h* : *real*

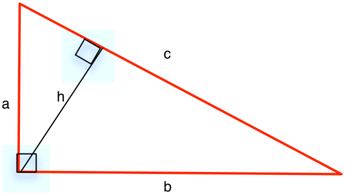
- the triangle is right, with hypotenuse *c* and catheters *a* and *b*,
- *h* is the height of the triangle on the hypotenuse
- *h* divides the hypotenuse in the proportion 3:7

● $a^2 + b^2 = c^2$

⊩ {Pythagoras' theorem}

● $\frac{a}{b} = \frac{\sqrt{3}}{\sqrt{7}}$

⊩ . . . proof . . .



Example situation

Right triangle

var $a, b, c, h : real$

- the triangle is right, with hypotenuse c and catheters a and b,

- h is the height of the triangle on the hypotenuse

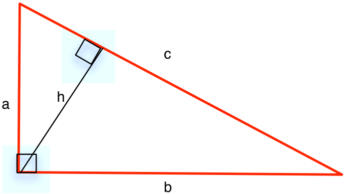
- h divides the hypotenuse in the proportion 3:7

• $a^2 + b^2 = c^2$

⊩ {Pythagoras' theorem}

• $\frac{a}{b} = \frac{\sqrt{3}}{\sqrt{7}}$

⊩ ... proof ...



Example situation

Right triangle

var $a,\ b\ ,\ c\ ,\ h\ :\ real$

- the triangle is right, with hypotenuse c and catheters a and b,

- h is the height of the triangle on the hypotenuse

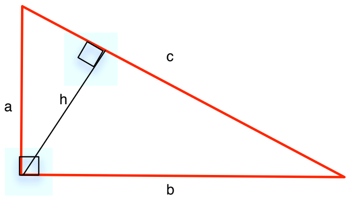
- h divides the hypotenuse in the proportion 3:7

• $a^2 + b^2 = c^2$

$\Vdash$ {Pythagoras' theorem}

• $\frac{a}{b} = \dfrac{\sqrt{3}}{\sqrt{7}}$

$\Vdash$ . . . proof . . .



Situation in general

Situation name

var	. . .	list of variables . . .	. . . example of the situation . . .
–	. . .	constraint . . .	
–	. . .	another constraint . . .	. . . another example . . .
⋮			
●	. . .	a property	
$\Vdash$	. . .	proof . . .	
●	. . .	another property	
$\Vdash$	. . .	proof . . .	
⋮			

Situation in general

Situation name

var . . . list of variables . . .

- . . . constraint . . .

- . . . another constraint . . .

$\vdots$

● . . . a property

$\Vdash$ . . . proof . . .

● . . . another property

$\Vdash$ . . . proof . . .

$\vdots$

. . . example of the situation . . .

. . . another example . . .

Situation in general

Situation name

var	. . . list of variables . . .
-	. . . constraint . . .
-	. . . another constraint . . .
⋮	
•	. . . a property
⊩	. . . proof . . .
•	. . . another property
⊩	. . . proof . . .
⋮	

. . . example of the situation . . .

. . . another example . . .

Situation in general

Situation name

var . . . list of variables . . .
- . . . constraint . . .
- . . . another constraint . . .
⋮
● . . . a property
⊩ . . . proof . . .
● . . . another property
⊩ . . . proof . . .
⋮

. . . example of the situation . . .

. . . another example . . .

Situation in general

Situation name

var	. . . list of variables . . .	. . . example of the situation . . .
-	. . . constraint . . .	
-	. . . another constraint . . .	. . . another example . . .
⋮		
•	. . . a property	
⊩	. . . proof . . .	
•	. . . another property	
⊩	. . . proof . . .	
⋮		

Situation in general

Situation name

var ... list of variables ...

- ... constraint ...

- ... another constraint ...

$\vdots$

● ... a property

$\Vdash$ ... proof ...

● ... another property

$\Vdash$ ... proof ...

$\vdots$

... example of the situation ...

... another example ...

Situation in general

<u>Situation name</u>

var	. . . list of variables . . .	. . . example of the situation . . .
-	. . . constraint . . .	
-	. . . another constraint . . .	. . . another example . . .
⋮		
●	. . . a property	
$\Vdash$	. . . proof . . .	
●	. . . another property	
$\Vdash$	. . . proof . . .	
⋮		

Situation in general

Situation name

var	. . . list of variables . . .	. . . example of the situation . . .
-	. . . constraint . . .	
-	. . . another constraint . . .	. . . another example . . .
⋮		
•	. . . a property	
⊩	. . . proof . . .	
•	. . . another property	
⊩	. . . proof . . .	
⋮		

Situation in general

Situation name

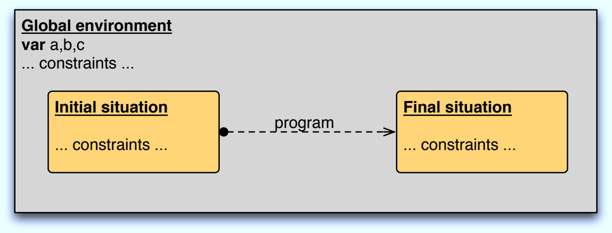
var	. . . list of variables . . .	. . . example of the situation . . .
-	. . . constraint . . .	
-	. . . another constraint . . .	. . . another example . . .
⋮		
•	. . . a property	
⊩	. . . proof . . .	
•	. . . another property	
⊩	. . . proof . . .	
⋮		

Outline

1. Programming as mathematics
2. Mathematics of programming
   - Situations
   - Programs
   - Correctness
   - Invariant diagrams
   - Consistency
   - Termination and liveness
3. Invariant based programming
4. Case study

Programs

A program can be seen as an activity that takes us from some given initial situation to a desired final situation



Global environment
var a,b,c
... constraints ...

Initial situation
... constraints ...

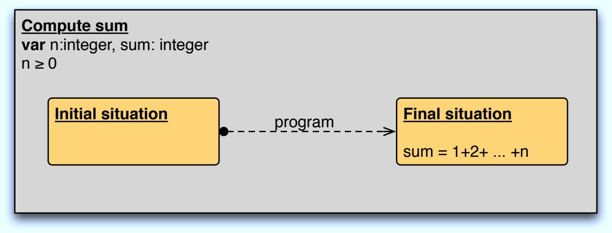
program

Final situation
... constraints ...

Here a, b, c are program variables declared in the environment.
Can consider program variables as attributes (observations) of the program state

Example: compute sum

Compute the sum of the first *n* integers.



Compute sum

var n:integer, sum: integer

n ≥ 0

Initial situation

program

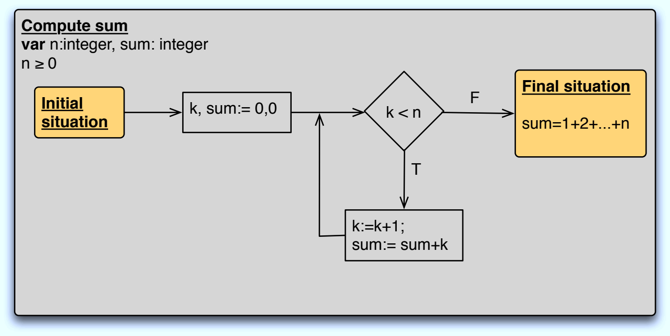
Final situation

sum = 1+2+ ... +n

Program variables n and sum are defined in the environment, constrained by $n \geq 0$*.*

The program computes the sum of the first n integers and assigns it to the variable sum.

Program as flow chart



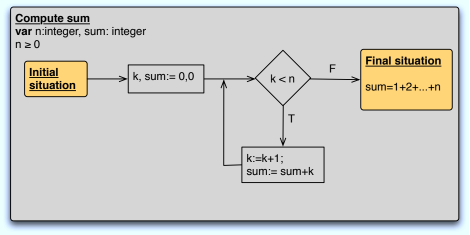
The program first initializes k to 0 and *sum* to 0. Then, it tests whether $k < n$. If this is true, then k is increase by 1, and *sum* is increase by k, and we repeat the test. If $k < n$ is false, then we are finished.

Outline

1. Programming as mathematics
2. Mathematics of programming
    - Situations
    - Programs
    - Correctness
    - Invariant diagrams
    - Consistency
    - Termination and liveness
3. Invariant based programming
4. Case study

Is the program correct

- Traditional method to check correctness for simple programs is by simulating execution by hand.

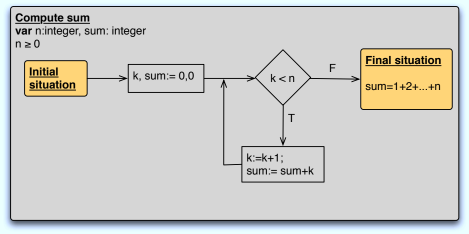


-

	n	k	sum
1	3	0	0
2	3	1	1
3	3	2	3
4	3	3	6

Is the program correct

- Traditional method to check correctness for simple programs is by simulating execution by hand.



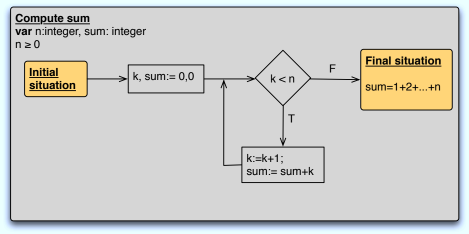
Compute sum
var n:integer, sum: integer
n ≥ 0

-

	n	k	sum
1	3	0	0
2	3	1	1
3	3	2	3
4	3	3	6

Is the program correct

- Traditional method to check correctness for simple programs is by simulating execution by hand.

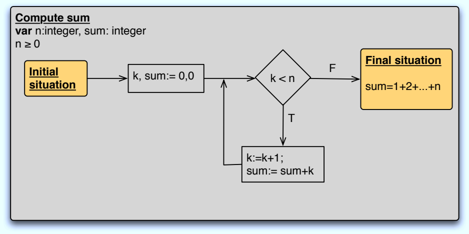


Compute sum
var n:integer, sum: integer
n ≥ 0

	n	k	sum
1	3	0	0
2	3	1	1
3	3	2	3
4	3	3	6

Is the program correct

- Traditional method to check correctness for simple programs is by simulating execution by hand.



Compute sum
var n:integer, sum: integer
n ≥ 0

Initial situation

k, sum:= 0,0

k < n

F

T

k:=k+1;
sum:= sum+k

Final situation

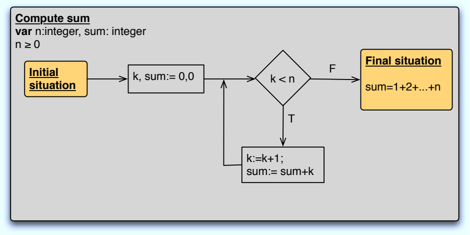
sum=1+2+...+n

-

	n	k	sum
1	3	0	0
2	3	1	1
3	3	2	3
4	3	3	6

Is the program correct

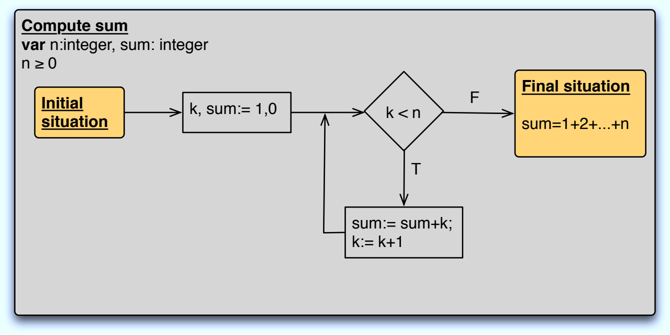
- Traditional method to check correctness for simple programs is by simulating execution by hand.



	n	k	sum
1	3	0	0
2	3	1	1
3	3	2	3
4	3	3	6

Alternative program

- But what about the following program, is it correct.



Compute sum
var n:integer, sum: integer
n ≥ 0

Initial situation

k, sum:= 1,0

k < n

F

T

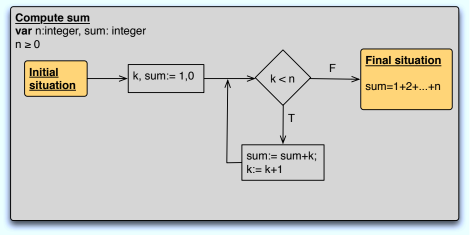
sum:= sum+k;
k:= k+1

Final situation

sum=1+2+...+n

Is the program correct

- Actually, no



Compute sum
var n:integer, sum: integer
n ≥ 0

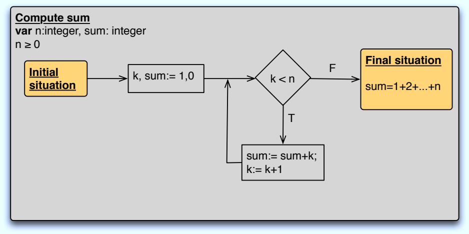
-

	n	k	sum
1	3	1	0
2	3	2	1
3	3	3	3

- wrong result!

Is the program correct

- Actually, no



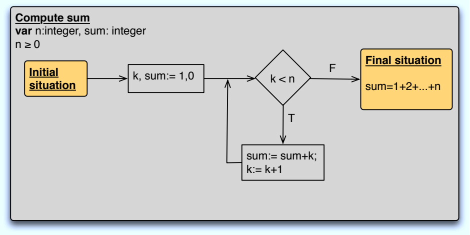
-

	n	k	sum
1	3	1	0
2	3	2	1
3	3	3	3

- wrong result!

Is the program correct

- Actually, no



Compute sum
var n:integer, sum: integer
n ≥ 0

Initial situation

k, sum:= 1,0

k < n

F

T

sum:= sum+k;
k:= k+1

Final situation

sum=1+2+...+n

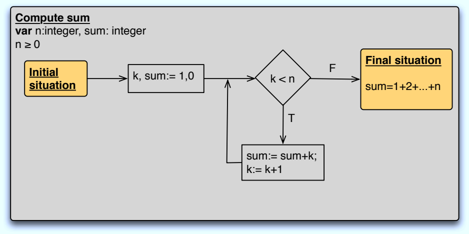
-

	n	k	sum
1	3	1	0
2	3	2	1
3	3	3	3

- wrong result!

Is the program correct

- Actually, no



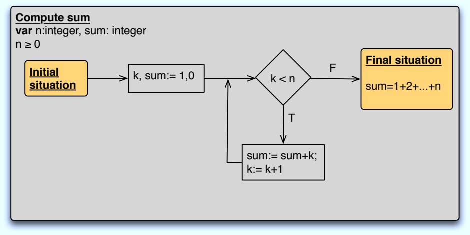
-

	n	k	sum
1	3	1	0
2	3	2	1
3	3	3	3

- wrong result!

Is the program correct

- Actually, no



-

	n	k	sum
1	3	1	0
2	3	2	1
3	3	3	3

- wrong result!

What is correctness

- The program is *correct*, if the following holds:
  - whenever the program is started in an initial situation,
  - then it eventually terminates in a final situation
  - with values for program variables that satisfy all constraints of the final situation
- The summation program is correct, if the following holds:
  - whenever $n \geq 0$ holds initially,
  - then the program eventually terminates in the final situation
  - where $sum = 1 + 2 + \ldots + n$

What is correctness

- The program is *correct*, if the following holds:
  - whenever the program is started in an initial situation,
  - then it eventually terminates in a final situation
  - with values for program variables that satisfy all constraints of the final situation
- The summation program is correct, if the following holds:
  - whenever $n \geq 0$ holds initially,
  - then the program eventually terminates in the final situation
  - where $sum = 1 + 2 + \ldots + n$

What is correctness

- The program is *correct*, if the following holds:
  - whenever the program is started in an initial situation,
  - then it eventually terminates in a final situation
  - with values for program variables that satisfy all constraints of the final situation
- The summation program is correct, if the following holds:
  - whenever $n \geq 0$ holds initially,
  - then the program eventually terminates in the final situation
  - where $sum = 1 + 2 + \ldots + n$

What is correctness

- The program is *correct*, if the following holds:
  - whenever the program is started in an initial situation,
  - then it eventually terminates in a final situation
  - with values for program variables that satisfy all constraints of the final situation
- The summation program is correct, if the following holds:
  - whenever $n \geq 0$ holds initially,
  - then the program eventually terminates in the final situation
  - where $sum = 1 + 2 + \ldots + n$

What is correctness

- The program is *correct*, if the following holds:
  - whenever the program is started in an initial situation,
  - then it eventually terminates in a final situation
  - with values for program variables that satisfy all constraints of the final situation
- The summation program is correct, if the following holds:
  - whenever $n \geq 0$ holds initially,
  - then the program eventually terminates in the final situation
  - where $sum = 1 + 2 + \ldots + n$

What is correctness

- The program is *correct*, if the following holds:
  - whenever the program is started in an initial situation,
  - then it eventually terminates in a final situation
  - with values for program variables that satisfy all constraints of the final situation
- The summation program is correct, if the following holds:
  - whenever $n \geq 0$ holds initially,
  - then the program eventually terminates in the final situation
  - where $sum = 1 + 2 + \ldots + n$

What is correctness

- The program is *correct*, if the following holds:
  - whenever the program is started in an initial situation,
  - then it eventually terminates in a final situation
  - with values for program variables that satisfy all constraints of the final situation
- The summation program is correct, if the following holds:
  - whenever $n \geq 0$ holds initially,
  - then the program eventually terminates in the final situation
  - where $sum = 1 + 2 + \ldots + n$

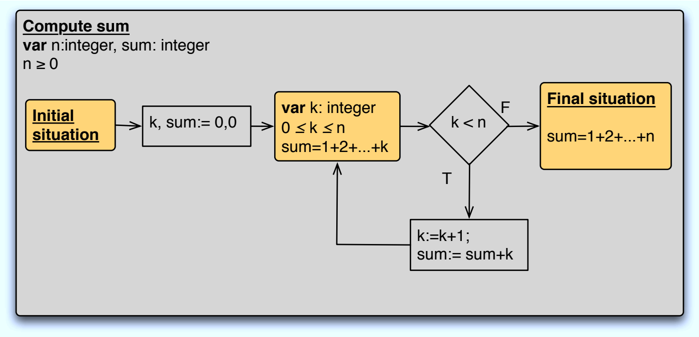
What is correctness

- The program is *correct*, if the following holds:
  - whenever the program is started in an initial situation,
  - then it eventually terminates in a final situation
  - with values for program variables that satisfy all constraints of the final situation
- The summation program is correct, if the following holds:
  - whenever $n \geq 0$ holds initially,
  - then the program eventually terminates in the final situation
  - where $sum = 1 + 2 + \ldots + n$

Checking correctness

- There are infinitely many possible executions in the sum program (one for each value on n),
  - therefore not possible to check that the program works correctly by just testing the program for each n,
- But we can prove mathematically that the program works correctly.
- Proving correctness requires that we add an intermediate situation (a *loop invariant*) to the program
  - The loop invariant corresponds to an induction hypothesis
  - It describes the situation at the indicated point in the loop

Loop invariant



Outline

1. Programming as mathematics

2. Mathematics of programming

    - Situations
    - Programs
    - Correctness
    - Invariant diagrams
    - Consistency
    - Termination and liveness

3. Invariant based programming

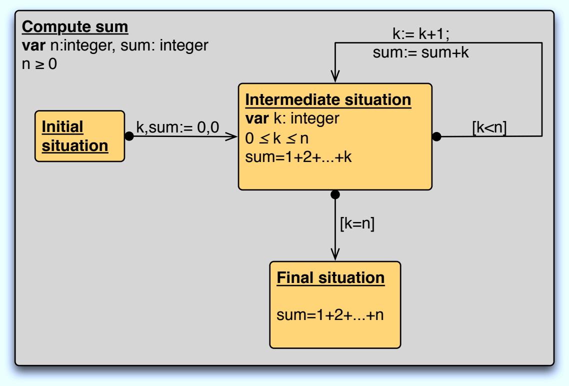
4. Case study

Nested invariant diagram

- In stead of flow charts, the program can be described as a *(nested) invariant diagram.* This describes the program in terms of
    - a collection of *situations* that can occur during program execution, and
    - a collection of *transitions* between these situations
- The situations can be divided into *initial situations*, *final situations*, and *intermediate situations.*
- A situation can be nested inside another situation:
    - the nested situation inherits the constraints of all enclosing situations
- An invariant diagram contains all the information that is needed to prove that the program is correct.

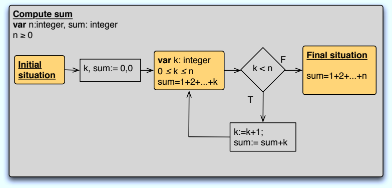
Sum program as invariant diagram

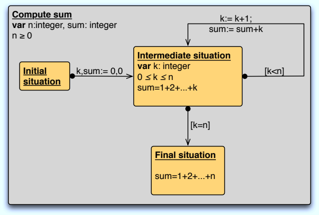
.



In the intermediate situation, the sum of the first k integers has been computed.

Flow chart versus invariant diagram





- Statements written on arrows, rather than in boxes
- Guards on arrows
- Situations can be nested

Transitions

- A *transition* is an arrow from one situation to another, where the arrow is labelled a statement
- A statement can be either:
  - a *guard* of the form $[b]$: the transition is only taken if the condition b holds for the present values of program variables
  - an *assignment* of the form $x_1, \ldots, x_n := e_1, \ldots, e_n$: assigns the value of expression e_i to the variable x_i, for $i = 1, \ldots, n$
  - a sequential composition $A_1; A_2; \ldots; A_k$ of guards and assignments

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y)$:	T
$x := x + y$;	$(3, 2)$
$y := y + 1$;	$(3, 3)$
$(x = y)$:	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y)$:	T
$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(6, 5)$
$(x = y)$:	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y)$;	T
$x := x + y$;	$(3, 2)$
$y := y + 1$;	$(3, 3)$
$(x = y)$;	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y)$;	T
$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(6, 5)$
$(x = y)$;	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y)$;	T
$x := x + y$;	$(3, 2)$
$y := y + 1$;	$(3, 3)$
$(x = y)$;	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y)$;	T
$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(6, 5)$
$(x = y)$;	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y);$	T
$x := x + y;$	$(3, 2)$
$y := y + 1;$	$(3, 3)$
$(x = y);$	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y);$	T
$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(6, 5)$
$(x = y);$	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)	statement	(x, y)
	$(1, 2)$		$(2, 4)$
$(x < y)$;	T	$(x < y)$;	T
$x := x + y$;	$(3, 2)$	$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(3, 3)$	$y := y + 1$;	$(6, 5)$
$(x = y)$;	T	$(x = y)$;	F
$x := x - y$	$(0, 3)$	$x := x - y$	
	enabled		not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)	statement	(x, y)
	$(1, 2)$		$(2, 4)$
$(x < y);$	T	$(x < y);$	T
$x := x + y;$	$(3, 2)$	$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(3, 3)$	$y := y + 1;$	$(6, 5)$
$(x = y);$	T	$(x = y);$	F
$x := x - y$	$(0, 3)$	$x := x - y$	
	enabled		not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)	statement	(x, y)
	$(1, 2)$		$(2, 4)$
$(x < y)$;	T	$(x < y)$;	T
$x := x + y$;	$(3, 2)$	$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(3, 3)$	$y := y + 1$;	$(6, 5)$
$(x = y)$;	T	$(x = y)$;	F
$x := x - y$	$(0, 3)$	$x := x - y$	
	enabled		not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y);$	T
$x := x + y;$	$(3, 2)$
$y := y + 1;$	$(3, 3)$
$(x = y);$	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y);$	T
$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(6, 5)$
$(x = y);$	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)	statement	(x, y)
	$(1, 2)$		$(2, 4)$
$(x < y)$;	T	$(x < y)$;	T
$x := x + y$;	$(3, 2)$	$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(3, 3)$	$y := y + 1$;	$(6, 5)$
$(x = y)$;	T	$(x = y)$;	F
$x := x - y$	$(0, 3)$	$x := x - y$	
	enabled		not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)	statement	(x, y)
	$(1, 2)$		$(2, 4)$
$(x < y);$	T	$(x < y);$	T
$x := x + y;$	$(3, 2)$	$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(3, 3)$	$y := y + 1;$	$(6, 5)$
$(x = y);$	T	$(x = y);$	F
$x := x - y$	$(0, 3)$	$x := x - y$	
	enabled		not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y);$	T
$x := x + y;$	$(3, 2)$
$y := y + 1;$	$(3, 3)$
$(x = y);$	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y);$	T
$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(6, 5)$
$(x = y);$	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y)$;	T
$x := x + y$;	$(3, 2)$
$y := y + 1$;	$(3, 3)$
$(x = y)$;	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y)$;	T
$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(6, 5)$
$(x = y)$;	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y);$	T
$x := x + y;$	$(3, 2)$
$y := y + 1;$	$(3, 3)$
$(x = y);$	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y);$	T
$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(6, 5)$
$(x = y);$	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y);$	T
$x := x + y;$	$(3, 2)$
$y := y + 1;$	$(3, 3)$
$(x = y);$	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y);$	T
$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(6, 5)$
$(x = y);$	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y);$	T
$x := x + y;$	$(3, 2)$
$y := y + 1;$	$(3, 3)$
$(x = y);$	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y);$	T
$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(6, 5)$
$(x = y);$	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y);$	T
$x := x + y;$	$(3, 2)$
$y := y + 1;$	$(3, 3)$
$(x = y);$	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y);$	T
$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(6, 5)$
$(x = y);$	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)	statement	(x, y)
	$(1, 2)$		$(2, 4)$
$(x < y);$	T	$(x < y);$	T
$x := x + y;$	$(3, 2)$	$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(3, 3)$	$y := y + 1;$	$(6, 5)$
$(x = y);$	T	$(x = y);$	F
$x := x - y$	$(0, 3)$	$x := x - y$	
	enabled		not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)	statement	(x, y)
	$(1, 2)$		$(2, 4)$
$(x < y)$;	T	$(x < y)$;	T
$x := x + y$;	$(3, 2)$	$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(3, 3)$	$y := y + 1$;	$(6, 5)$
$(x = y)$;	T	$(x = y)$;	F
$x := x - y$	$(0, 3)$	$x := x - y$	
	enabled		not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y);$	T
$x := x + y;$	$(3, 2)$
$y := y + 1;$	$(3, 3)$
$(x = y);$	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y);$	T
$x := x + y;$	$(6, 4)$
$y := y + 1;$	$(6, 5)$
$(x = y);$	F
$x := x - y$	
	not enabled

Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y)$;	T
$x := x + y$;	$(3, 2)$
$y := y + 1$;	$(3, 3)$
$(x = y)$;	T
$x := x - y$	$(0, 3)$
	enabled

statement	(x, y)
	$(2, 4)$
$(x < y)$;	T
$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(6, 5)$
$(x = y)$;	F
$x := x - y$	
	not enabled

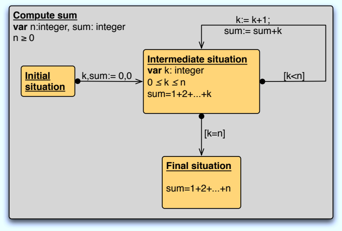
Executing a transition

- The transition is executed one guard or assignment at a time
- If we reach a guard which is not satisfied, then the whole transition is disabled (i.e., will not be executed)

statement	(x, y)
	$(1, 2)$
$(x < y)$;	T
$x := x + y$;	$(3, 2)$
$y := y + 1$;	$(3, 3)$
$(x = y)$;	T
$x := x - y$	$(0, 3)$
	enabled

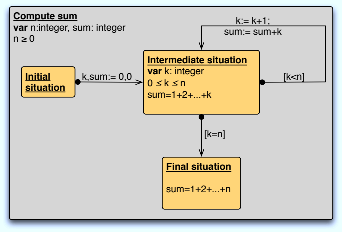
statement	(x, y)
	$(2, 4)$
$(x < y)$;	T
$x := x + y$;	$(6, 4)$
$y := y + 1$;	$(6, 5)$
$(x = y)$;	F
$x := x - y$	
	not enabled

Program execution



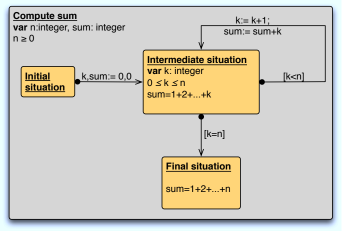
- Starts in initial situation, with values for the program variables
- execution follows the arrows, from one situation to the next
- an arrow can only be traversed if the guards on the arrow are satisfied (transition is *enabled*)
- if two or more transitions are enabled in the same situation, one of them is chosen (non deterministically)
- the statement on the selected arrow is then executed

Program execution



- Starts in initial situation, with values for the program variables
- execution follows the arrows, from one situation to the next
- an arrow can only be traversed if the guards on the arrow are satisfied (transition is *enabled*)
- if two or more transitions are enabled in the same situation, one of them is chosen (non deterministically)
- the statement on the selected arrow is then executed

Program execution



Compute sum
var n:integer, sum: integer
n ≥ 0

Initial situation

k,sum:= 0,0

Intermediate situation
var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k:= k+1;
sum:= sum+k

[k<n]

[k=n]

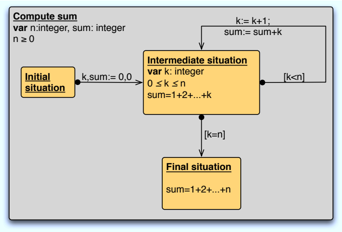
Final situation

sum=1+2+...+n

- Starts in initial situation, with values for the program variables
- execution follows the arrows, from one situation to the next

- an arrow can only be traversed if the guards on the arrow are satisfied (transition is *enabled*)
- if two or more transitions are enabled in the same situation, one of them is chosen (non deterministically)
- the statement on the selected arrow is then executed

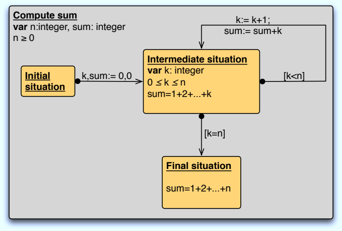
Program execution



- Starts in initial situation, with values for the program variables
- execution follows the arrows, from one situation to the next

- an arrow can only be traversed if the guards on the arrow are satisfied (transition is *enabled*)
- if two or more transitions are enabled in the same situation, one of them is chosen (non deterministically)
- the statement on the selected arrow is then executed

Program execution



Compute sum
var n:integer, sum: integer
n ≥ 0

Initial situation

k,sum:= 0,0

Intermediate situation
var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k:= k+1;
sum:= sum+k

[k<n]

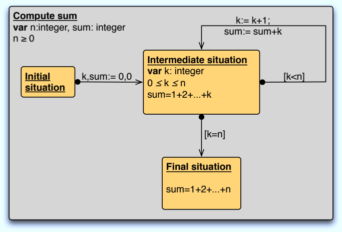
[k=n]

Final situation
sum=1+2+...+n

- Starts in initial situation, with values for the program variables
- execution follows the arrows, from one situation to the next

- an arrow can only be traversed if the guards on the arrow are satisfied (transition is *enabled*)
- if two or more transitions are enabled in the same situation, one of them is chosen (non deterministically)
- the statement on the selected arrow is then executed

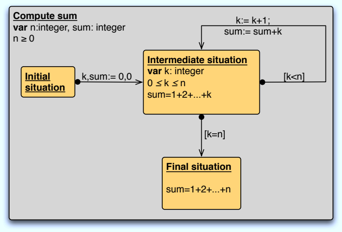
Program execution



- Starts in initial situation, with values for the program variables
- execution follows the arrows, from one situation to the next

- an arrow can only be traversed if the guards on the arrow are satisfied (transition is *enabled*)
- if two or more transitions are enabled in the same situation, one of them is chosen (non deterministically)
- the statement on the selected arrow is then executed

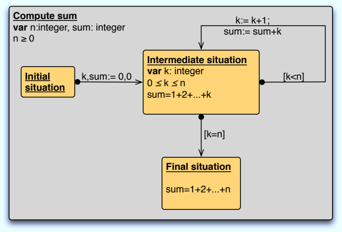
Program execution



- Starts in initial situation, with values for the program variables
- execution follows the arrows, from one situation to the next

- an arrow can only be traversed if the guards on the arrow are satisfied (transition is *enabled*)
- if two or more transitions are enabled in the same situation, one of them is chosen (non deterministically)
- the statement on the selected arrow is then executed

Program execution



- Starts in initial situation, with values for the program variables
- execution follows the arrows, from one situation to the next

- an arrow can only be traversed if the guards on the arrow are satisfied (transition is *enabled*)
- if two or more transitions are enabled in the same situation, one of them is chosen (non deterministically)
- the statement on the selected arrow is then executed

Termination

Execution terminates when we reach a situation from which no transition is enabled

- if the situation is a final situation, then the program terminates *normally*
- if the situation is not a final situation, then the program has *deadlocked.*

Termination

Execution terminates when we reach a situation from which no transition is enabled

- if the situation is a final situation, then the program terminates *normally*
- if the situation is not a final situation, then the program has *deadlocked.*

Proving correctness

To prove that the program is correct, we need to establish three properties: *consistency, termination* and *liveness.*

- consistency means that each transition is *correct*, in the sense that :
  - if the start situation holds before executing the transition,
  - then the end situation holds when the transition finishes
- termination means that execution always terminates when we start from an initial situation,
  - i.e., we eventually reach a situation where no transition is enabled
- liveness means that we only terminate in a final situation

Proving correctness

To prove that the program is correct, we need to establish three properties: *consistency, termination* and *liveness.*

- consistency means that each transition is *correct*, in the sense that :
  - if the start situation holds before executing the transition,
  - then the end situation holds when the transition finishes
- termination means that execution always terminates when we start from an initial situation,
  - i.e., we eventually reach a situation where no transition is enabled
- liveness means that we only terminate in a final situation

Proving correctness

To prove that the program is correct, we need to establish three properties: *consistency, termination* and *liveness.*

- consistency means that each transition is *correct*, in the sense that :
  - if the start situation holds before executing the transition,
  - then the end situation holds when the transition finishes
- termination means that execution always terminates when we start from an initial situation,
  - i.e., we eventually reach a situation where no transition is enabled
- liveness means that we only terminate in a final situation

Proving correctness

To prove that the program is correct, we need to establish three properties: *consistency, termination* and *liveness.*

- consistency means that each transition is *correct*, in the sense that :
  - if the start situation holds before executing the transition,
  - then the end situation holds when the transition finishes
- termination means that execution always terminates when we start from an initial situation,
  - i.e., we eventually reach a situation where no transition is enabled
- liveness means that we only terminate in a final situation

Proving correctness

To prove that the program is correct, we need to establish three properties: *consistency, termination* and *liveness.*

- consistency means that each transition is *correct*, in the sense that :
  - if the start situation holds before executing the transition,
  - then the end situation holds when the transition finishes
- termination means that execution always terminates when we start from an initial situation,
  - i.e., we eventually reach a situation where no transition is enabled
- liveness means that we only terminate in a final situation

Proving correctness

To prove that the program is correct, we need to establish three properties: *consistency, termination* and *liveness.*

- consistency means that each transition is *correct*, in the sense that :
  - if the start situation holds before executing the transition,
  - then the end situation holds when the transition finishes
- termination means that execution always terminates when we start from an initial situation,
  - i.e., we eventually reach a situation where no transition is enabled
- liveness means that we only terminate in a final situation

Outline

1. Programming as mathematics

2. Mathematics of programming

   - Situations
   - Programs
   - Correctness
   - Invariant diagrams
   - Consistency
   - Termination and liveness

3. Invariant based programming

4. Case study

Check consistency of sum program

Need to prove that the following transitions are correct:

- Initialization transition
- Finalization transition, and
- Loop transition

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y);$	—>	$x < y$
$x := x + y;$		$x' = x + y$
$y := y + 1;$		$y' = y + 1$
$(x \leq y + 10);$		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
    - x, x', x'' are the successive values assigned to x, and
    - $y, y'$are the successive values assigned to y.

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y)$;	—>	$x < y$
$x := x + y$;		$x' = x + y$
$y := y + 1$;		$y' = y + 1$
$(x \leq y + 10)$;		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
  - x, x', x'' are the successive values assigned to x, and
  - $y, y'$are the successive values assigned to y.

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y);$	—>	$x < y$
$x := x + y;$		$x' = x + y$
$y := y + 1;$		$y' = y + 1$
$(x \leq y + 10);$		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
    - x, x', x'' are the successive values assigned to x, and
    - $y, y'$are the successive values assigned to y.

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y)$;	—>	$x < y$
$x := x + y$;		$x' = x + y$
$y := y + 1$;		$y' = y + 1$
$(x \leq y + 10)$;		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
  - x, x', x'' are the successive values assigned to x, and
  - $y, y'$are the successive values assigned to y.

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y)$;	—>	$x < y$
$x := x + y$;		$x' = x + y$
$y := y + 1$;		$y' = y + 1$
$(x \leq y + 10)$;		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
  - x, x', x'' are the successive values assigned to x, and
  - $y, y'$are the successive values assigned to y.

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y)$;	—>	$x < y$
$x := x + y$;		$x' = x + y$
$y := y + 1$;		$y' = y + 1$
$(x \leq y + 10)$;		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
  - x, x', x'' are the successive values assigned to x, and
  - y, y' are the successive values assigned to y.

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y);$	—>	$x < y$
$x := x + y;$		$x' = x + y$
$y := y + 1;$		$y' = y + 1$
$(x \leq y + 10);$		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
  - x, x', x'' are the successive values assigned to x, and
  - $y, y'$are the successive values assigned to y.

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y);$	—>	$x < y$
$x := x + y;$		$x' = x + y$
$y := y + 1;$		$y' = y + 1$
$(x \leq y + 10);$		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
  - x, x', x'' are the successive values assigned to x, and
  - $y, y'$are the successive values assigned to y.

Adapting transitions

We rewrite the transitions to explicitly indicate the new values computed when executing the transition, by introducing new (dashed) variables for all new values computed:

statement		constraint
$(x < y);$	—>	$x < y$
$x := x + y;$		$x' = x + y$
$y := y + 1;$		$y' = y + 1$
$(x \leq y + 10);$		$x' \leq y' + 10$
$x := x - y$		$x'' = x' - y'$

- Here
  - x, x', x'' are the successive values assigned to x, and
  - $y, y'$are the successive values assigned to y.

Correctness of transition

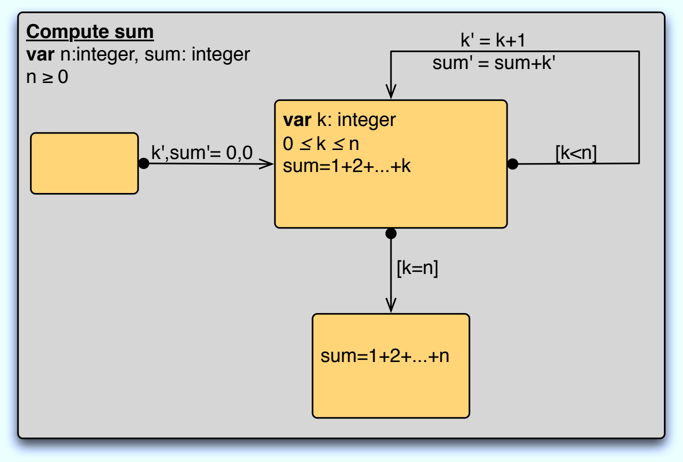
We prove that a transition S from situation P to situation Q is correct, by showing the following:

- assuming that the constraints of P hold for the program variables
- and that the new values of the program variables at the end of the transition are computed according to statement S,
- then all constraints in situation Q hold for the new values of the program variables

Summation program

Sum program with transitions described using dashed variables.

We also omit situation names here (not needed for mathematical analysis)



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

k' = k+1

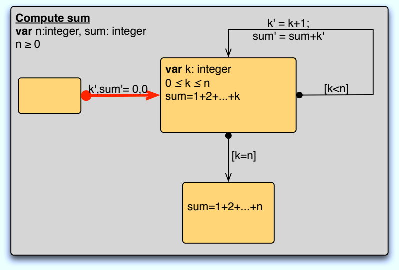
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Initialization transition



Assume

- $n, sum : integer,\ n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$\bullet \quad k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv \quad$ {substitute values $k' = 0$ and $sum' = 0$}

$0 : integer \wedge$

$0 \leq 0 \leq n \wedge$

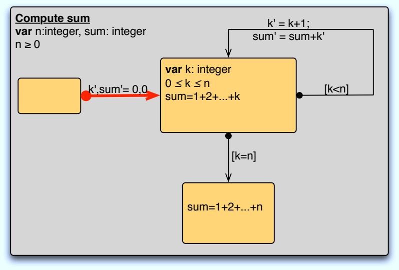
$0 = 1 + 2 + \ldots + 0 \wedge$

$0 : integer$

$\Leftarrow \quad$ {assumption $n \geq 0$}

T

Initialization transition



Assume

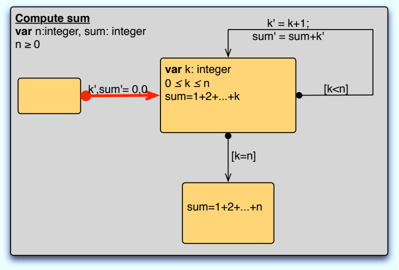
- $n, sum : integer,\ n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$$
\begin{aligned}
\bullet \quad & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv \quad & \{\text{substitute values } k' = 0 \\
& \text{and } sum' = 0\} \\
& 0 : integer \wedge \\
& 0 \leq 0 \leq n \wedge \\
& 0 = 1 + 2 + \ldots + 0 \wedge \\
& 0 : integer \\
\Leftarrow \quad & \{\text{assumption } n \geq 0\} \\
& T
\end{aligned}
$$

Initialization transition



Assume

- $n, sum : integer, \ n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

- $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv$ {substitute values $k' = 0$ and $sum' = 0$}

$0 : integer \wedge$

$0 \leq 0 \leq n \wedge$

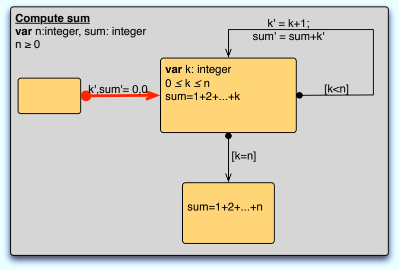
$0 = 1 + 2 + \ldots + 0 \wedge$

$0 : integer$

$\Leftarrow$ {assumption $n \geq 0$}

T

Initialization transition



Assume

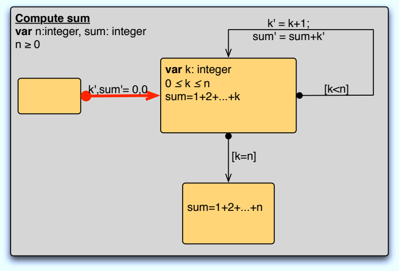
- $n, sum : integer, \; n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv & \{\text{substitute values } k' = 0 \text{ and } sum' = 0\} \\
& 0 : integer \wedge \\
& 0 \leq 0 \leq n \wedge \\
& 0 = 1 + 2 + \ldots + 0 \wedge \\
& 0 : integer \\
\Leftarrow & \{\text{assumption } n \geq 0\} \\
& T
\end{array}
$$

Initialization transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer,\ n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

- $k' : integer \wedge$

  $0 \leq k' \leq n \wedge$

  $sum' = 1 + 2 + \ldots + k' \wedge$

  $sum' : integer$

$\equiv$ {substitute values $k' = 0$ and $sum' = 0$}

$0 : integer \wedge$

$0 \leq 0 \leq n \wedge$

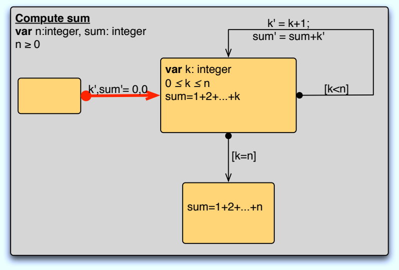
$0 = 1 + 2 + \ldots + 0 \wedge$

$0 : integer$

$\Leftarrow$ {assumption $n \geq 0$}

T

Initialization transition



Assume

- $n, sum : integer, \; n \ge 0$

Transition

- $k' = 0 \wedge sum' = 0$

- $k' : integer \wedge$

$0 \le k' \le n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv$ {substitute values $k' = 0$ and $sum' = 0$}

$0 : integer \wedge$

$0 \le 0 \le n \wedge$

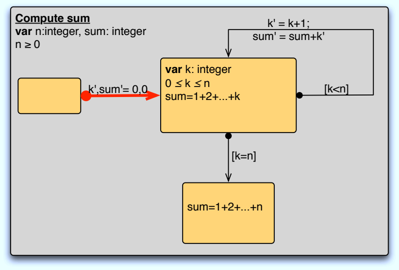
$0 = 1 + 2 + \ldots + 0 \wedge$

$0 : integer$

$\Leftarrow$ {assumption $n \ge 0$}

T

Initialization transition



Assume

- $n, sum : integer, \; n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

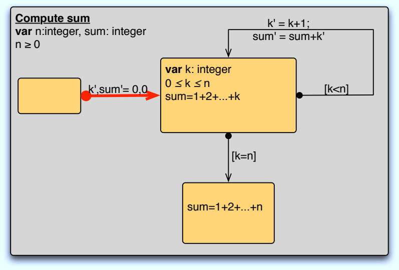
$\equiv$ {substitute values $k' = 0$ and $sum' = 0$}

$0 : integer \wedge$
$0 \leq 0 \leq n \wedge$
$0 = 1 + 2 + \ldots + 0 \wedge$
$0 : integer$

$\Leftarrow$ {assumption $n \geq 0$}

T

Initialization transition



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

k' = k+1;

sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer, \; n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

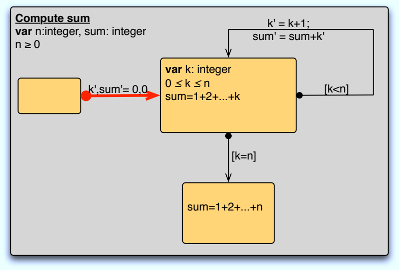
$\equiv$ {substitute values $k' = 0$ and $sum' = 0$}

$0 : integer \wedge$
$0 \leq 0 \leq n \wedge$
$0 = 1 + 2 + \ldots + 0 \wedge$
$0 : integer$

$\Leftarrow$ {assumption $n \geq 0$}

T

Initialization transition



Assume

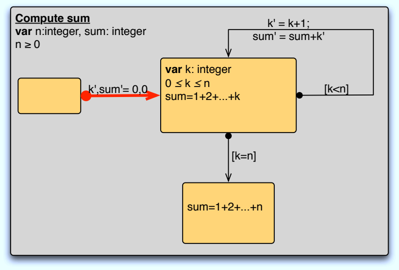
- $n, sum : integer, \; n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$$
\begin{aligned}
\bullet\quad & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv\quad & \{\text{substitute values } k' = 0 \\
& \text{and } sum' = 0\} \\
& 0 : integer \wedge \\
& 0 \leq 0 \leq n \wedge \\
& 0 = 1 + 2 + \ldots + 0 \wedge \\
& 0 : integer \\
\Leftarrow\quad & \{\text{assumption } n \geq 0\} \\
& T
\end{aligned}
$$

Initialization transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

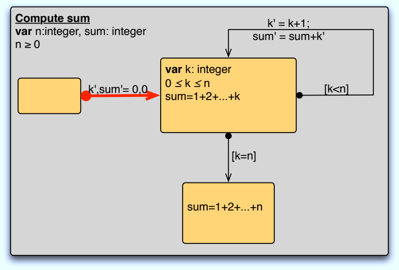
- $n, sum : integer,\ n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
 & 0 \leq k' \leq n \wedge \\
 & sum' = 1 + 2 + \ldots + k' \wedge \\
 & sum' : integer \\
\equiv & \{\text{substitute values } k' = 0 \\
 & \text{and } sum' = 0\} \\
 & 0 : integer \wedge \\
 & 0 \leq 0 \leq n \wedge \\
 & 0 = 1 + 2 + \ldots + 0 \wedge \\
 & 0 : integer \\
\Leftarrow & \{\text{assumption } n \geq 0\} \\
 & T
\end{array}
$$

Initialization transition



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

k' = k+1;

sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

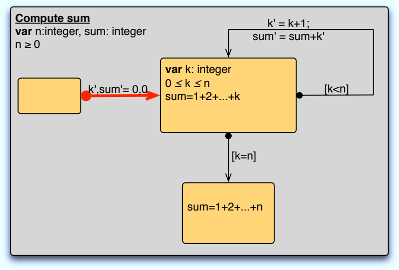
- $n, sum : integer,\ n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$$
\begin{aligned}
\bullet\quad & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv\quad & \{\text{substitute values } k' = 0 \\
& \text{and } sum' = 0\} \\
& 0 : integer \wedge \\
& 0 \leq 0 \leq n \wedge \\
& 0 = 1 + 2 + \ldots + 0 \wedge \\
& 0 : integer \\
\Leftarrow\quad & \{\text{assumption } n \geq 0\} \\
& T
\end{aligned}
$$

Initialization transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

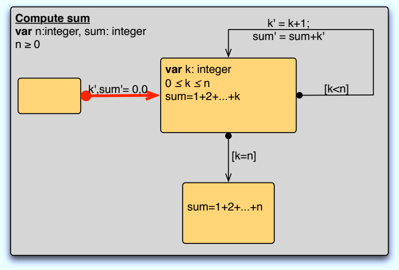
- $n, sum : integer,\ n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$$
\begin{aligned}
\bullet \quad & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv \quad & \{\text{substitute values } k' = 0 \\
& \text{and } sum' = 0\} \\
& 0 : integer \wedge \\
& 0 \leq 0 \leq n \wedge \\
& 0 = 1 + 2 + \ldots + 0 \wedge \\
& 0 : integer \\
\Leftarrow \quad & \{\text{assumption } n \geq 0\} \\
& T
\end{aligned}
$$

Initialization transition



Compute sum

var n:integer, sum: integer

n ≥ 0

k' = k+1;
sum' = sum+k'

[k<n]

k',sum'= 0,0

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer,\ n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$\bullet \quad k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv \quad$ {substitute values $k' = 0$ and $sum' = 0$}

$0 : integer \wedge$

$0 \leq 0 \leq n \wedge$

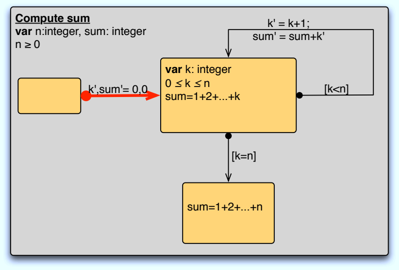
$0 = 1 + 2 + \ldots + 0 \wedge$

$0 : integer$

$\Leftarrow \quad$ {assumption $n \geq 0$}

T

Initialization transition



Assume

- $n, sum : integer, \; n \geq 0$

Transition

- $k' = 0 \wedge sum' = 0$

$\bullet$ $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv$ {substitute values $k' = 0$ and $sum' = 0$}

$0 : integer \wedge$

$0 \leq 0 \leq n \wedge$

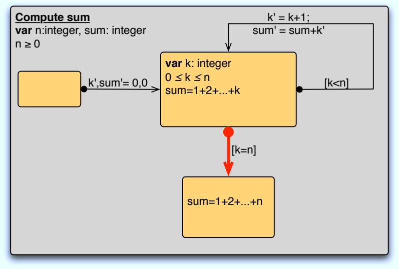
$0 = 1 + 2 + \ldots + 0 \wedge$

$0 : integer$

$\Leftarrow$ {assumption $n \geq 0$}

T

Finalization transition



Assume

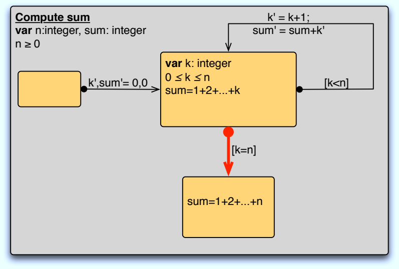
- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

$$\begin{aligned} \bullet \quad & sum = 1 + 2 + \ldots + n \\ \equiv \quad & \{\text{guard } k = n\} \\ & sum = 1 + 2 + \ldots + k \\ \Leftarrow \quad & \{\text{assumption}\} \\ & T \end{aligned}$$

Finalization transition



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

k' = k+1;

sum' = sum+k'

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer, \ n \geq 0$
- $k : integer, \ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

- $sum = 1 + 2 + \ldots + n$

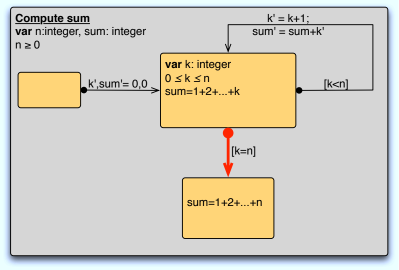
$\equiv$ $\{\text{guard } k = n\}$

$sum = 1 + 2 + \ldots + k$

$\Leftarrow$ $\{\text{assumption}\}$

T

Finalization transition



- $sum = 1 + 2 + \ldots + n$

$\equiv$ $\{\text{guard } k = n\}$

$sum = 1 + 2 + \ldots + k$

$\Leftarrow$ $\{\text{assumption}\}$

T

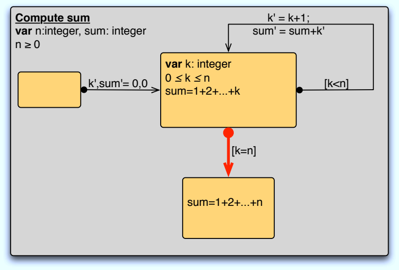
Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

Finalization transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

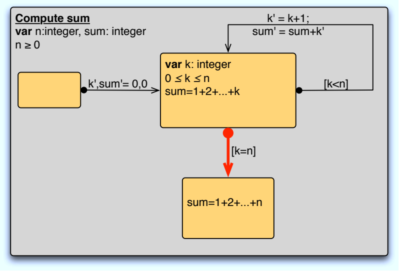
- $n, sum : integer, \ n \geq 0$
- $k : integer, \ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

$$
\begin{array}{ll}
\bullet & sum = 1 + 2 + \ldots + n \\
\equiv & \{\text{guard } k = n\} \\
 & sum = 1 + 2 + \ldots + k \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Finalization transition



Assume

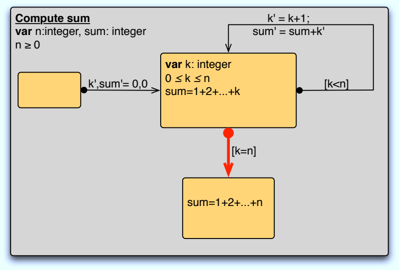
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

$$
\begin{aligned}
\bullet\quad & sum = 1 + 2 + \ldots + n \\
\equiv\quad & \{\text{guard } k = n\} \\
& sum = 1 + 2 + \ldots + k \\
\Leftarrow\quad & \{\text{assumption}\} \\
& T
\end{aligned}
$$

Finalization transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

- $sum = 1 + 2 + \ldots + n$

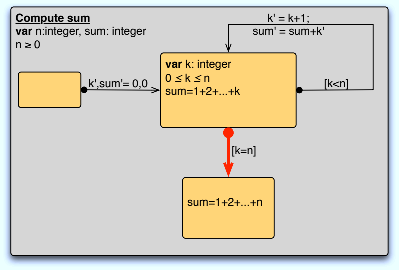
$\equiv$ $\{\text{guard } k = n\}$

$sum = 1 + 2 + \ldots + k$

$\Leftarrow$ $\{\text{assumption}\}$

T

Finalization transition



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

k' = k+1;
sum' = sum+k'

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

[k<n]

[k=n]

sum=1+2+...+n

Assume

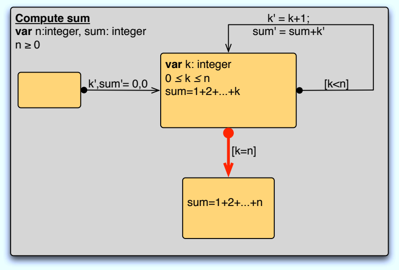
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

$$
\begin{array}{ll}
\bullet & sum = 1 + 2 + \ldots + n \\
\equiv & \{\text{guard } k = n\} \\
 & sum = 1 + 2 + \ldots + k \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Finalization transition



Assume

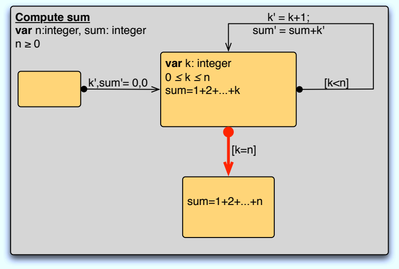
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

$$
\begin{array}{ll}
\bullet & sum = 1 + 2 + \ldots + n \\
\equiv & \{\text{guard } k = n\} \\
 & sum = 1 + 2 + \ldots + k \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Finalization transition



Assume

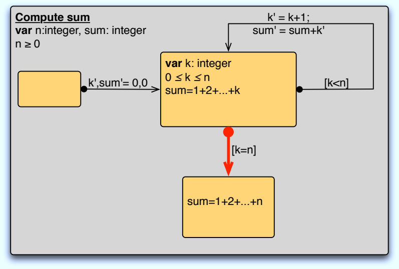
- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

$$
\begin{array}{ll}
\bullet & sum = 1 + 2 + \ldots + n \\
\equiv & \{\text{guard } k = n\} \\
 & sum = 1 + 2 + \ldots + k \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Finalization transition



Assume

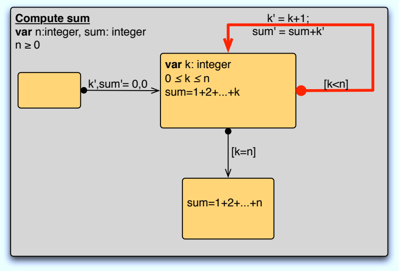
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k = n$

$$
\begin{array}{ll}
\bullet & sum = 1 + 2 + \ldots + n \\
\equiv & \{\text{guard } k = n\} \\
 & sum = 1 + 2 + \ldots + k \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$\bullet$ $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k + 1 : integer$

$\equiv$ {assumptions}

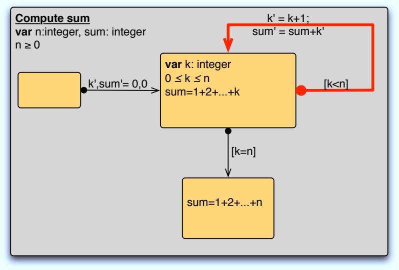
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

T

Loop transition



Assume

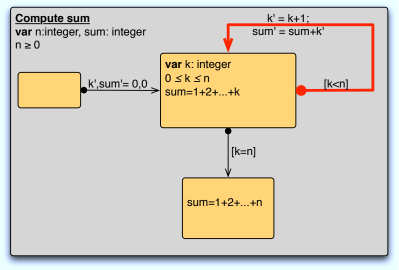
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
 & 0 \leq k' \leq n \wedge \\
 & sum' = 1 + 2 + \ldots + k' \wedge \\
 & sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
 & sum' = sum + k' = \\
 & sum + k + 1\} \\
 & k + 1 : integer \wedge \\
 & 0 \leq k + 1 \leq n \wedge \\
 & sum + k + 1 = \\
 & 1 + 2 + \ldots + (k + 1) \wedge \\
 & sum + k + 1 : integer \\
\equiv & \{\text{assumptions}\} \\
 & k + 1 \leq n \wedge \\
 & sum = 1 + 2 + \ldots + k \\
\equiv & \{\text{guard and assumption}\} \\
 & T
\end{array}
$$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$\bullet$ $k' : integer \wedge$
$0 \leq k' \leq n \wedge$
$sum' = 1 + 2 + \ldots + k' \wedge$
$sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k + 1 : integer$

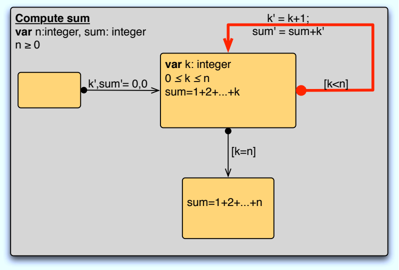
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

T

Loop transition



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

k' = k+1;

sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \land$

  $0 \leq k' \leq n \land$

  $sum' = 1 + 2 + \ldots + k' \land$

  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \land$

$0 \leq k + 1 \leq n \land$

$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \land$

$sum + k + 1 : integer$

$\equiv$ {assumptions}

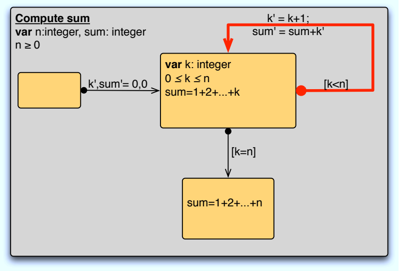
$k + 1 \leq n \land$

$sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

T

Loop transition



Assume

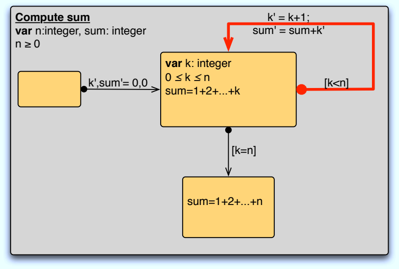
- $n, sum : integer, \; n \geq 0$
- $k : integer, \, 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$\begin{array}{ll}
\bullet & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
& sum' = sum + k' = \\
& sum + k + 1\} \\
& k + 1 : integer \wedge \\
& 0 \leq k + 1 \leq n \wedge \\
& sum + k + 1 = \\
& 1 + 2 + \ldots + (k + 1) \wedge \\
& sum + k + 1 : integer \\
\equiv & \{\text{assumptions}\} \\
& k + 1 \leq n \wedge \\
& sum = 1 + 2 + \ldots + k \\
\equiv & \{\text{guard and assumption}\} \\
& T
\end{array}$$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

≡ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k + 1 : integer$

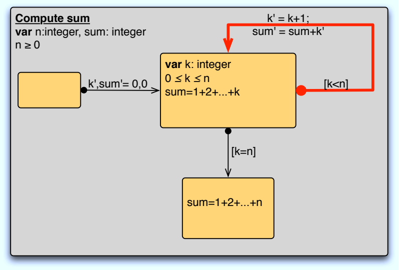
≡ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k$

≡ {guard and assumption}

T

Loop transition



Assume

- $n, sum : integer, \ n \geq 0$
- $k : integer, 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k + 1 : integer$

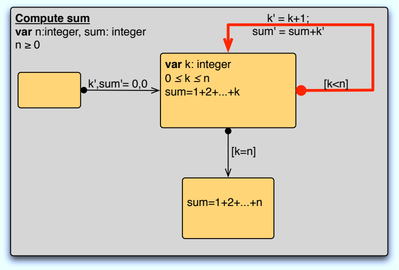
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

T

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer, \ n \geq 0$
- $k : integer, 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$

  $0 \leq k' \leq n \wedge$

  $sum' = 1 + 2 + \ldots + k' \wedge$

  $sum' : integer$

$\equiv$ $\{$substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1\}$

  $k + 1 : integer \wedge$

  $0 \leq k + 1 \leq n \wedge$

  $sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$

  $sum + k + 1 : integer$

$\equiv$ $\{$assumptions$\}$

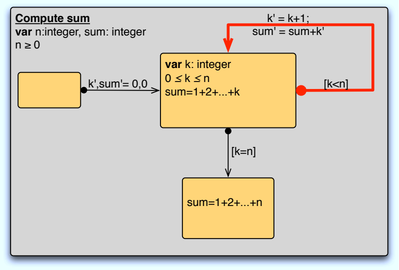
  $k + 1 \leq n \wedge$

  $sum = 1 + 2 + \ldots + k$

$\equiv$ $\{$guard and assumption$\}$

  T

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer, 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$

  $0 \leq k' \leq n \wedge$

  $sum' = 1 + 2 + \ldots + k' \wedge$

  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

  $k + 1 : integer \wedge$

  $0 \leq k + 1 \leq n \wedge$

  $sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$

  $sum + k + 1 : integer$

$\equiv$ {assumptions}

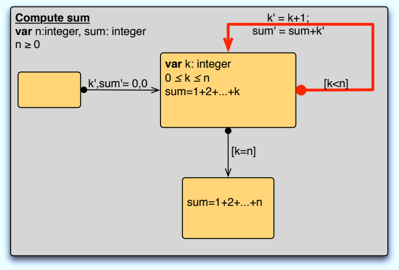
  $k + 1 \leq n \wedge$

  $sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

  T

Loop transition



Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k + 1 : integer$

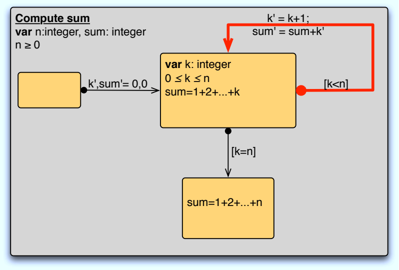
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

T

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer, 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k + 1 : integer$

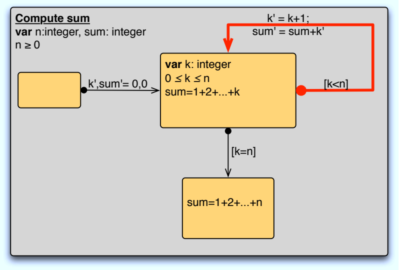
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

T

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k + 1 : integer$

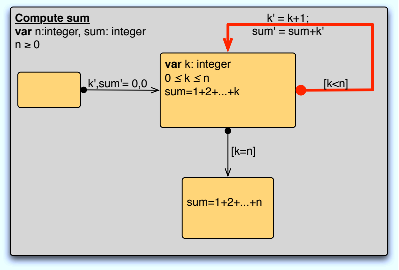
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

T

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

≡ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k + 1 : integer$

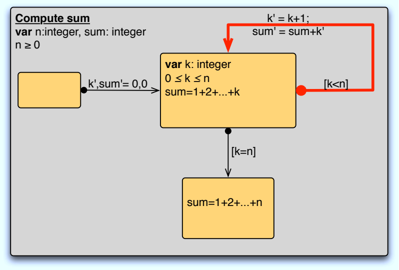
≡ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k$

≡ {guard and assumption}

T

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

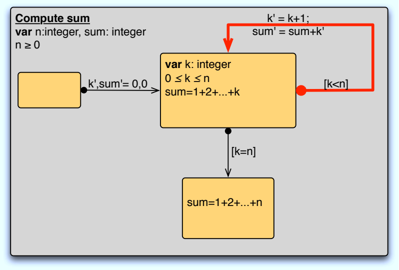
- $n, sum : integer,\ n \geq 0$
- $k : integer, 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
& sum' = sum + k' = \\
& sum + k + 1\} \\
& k + 1 : integer \wedge \\
& 0 \leq k + 1 \leq n \wedge \\
& sum + k + 1 = \\
& 1 + 2 + \ldots + (k + 1) \wedge \\
& sum + k + 1 : integer \\
\equiv & \{\text{assumptions}\} \\
& k + 1 \leq n \wedge \\
& sum = 1 + 2 + \ldots + k \\
\equiv & \{\text{guard and assumption}\} \\
& T
\end{array}
$$

Loop transition



Assume

- $n, sum : integer, \ n \geq 0$
- $k : integer, \ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

● $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

≡ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k + 1 : integer$

≡ {assumptions}

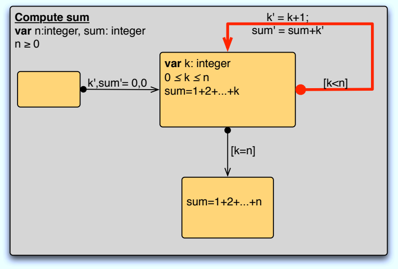
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k$

≡ {guard and assumption}

T

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

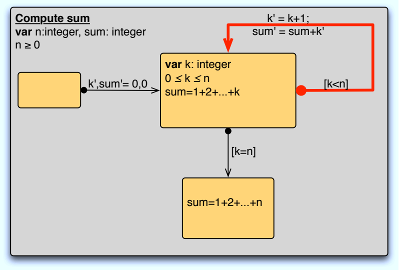
- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
& sum' = sum + k' = \\
& sum + k + 1\} \\
& k + 1 : integer \wedge \\
& 0 \leq k + 1 \leq n \wedge \\
& sum + k + 1 = \\
& 1 + 2 + \ldots + (k + 1) \wedge \\
& sum + k + 1 : integer \\
\equiv & \{\text{assumptions}\} \\
& k + 1 \leq n \wedge \\
& sum = 1 + 2 + \ldots + k \\
\equiv & \{\text{guard and assumption}\} \\
& T
\end{array}
$$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k' = sum + k + 1$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k + 1 = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k + 1 : integer$

$\equiv$ {assumptions}

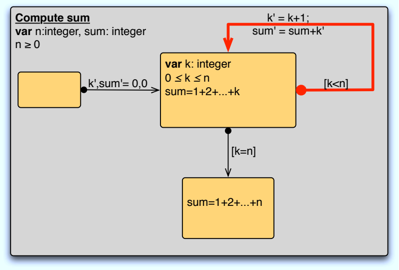
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k$

$\equiv$ {guard and assumption}

T

Loop transition



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

k' = k+1;

sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

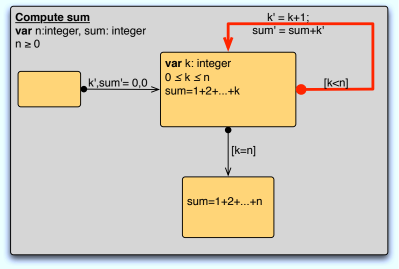
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
 & 0 \leq k' \leq n \wedge \\
 & sum' = 1 + 2 + \ldots + k' \wedge \\
 & sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
 & sum' = sum + k' = \\
 & sum + k + 1\} \\
 & k + 1 : integer \wedge \\
 & 0 \leq k + 1 \leq n \wedge \\
 & sum + k + 1 = \\
 & 1 + 2 + \ldots + (k + 1) \wedge \\
 & sum + k + 1 : integer \\
\equiv & \{\text{assumptions}\} \\
 & k + 1 \leq n \wedge \\
 & sum = 1 + 2 + \ldots + k \\
\equiv & \{\text{guard and assumption}\} \\
 & T
\end{array}
$$

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

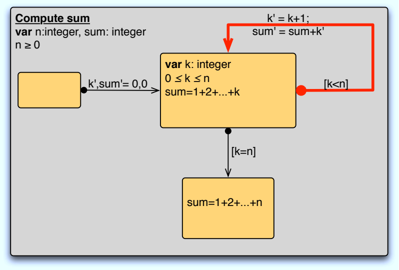
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
& sum' = sum + k' = \\
& sum + k + 1\} \\
& k + 1 : integer \wedge \\
& 0 \leq k + 1 \leq n \wedge \\
& sum + k + 1 = \\
& 1 + 2 + \ldots + (k + 1) \wedge \\
& sum + k + 1 : integer \\
\equiv & \{\text{assumptions}\} \\
& k + 1 \leq n \wedge \\
& sum = 1 + 2 + \ldots + k \\
\equiv & \{\text{guard and assumption}\} \\
& T
\end{array}
$$

Loop transition



Assume

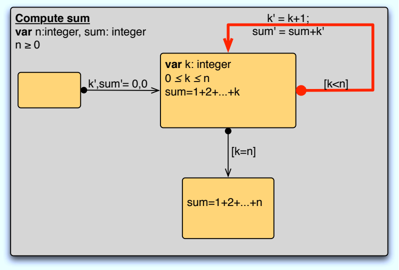
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
& sum' = sum + k' = \\
& sum + k + 1\} \\
& k + 1 : integer \wedge \\
& 0 \leq k + 1 \leq n \wedge \\
& sum + k + 1 = \\
& 1 + 2 + \ldots + (k + 1) \wedge \\
& sum + k + 1 : integer \\
\equiv & \{\text{assumptions}\} \\
& k + 1 \leq n \wedge \\
& sum = 1 + 2 + \ldots + k \\
\equiv & \{\text{guard and assumption}\} \\
& T
\end{array}
$$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
 & 0 \leq k' \leq n \wedge \\
 & sum' = 1 + 2 + \ldots + k' \wedge \\
 & sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
 & sum' = sum + k' = \\
 & sum + k + 1\} \\
 & k + 1 : integer \wedge \\
 & 0 \leq k + 1 \leq n \wedge \\
 & sum + k + 1 = \\
 & 1 + 2 + \ldots + (k + 1) \wedge \\
 & sum + k + 1 : integer \\
\equiv & \{\text{assumptions}\} \\
 & k + 1 \leq n \wedge \\
 & sum = 1 + 2 + \ldots + k \\
\equiv & \{\text{guard and assumption}\} \\
 & T
\end{array}
$$

Detecting errors

Assume that we make a small error in the loop transition:

- we write

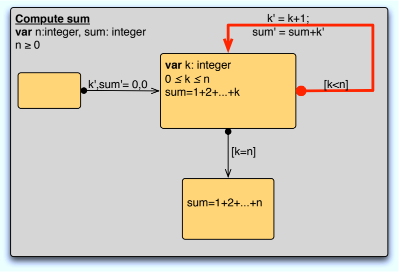
$$sum := sum + k;\ k := k + 1$$

- in stead of

$$k := k + 1;\ sum := sum + k$$

What happens with the proof.

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

$\equiv$ {assumptions}

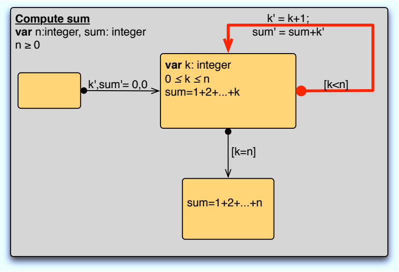
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k : integer$

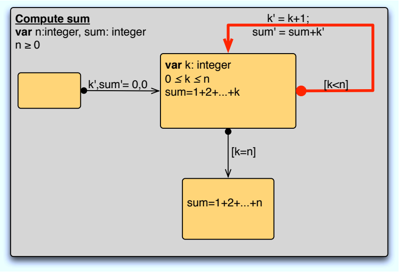
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k : integer$

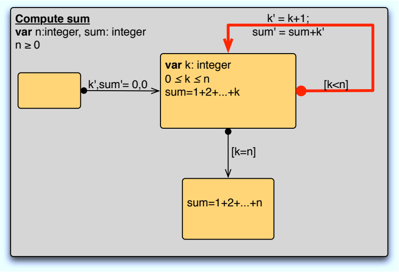
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

● $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

≡ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

≡ {assumptions}

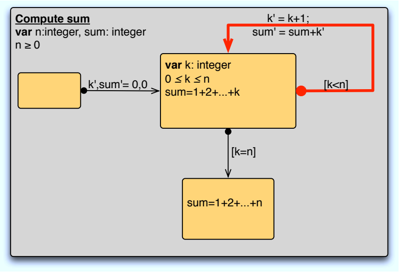
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

≡ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k : integer$

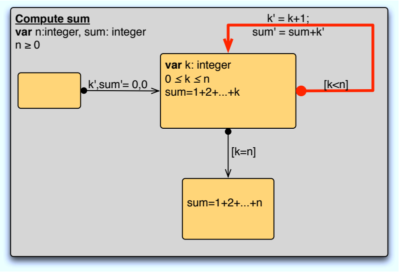
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k : integer$

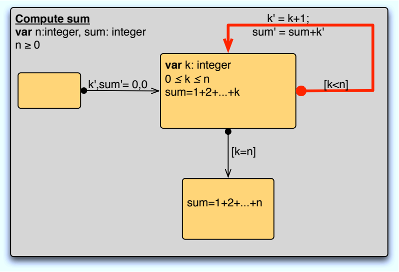
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

$\equiv$ {assumptions}

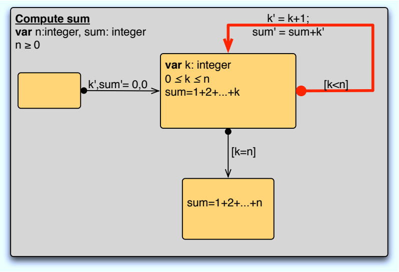
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer, \ n \geq 0$
- $k : integer, \ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$

  $0 \leq k' \leq n \wedge$

  $sum' = 1 + 2 + \ldots + k' \wedge$

  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

  $k + 1 : integer \wedge$

  $0 \leq k + 1 \leq n \wedge$

  $sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

  $sum + k : integer$

$\equiv$ {assumptions}

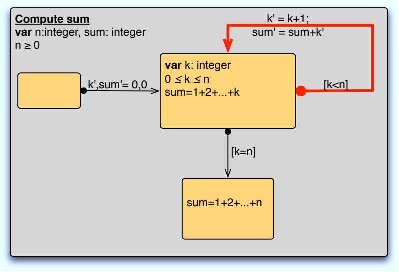
  $k + 1 \leq n \wedge$

  $sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

  $0 = 1$

Loop transition



Assume

- n, sum : *integer*, $n \geq 0$
- k : *integer*, $0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- k' : *integer* $\wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

sum' : *integer*

≡ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1$: *integer* $\wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k$: *integer*

≡ {assumptions}

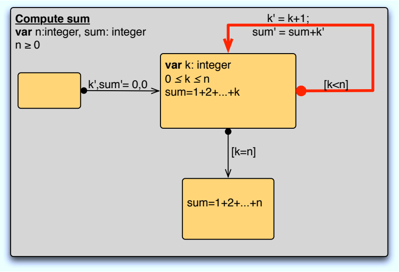
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

≡ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \ge 0$
- $k : integer,\ 0 \le k \le n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \le n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$
  $0 \le k' \le n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$
$0 \le k + 1 \le n \wedge$
$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k : integer$

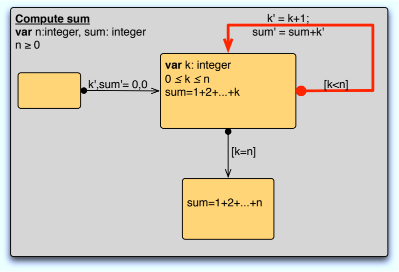
$\equiv$ {assumptions}

$k + 1 \le n \wedge$
$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ $\{$substitute $k' = k + 1$ and $sum' = sum + k\}$

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

$\equiv$ $\{$assumptions$\}$

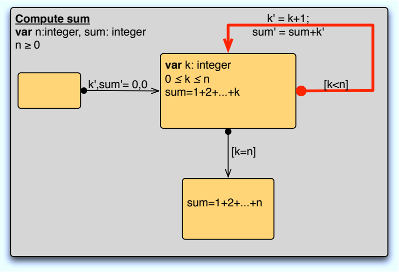
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ $\{$guard and assumption $sum = 1 + 2 + \ldots + k\}$

$0 = 1$

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

≡ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

≡ {assumptions}

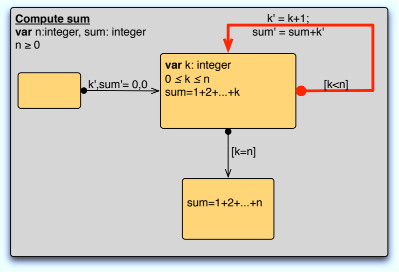
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

≡ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$
$sum + k : integer$

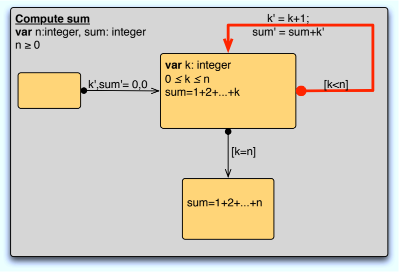
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

≡ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

≡ {assumptions}

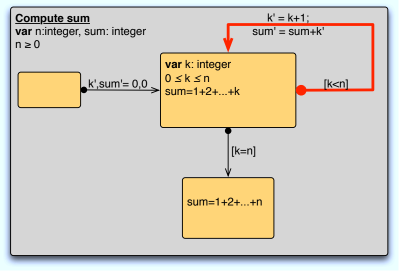
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

≡ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

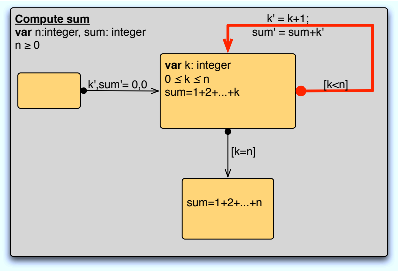
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

$$
\begin{aligned}
\bullet\quad & k' : integer \wedge \\
& 0 \leq k' \leq n \wedge \\
& sum' = 1 + 2 + \ldots + k' \wedge \\
& sum' : integer \\
\equiv\quad & \{\text{substitute } k' = k + 1 \text{ and} \\
& sum' = sum + k\} \\
& k + 1 : integer \wedge \\
& 0 \leq k + 1 \leq n \wedge \\
& sum + k = \\
& 1 + 2 + \ldots + (k + 1) \wedge \\
& sum + k : integer \\
\equiv\quad & \{\text{assumptions}\} \\
& k + 1 \leq n \wedge \\
& sum = 1 + 2 + \ldots + k + 1 \\
\equiv\quad & \{\text{guard and assumption} \\
& sum = 1 + 2 + \ldots + k\} \\
& 0 = 1
\end{aligned}
$$

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$
  $0 \leq k' \leq n \wedge$
  $sum' = 1 + 2 + \ldots + k' \wedge$
  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$
$0 \leq k + 1 \leq n \wedge$
$sum + k =$
$1 + 2 + \ldots + (k + 1) \wedge$
$sum + k : integer$

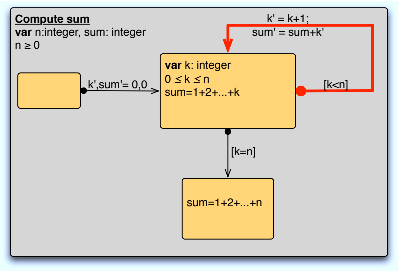
$\equiv$ {assumptions}

$k + 1 \leq n \wedge$
$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$

  $0 \leq k' \leq n \wedge$

  $sum' = 1 + 2 + \ldots + k' \wedge$

  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k =$
$1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

$\equiv$ {assumptions}

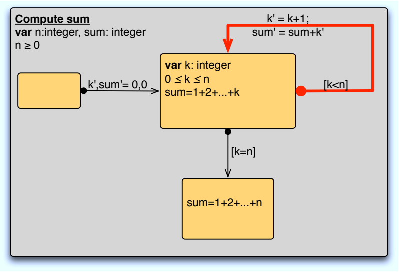
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

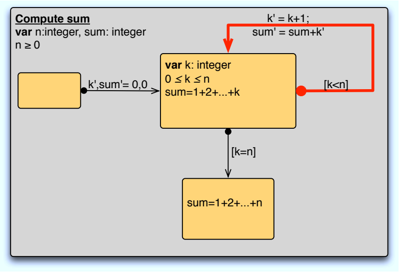
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
 & 0 \leq k' \leq n \wedge \\
 & sum' = 1 + 2 + \ldots + k' \wedge \\
 & sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
 & sum' = sum + k\} \\
 & k + 1 : integer \wedge \\
 & 0 \leq k + 1 \leq n \wedge \\
 & sum + k = \\
 & 1 + 2 + \ldots + (k + 1) \wedge \\
 & sum + k : integer \\
\equiv & \{\text{assumptions}\} \\
 & k + 1 \leq n \wedge \\
 & sum = 1 + 2 + \ldots + k + 1 \\
\equiv & \{\text{guard and assumption} \\
 & sum = 1 + 2 + \ldots + k\} \\
 & 0 = 1
\end{array}
$$

Loop transition



Assume

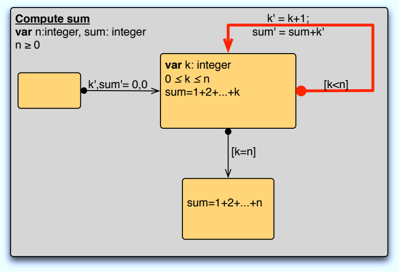
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

$$
\begin{array}{ll}
\bullet & k' : integer \wedge \\
 & 0 \leq k' \leq n \wedge \\
 & sum' = 1 + 2 + \ldots + k' \wedge \\
 & sum' : integer \\
\equiv & \{\text{substitute } k' = k + 1 \text{ and} \\
 & sum' = sum + k\} \\
 & k + 1 : integer \wedge \\
 & 0 \leq k + 1 \leq n \wedge \\
 & sum + k = \\
 & 1 + 2 + \ldots + (k + 1) \wedge \\
 & sum + k : integer \\
\equiv & \{\text{assumptions}\} \\
 & k + 1 \leq n \wedge \\
 & sum = 1 + 2 + \ldots + k + 1 \\
\equiv & \{\text{guard and assumption} \\
 & sum = 1 + 2 + \ldots + k\} \\
 & 0 = 1
\end{array}
$$

Loop transition



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1;
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

$\bullet$ $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

$\equiv$ {assumptions}

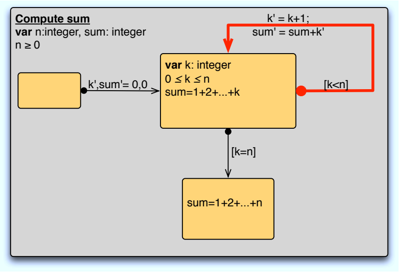
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

● $k' : integer \wedge$

$0 \leq k' \leq n \wedge$

$sum' = 1 + 2 + \ldots + k' \wedge$

$sum' : integer$

≡ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k =$
$1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

≡ {assumptions}

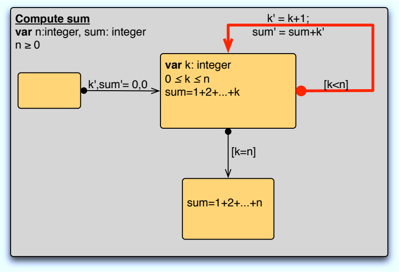
$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

≡ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Loop transition



Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $sum' = sum + k$
- $k' = k + 1$
-

- $k' : integer \wedge$

  $0 \leq k' \leq n \wedge$

  $sum' = 1 + 2 + \ldots + k' \wedge$

  $sum' : integer$

$\equiv$ {substitute $k' = k + 1$ and $sum' = sum + k$}

$k + 1 : integer \wedge$

$0 \leq k + 1 \leq n \wedge$

$sum + k = 1 + 2 + \ldots + (k + 1) \wedge$

$sum + k : integer$

$\equiv$ {assumptions}

$k + 1 \leq n \wedge$

$sum = 1 + 2 + \ldots + k + 1$

$\equiv$ {guard and assumption $sum = 1 + 2 + \ldots + k$}

$0 = 1$

Outline

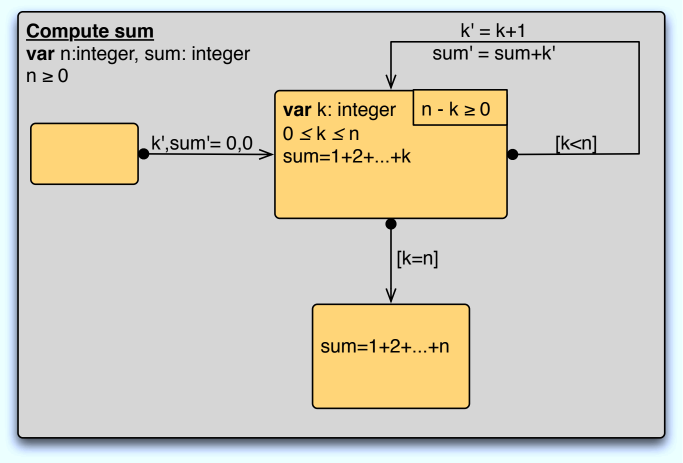
1. Programming as mathematics
2. Mathematics of programming
   - Situations
   - Programs
   - Correctness
   - Invariant diagrams
   - Consistency
   - Termination and liveness
3. Invariant based programming
4. Case study

Termination

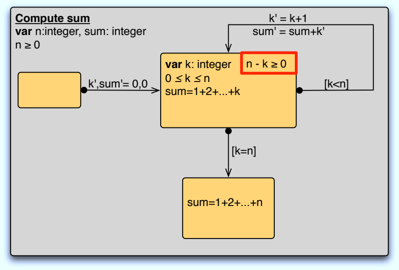
- Select intermediate situations such that each loop is cut by one of these intermediate situation
- Associate a *variant* expression e with each such intermediate situation, and check that
    - $e \geq 0$ holds in this situation,
    - the value of e has decreased whenever we re-enter this situation, and
    - the value of e is never increased in the program
- We express the termination condition by writing $e \geq 0$ in the upper right corner of the intermediate situation.

Variant for sum program

The value of $n - k$ is decreased by each iteration in the sum program, but it never becomes negative. Choose $n - k$ as the variant.



Boundedness



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

n - k ≥ 0

k' = k+1
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

- $n - k \geq 0$

$\equiv$ $\{\text{arithmetic}\}$

$n \geq k$

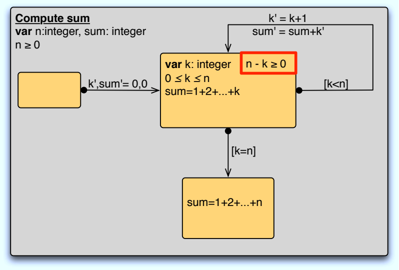
$\Leftarrow$ $\{\text{assumption}\}$

T

Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Boundedness

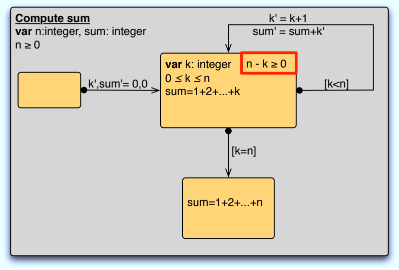


$$
\begin{array}{ll}
\bullet & n - k \geq 0 \\
\equiv & \{\text{arithmetic}\} \\
 & n \geq k \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Boundedness

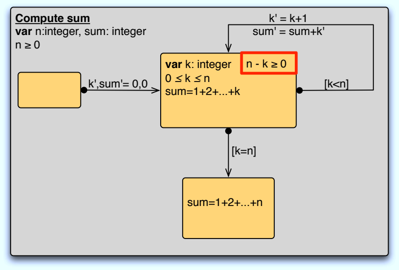


$$
\begin{array}{ll}
\bullet & n - k \geq 0 \\
\equiv & \{\text{arithmetic}\} \\
 & n \geq k \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Assume

- $n, sum : integer, \ n \geq 0$
- $k : integer, \ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Boundedness

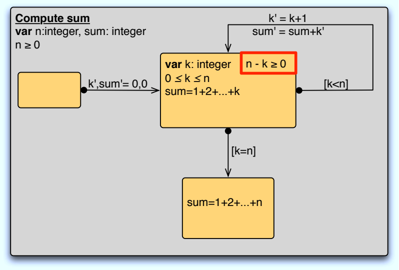


$$\begin{array}{ll} \bullet & n - k \geq 0 \\ \equiv & \{\text{arithmetic}\} \\ & n \geq k \\ \Leftarrow & \{\text{assumption}\} \\ & T \end{array}$$

Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Boundedness

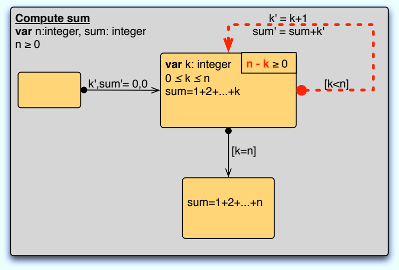


$$
\begin{array}{ll}
\bullet & n - k \geq 0 \\
\equiv & \{\text{arithmetic}\} \\
 & n \geq k \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Decrease



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

n - k ≥ 0

k' = k+1
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

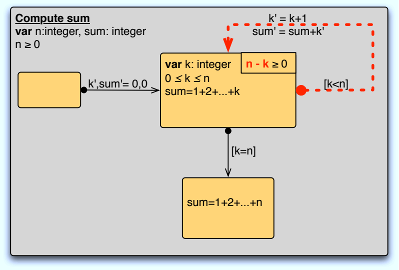
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & n - k' < n - k \\
\equiv & \{\text{substitute } k' = k + 1\} \\
 & n - (k + 1) < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & n - k - 1 < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & -1 < 0 \\
\equiv & \{\text{arithmetic}\} \\
 & T
\end{array}
$$

Decrease



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

n - k ≥ 0

k' = k+1
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

- $n - k' < n - k$

$\equiv$ {substitute $k' = k + 1$}

$n - (k + 1) < n - k$

$\equiv$ {arithmetic}

$n - k - 1 < n - k$

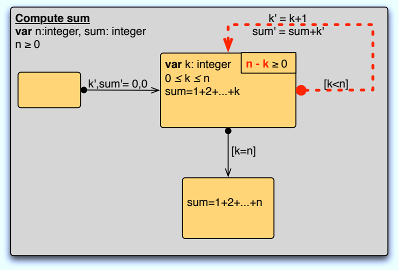
$\equiv$ {arithmetic}

$-1 < 0$

$\equiv$ {arithmetic}

T

Decrease



Assume

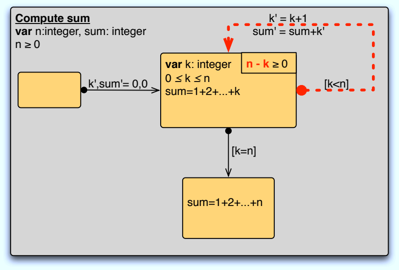
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & n - k' < n - k \\
\equiv & \{\text{substitute } k' = k + 1\} \\
 & n - (k + 1) < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & n - k - 1 < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & -1 < 0 \\
\equiv & \{\text{arithmetic}\} \\
 & T
\end{array}
$$

Decrease



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

var k: integer

n - k ≥ 0

0 ≤ k ≤ n

sum=1+2+...+k

k' = k+1

sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

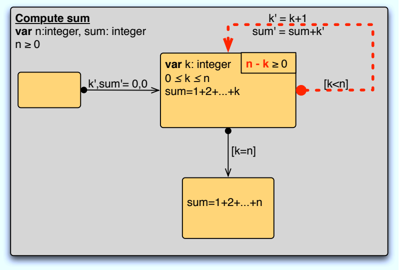
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & n - k' < n - k \\
\equiv & \{\text{substitute } k' = k + 1\} \\
 & n - (k + 1) < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & n - k - 1 < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & -1 < 0 \\
\equiv & \{\text{arithmetic}\} \\
 & T
\end{array}
$$

Decrease



Assume

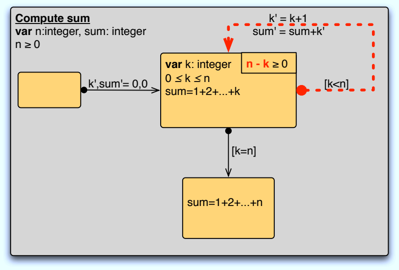
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$\begin{array}{ll}
\bullet & n - k' < n - k \\
\equiv & \{\text{substitute } k' = k + 1\} \\
 & n - (k + 1) < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & n - k - 1 < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & -1 < 0 \\
\equiv & \{\text{arithmetic}\} \\
 & T
\end{array}$$

Decrease



Assume

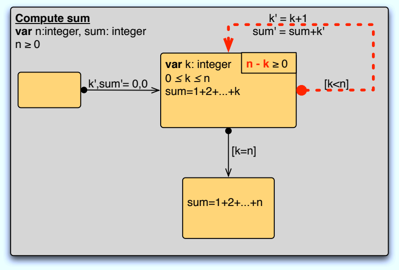
- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & n - k' < n - k \\
\equiv & \{\text{substitute } k' = k + 1\} \\
 & n - (k + 1) < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & n - k - 1 < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & -1 < 0 \\
\equiv & \{\text{arithmetic}\} \\
 & T
\end{array}
$$

Decrease



Assume

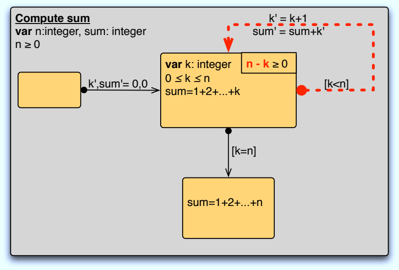
- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & n - k' < n - k \\
\equiv & \{\text{substitute } k' = k + 1\} \\
 & n - (k + 1) < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & n - k - 1 < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & -1 < 0 \\
\equiv & \{\text{arithmetic}\} \\
 & T
\end{array}
$$

Decrease



Compute sum

var n:integer, sum: integer

n ≥ 0

k',sum'= 0,0

var k: integer

0 ≤ k ≤ n

sum=1+2+...+k

n - k ≥ 0

k' = k+1

sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

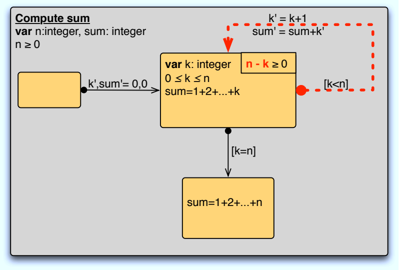
- $n, sum : integer,\ n \geq 0$
- $k : integer,\ 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & n - k' < n - k \\
\equiv & \{\text{substitute } k' = k + 1\} \\
 & n - (k + 1) < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & n - k - 1 < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & -1 < 0 \\
\equiv & \{\text{arithmetic}\} \\
 & T
\end{array}
$$

Decrease



Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Transition

- $k \leq n$
- $k' = k + 1$
- $sum' = sum + k + 1$

$$
\begin{array}{ll}
\bullet & n - k' < n - k \\
\equiv & \{\text{substitute } k' = k + 1\} \\
 & n - (k + 1) < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & n - k - 1 < n - k \\
\equiv & \{\text{arithmetic}\} \\
 & -1 < 0 \\
\equiv & \{\text{arithmetic}\} \\
 & T
\end{array}
$$

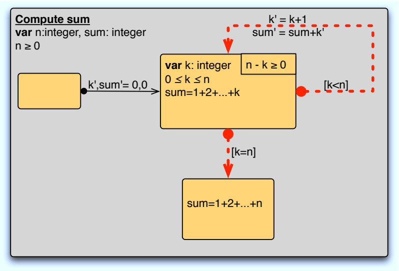
Check liveness

Finally, we need to prove liveness:

- check that execution does not get stuck in an intermediate situation

This is true, if at least one of the outgoing transitions is always enabled in an intermediate situation.

Liveness



Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

- $k < n \vee k = n$

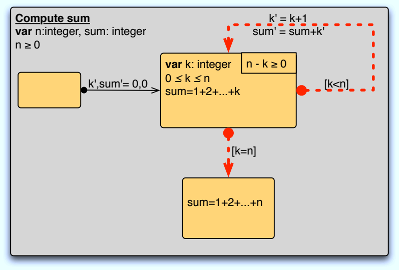
$\equiv$ {arithmetic}

$k \leq n$

$\Leftarrow$ {assumption}

T

Liveness



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
n - k ≥ 0
0 ≤ k ≤ n
sum=1+2+...+k

k' = k+1
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

- $k < n \lor k = n$

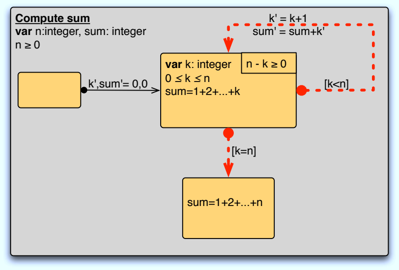
$\equiv$ {arithmetic}

$k \leq n$

$\Leftarrow$ {assumption}

T

Liveness



- $k < n \vee k = n$

$\equiv$ {arithmetic}

$k \leq n$

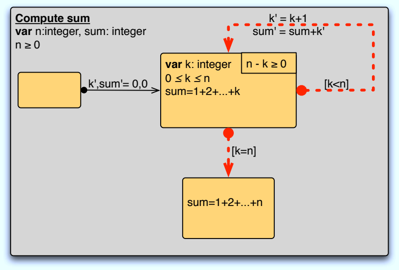
$\Leftarrow$ {assumption}

T

Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

Liveness



Assume

- $n, sum : integer, \; n \geq 0$
- $k : integer, \; 0 \leq k \leq n$
- $sum = 1 + 2 + \ldots + k$

• $k < n \vee k = n$

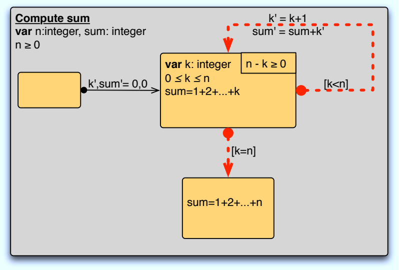
$\equiv$ {arithmetic}

$k \leq n$

$\Leftarrow$ {assumption}

T

Liveness



Compute sum
var n:integer, sum: integer
n ≥ 0

k',sum'= 0,0

var k: integer
0 ≤ k ≤ n
sum=1+2+...+k

n - k ≥ 0

k' = k+1
sum' = sum+k'

[k<n]

[k=n]

sum=1+2+...+n

$$
\begin{array}{ll}
\bullet & k < n \vee k = n \\
\equiv & \{\text{arithmetic}\} \\
 & k \le n \\
\Leftarrow & \{\text{assumption}\} \\
 & T
\end{array}
$$

Assume

- $n, sum : integer, \; n \ge 0$
- $k : integer, \; 0 \le k \le n$
- $sum = 1 + 2 + \ldots + k$

Opportunity for SAT/SMT solvers

- The number of things that must be checked is quite large
- But most of the properties checked are rather trivial
- Would like an automatic way of discharging most of the simple proof obligations
- Show only to the programmer
    - those properties that are false, and
    - those properties that could not be proved
- These are most likely indications of some errors in the program
    - either some situation is wrongly or incompletely described
    - or some transition or termination function is wrong
    - or some theory is wrong or incomplete

Constructing correct programs: alternative approaches

- **A posteriori correctness proof** (Floyd, Naur, Hoare, ...). Prove correctness after program has been written and debugged.
- **Constructive proofs** (Dijkstra, ...). Construct the program and its proof hand in hand, to satisfy given pre- and postconditions.
- **Invariant based programming** (Reynolds, van Emden, Back, ...). Formulate the program invariants first, then construct code that maintains these invariants. (Hehner has similar idea, but starts from relations rather than predicates)

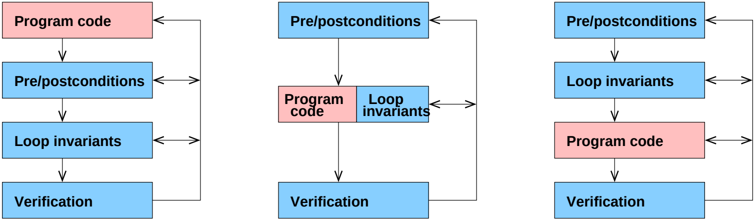
Constructing correct programs: alternative approaches

- **A posteriori correctness proof** (Floyd, Naur, Hoare, ...). Prove correctness after program has been written and debugged.
- **Constructive proofs** (Dijkstra, ...). Construct the program and its proof hand in hand, to satisfy given pre- and postconditions.
- **Invariant based programming** (Reynolds, van Emden, Back, ...). Formulate the program invariants first, then construct code that maintains these invariants. (Hehner has similar idea, but starts from relations rather than predicates)

Constructing correct programs: alternative approaches

- **A posteriori correctness proof** (Floyd, Naur, Hoare, ...). Prove correctness after program has been written and debugged.
- **Constructive proofs** (Dijkstra, ...). Construct the program and its proof hand in hand, to satisfy given pre- and postconditions.
- **Invariant based programming** (Reynolds, van Emden, Back, ...). Formulate the program invariants first, then construct code that maintains these invariants. (Hehner has similar idea, but starts from relations rather than predicates)

Place of coding in work flow



A posteriori proof

Constructive approach

Invariant based programming

Questions

- Is it feasible (and practical) to construct the program invariants (situations) before we have constructed any code
- What are the main difficulties when using invariant based programming
- Is it feasible to teach invariant based programming to novices (CS students, high school students)
- What kind of computer support can we provide for invariant based programming
- Does the approach scale up to larger programs and more complex software systems.
- Where do I find out more about the approach

Questions

- Is it feasible (and practical) to construct the program invariants (situations) before we have constructed any code
- What are the main difficulties when using invariant based programming
- Is it feasible to teach invariant based programming to novices (CS students, high school students)
- What kind of computer support can we provide for invariant based programming
- Does the approach scale up to larger programs and more complex software systems.
- Where do I find out more about the approach

Questions

- Is it feasible (and practical) to construct the program invariants (situations) before we have constructed any code
- What are the main difficulties when using invariant based programming
- Is it feasible to teach invariant based programming to novices (CS students, high school students)
- What kind of computer support can we provide for invariant based programming
- Does the approach scale up to larger programs and more complex software systems.
- Where do I find out more about the approach

Questions

- Is it feasible (and practical) to construct the program invariants (situations) before we have constructed any code
- What are the main difficulties when using invariant based programming
- Is it feasible to teach invariant based programming to novices (CS students, high school students)
- What kind of computer support can we provide for invariant based programming
- Does the approach scale up to larger programs and more complex software systems.
- Where do I find out more about the approach

Questions

- Is it feasible (and practical) to construct the program invariants (situations) before we have constructed any code
- What are the main difficulties when using invariant based programming
- Is it feasible to teach invariant based programming to novices (CS students, high school students)
- What kind of computer support can we provide for invariant based programming
- Does the approach scale up to larger programs and more complex software systems.
- Where do I find out more about the approach

Questions

- Is it feasible (and practical) to construct the program invariants (situations) before we have constructed any code
- What are the main difficulties when using invariant based programming
- Is it feasible to teach invariant based programming to novices (CS students, high school students)
- What kind of computer support can we provide for invariant based programming
- Does the approach scale up to larger programs and more complex software systems.
- Where do I find out more about the approach

From algorithm to correct program

- Invariant based programming starts from a rough idea of how the algorithm is intended to work
- The basic work flow of invariant based programming is intended to turn this algorithmic idea into
  - an executable program
  - that has been mathematically proved correct.
- The level of rigour in the mathematical proof can vary
  - from rough hand checked transitions,
  - through rigorous mathematical proofs(e.g., using structured derivations),
  - to completely machine checked proofs.

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
    - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
    - Check that each transition is correct at the same time
5. Then check that
    - the program terminates
    - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
    - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
    - Check that each transition is correct at the same time
5. Then check that
    - the program terminates
    - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
    - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
    - Check that each transition is correct at the same time
5. Then check that
    - the program terminates
    - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
   - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
   - Check that each transition is correct at the same time
5. Then check that
   - the program terminates
   - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
   - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
   - Check that each transition is correct at the same time
5. Then check that
   - the program terminates
   - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
    - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
    - Check that each transition is correct at the same time
5. Then check that
    - the program terminates
    - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
   - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
   - Check that each transition is correct at the same time
5. Then check that
   - the program terminates
   - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
    - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
    - Check that each transition is correct at the same time
5. Then check that
    - the program terminates
    - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
    - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
    - Check that each transition is correct at the same time
5. Then check that
    - the program terminates
    - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

Basic work flow

1. Draw figures that illustrates the basic data structures
2. Identify the basic situations in the algorithm
3. Formalize the constraints of each situation in some logical language
    - extend the underlying theory with new definitions and concepts as needed
4. Connect situations with transitions
    - Check that each transition is correct at the same time
5. Then check that
    - the program terminates
    - and that the program is live
6. Adjust situations and transitions whenever there is a problem in the proof, and recheck

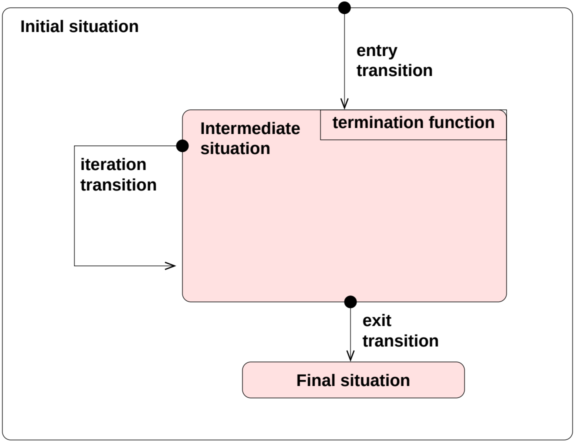
Experiments with invariant based programs

- We have been trying this approach in a number of small sessions
- Usually two persons constructing an invariant based program together
    - IFIP WG 2.3 members
    - programmers without a priori knowledge of formal methods
    - undergraduate, graduate and Ph.D. students
- Session usually takes 2.5 - 3 hours. Some 15 sessions done this far
- Programming problem a standard small one:
    - sorting, searching, Dutch national flag, computing some property of a tree, etc. Usually doable with one or two nested loops.

Experiences

- The approach works well, for IFIP WG2.3 members as well as for novices to program verification
- Finding initial invariants is quite easy when one starts from a figure
- Invariant is improved when transitions are introduced one by one
- Some very subtle bugs are found in transitions/invariants during verification
- Tool support for automatic checking highly desirable (but we can live without it for small programs)

Stages in work flow for single loop program



- draw figure
- extend theory
- formulate in logic
- extend diagram
- check correctness

Difficulty (subjective assessment)

	figures	extend theory	formulate	extend diagram	check
initial situation	X	X	X	X	
final situation	XX	XXX	XX	X	
intermediate sit	X	XXX	XX	X	
entry transition			X	X	X
exit transition			X	X	X
iteration trans			XX	X	XX
termination		X	X	X	X
liveness			X		X
difficulty	medium	can be hard	medium/easy	easy	medium

CS curriculum at Åbo Akademi

- First year students:
  - Structured derivations (logic course based on structured derivations)
  - Introduction to programming (based on Python language)
  - Mathematics of programming (invariant based programming)
- These courses have been taught now for 3 - 4 years
  - experiences are good
  - students master these courses
  - they appreciate the added understanding that it brings to mathematics and programming
- First two courses have been taught in high school also, but third (invariant based programming) has only been taught at university level

Computer support: the Socos environment

- The Socos environment supports invariant based programming
- Provides a graphical and textual representation of invariant based programs
- Uses theorem provers to automatically discharge verification conditions (PVS, Simplify, Yices)
- Socos only shows proof obligations that have not been proved automatically.
- Environment compiles invariant based program directly to Python; executes them, has a debugging mode
- Can also check procedure pre- and postconditions and invariants during run time
- New version, Socos 2, is being finalized.

Lessons learnt

- Novices have difficulties with formulating invariants. Teaching logic (structured derivations) to programmers should take care of this difficulty
- Proving correctness of transitions is important and revealing, but it is tedious, both for experts and novices. Providing mechanized tool support for proving verification conditions is crucial for scaling up the approach.
- Formulating iteration transition is error prone. Verifying the correctness of the transition is a very efficient way of revealing errors here.
- Some students do not know how to draw diagrams and figures anymore. Thinking is done directly in a terms of programming language constructs.

Lessons learnt

- Novices have difficulties with formulating invariants. Teaching logic (structured derivations) to programmers should take care of this difficulty
- Proving correctness of transitions is important and revealing, but it is tedious, both for experts and novices. Providing mechanized tool support for proving verification conditions is crucial for scaling up the approach.
- Formulating iteration transition is error prone. Verifying the correctness of the transition is a very efficient way of revealing errors here.
- Some students do not know how to draw diagrams and figures anymore. Thinking is done directly in a terms of programming language constructs.

Lessons learnt

- Novices have difficulties with formulating invariants. Teaching logic (structured derivations) to programmers should take care of this difficulty
- Proving correctness of transitions is important and revealing, but it is tedious, both for experts and novices. Providing mechanized tool support for proving verification conditions is crucial for scaling up the approach.
- Formulating iteration transition is error prone. Verifying the correctness of the transition is a very efficient way of revealing errors here.
- Some students do not know how to draw diagrams and figures anymore. Thinking is done directly in a terms of programming language constructs.

Lessons learnt

- Novices have difficulties with formulating invariants. Teaching logic (structured derivations) to programmers should take care of this difficulty
- Proving correctness of transitions is important and revealing, but it is tedious, both for experts and novices. Providing mechanized tool support for proving verification conditions is crucial for scaling up the approach.
- Formulating iteration transition is error prone. Verifying the correctness of the transition is a very efficient way of revealing errors here.
- Some students do not know how to draw diagrams and figures anymore. Thinking is done directly in a terms of programming language constructs.

Theory building

- Working out the central concepts needed in order to formulate pre- and postconditions and invariants (*theory building*) is the most demanding task.
- Usually takes almost half of overall session time. Difficult for both novices and experts.
- Effort for building theory can be amortized over many different programs constructed over the same application domain. Should not be counted fully when evaluating how difficult and time consuming it is to build formally verified programs.
- Theory building needs to be done anyway, in order to specify application program modules, to define application libraries, to determine primitive operations for the application domain, etc.
- Theory building for algorithms in different domains can be seen as one of the central research topics in Computer Science

Research topics

- Semantics and proof theory of invariant based programming
- Scaling up approach to procedures, data modules, classes and objects, concurrent and distributed systems
- Automatic verification of transition correctness
- Using invariant based programming for complex algorithmic problems (e.g., geographic algorithms, pointer manipulation programs, etc.)
- Teaching invariant based programming to novices
- Experimenting with constructing larger, modularized invariant based program

Find out more

- **Imped**: Improving mathematics and programming education in high school (resource center, crest.abo.fi/imped)
    - Using structured derivations (calculational style) in teaching mathematics in high school
    - Using Python as a first programing course in high school
    - Teaching invariant based programming in high school and to first year university/polytechnic students
- Basic papers
    - Back, Ralph-Johan: Invariant based programming: basic approach and teaching experiences, Formal Aspects of Programming 2008
    - Back, Ralph-Johan: Structured derivation as a unified approach to teaching mathematics, Formal Aspects of Programming 2009

Case study: sorting

Problem: Sort an array of integers into non-decreasing order.

The algorithmic solution

- We consider the simplest possible sorting program, selection sort.
- Essentially, we sort the array by moving a cursor from left to right in the array.
- At each stage we find the smallest element to the right of the cursor, and exchange this element with the cursor element.
- After this, we advance the cursor, until we have traversed the whole array.

The algorithmic solution

- We consider the simplest possible sorting program, selection sort.
- Essentially, we sort the array by moving a cursor from left to right in the array.
- At each stage we find the smallest element to the right of the cursor, and exchange this element with the cursor element.
- After this, we advance the cursor, until we have traversed the whole array.

The algorithmic solution

- We consider the simplest possible sorting program, selection sort.
- Essentially, we sort the array by moving a cursor from left to right in the array.
- At each stage we find the smallest element to the right of the cursor, and exchange this element with the cursor element.
- After this, we advance the cursor, until we have traversed the whole array.

The algorithmic solution

- We consider the simplest possible sorting program, selection sort.
- Essentially, we sort the array by moving a cursor from left to right in the array.
- At each stage we find the smallest element to the right of the cursor, and exchange this element with the cursor element.
- After this, we advance the cursor, until we have traversed the whole array.

A little domain theory

- $Sorted(A, i, j)$ means that the array elements are non-decreasing in the (closed) interval $[i, j]$,
- $Partitioned(A, i)$ means that every element in array A below index i is smaller or equal to any element in A at index i or higher, and
- $Permutation(A, A0)$ means that the elements in array A form a permutation of the elements in array $A0$.

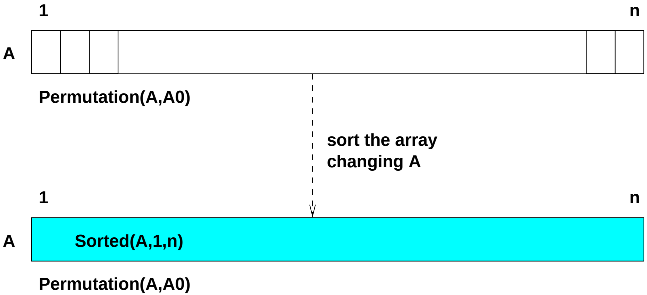
A little domain theory

- $Sorted(A, i, j)$ means that the array elements are non-decreasing in the (closed) interval $[i, j]$,
- $Partitioned(A, i)$ means that every element in array A below index i is smaller or equal to any element in A at index i or higher, and
- $Permutation(A, A0)$ means that the elements in array A form a permutation of the elements in array $A0$.

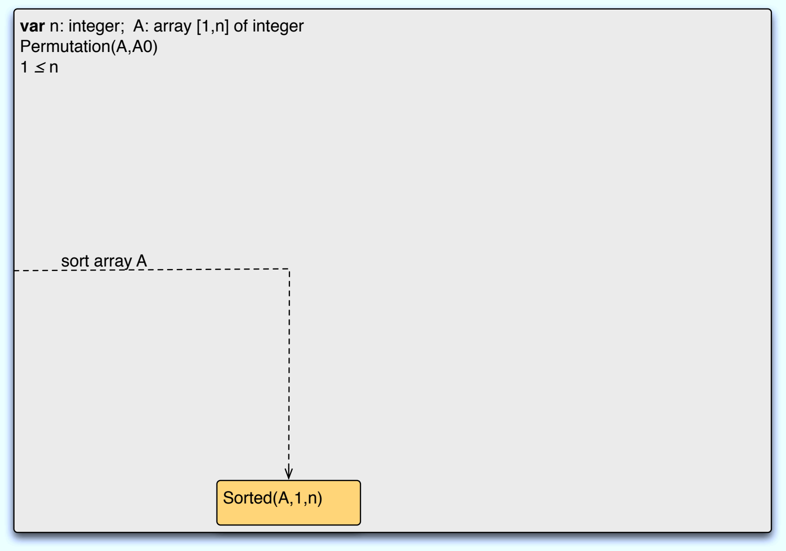
A little domain theory

- $Sorted(A, i, j)$ means that the array elements are non-decreasing in the (closed) interval $[i, j]$,
- $Partitioned(A, i)$ means that every element in array A below index i is smaller or equal to any element in A at index i or higher, and
- $Permutation(A, A0)$ means that the elements in array A form a permutation of the elements in array $A0$.

Initial and final situation



Initial and final situations

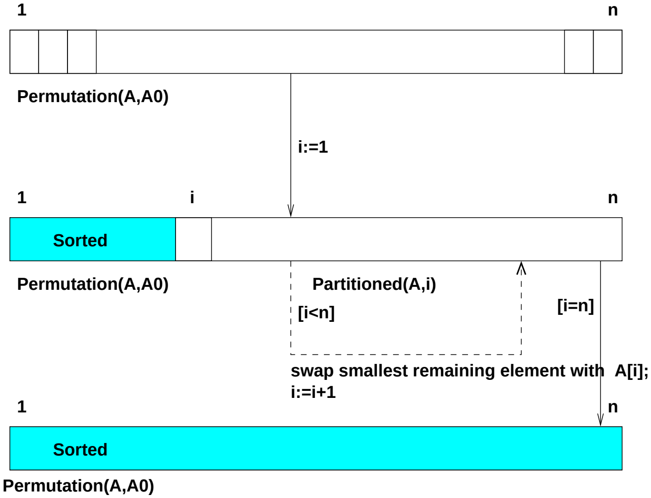


var n: integer; A: array [1,n] of integer
Permutation(A,A0)
1 ≤ n

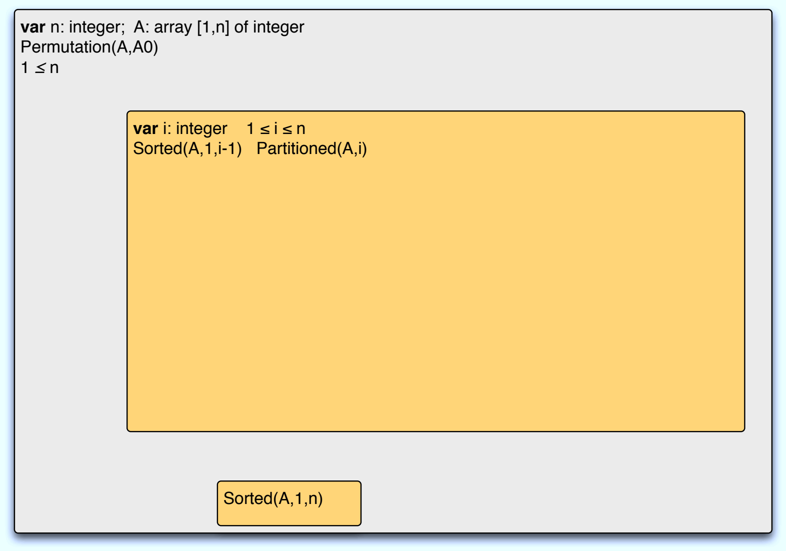
sort array A

Sorted(A,1,n)

Intermediate situation



Formalizing intermediate situation

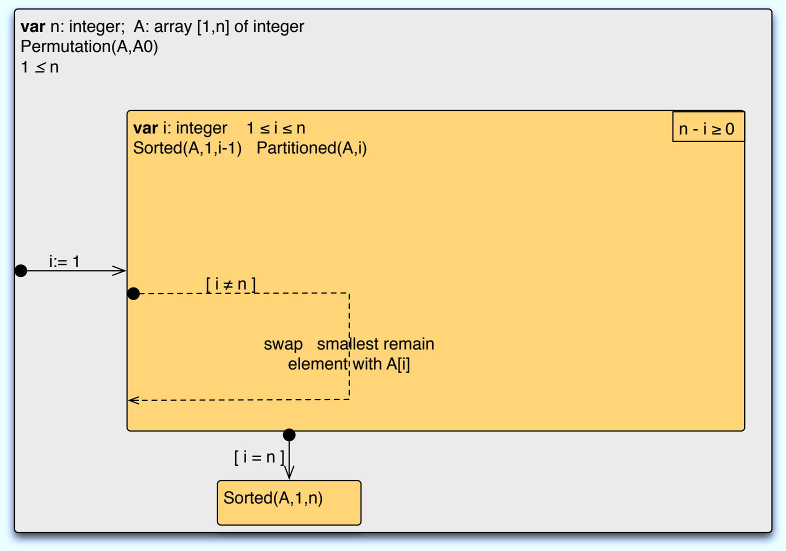


var n: integer; A: array [1,n] of integer
Permutation(A,A0)
1 ≤ n

var i: integer 1 ≤ i ≤ n
Sorted(A,1,i-1) Partitioned(A,i)

Sorted(A,1,n)

Initial and final transitions



var n: integer; A: array [1,n] of integer
Permutation(A,A0)
$1 \le n$

i:= 1

var i: integer $1 \le i \le n$
Sorted(A,1,i-1)   Partitioned(A,i)

$n - i \geq 0$

[$i \neq n$]

swap   smallest remain
element with A[i]

[$i = n$]

Sorted(A,1,n)

Check entry transition

Assume

- $n : \mathit{integer},$
- $A : \ \mathit{array}\ [1, n]\ \mathit{of}\ \mathit{integer}$
- $n \geq 1,$
- $\mathit{Permutation}(A, A0)$

Transition

- $i' = 1$

- $i' : \mathit{integer}$

$1 \leq i' \leq n$

$\mathit{Sorted}(A, 1, i' - 1)$

$\mathit{Partitioned}(A, i')$

$\equiv$ {transition $i' = 1$}

$1 : \mathit{integer}$

$1 \leq 1 \leq n$

$\mathit{Sorted}(A, 1, 0)$

$\mathit{Partitioned}(A, 1)$

$\equiv$ {assumptions}

T

Check entry transition

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$\begin{array}{ll}
\bullet & i' : integer \\
 & 1 \leq i' \leq n \\
 & Sorted(A, 1, i' - 1) \\
 & Partitioned(A, i') \\
\equiv & \{\text{transition } i' = 1\} \\
 & 1 : integer \\
 & 1 \leq 1 \leq n \\
 & Sorted(A, 1, 0) \\
 & Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
 & T
\end{array}$$

Check entry transition

Assume

- $n : integer,$
- $A :\ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$
\begin{array}{ll}
\bullet & i' : integer \\
& 1 \leq i' \leq n \\
& Sorted(A, 1, i' - 1) \\
& Partitioned(A, i') \\
\equiv & \{\text{transition } i' = 1\} \\
& 1 : integer \\
& 1 \leq 1 \leq n \\
& Sorted(A, 1, 0) \\
& Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
& T
\end{array}
$$

Check entry transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$
\begin{array}{ll}
\bullet & i' : integer \\
 & 1 \leq i' \leq n \\
 & Sorted(A, 1, i' - 1) \\
 & Partitioned(A, i') \\
\equiv & \{\text{transition } i' = 1\} \\
 & 1 : integer \\
 & 1 \leq 1 \leq n \\
 & Sorted(A, 1, 0) \\
 & Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
 & T
\end{array}
$$

Check entry transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$
\begin{array}{ll}
\bullet & i' : integer \\
 & 1 \leq i' \leq n \\
 & Sorted(A, 1, i' - 1) \\
 & Partitioned(A, i') \\
\equiv & \{\text{transition } i' = 1\} \\
 & 1 : integer \\
 & 1 \leq 1 \leq n \\
 & Sorted(A, 1, 0) \\
 & Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
 & T
\end{array}
$$

Check entry transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$

Transition

- $i' = 1$

- $i' : integer$

$1 \leq i' \leq n$

$Sorted(A, 1, i' - 1)$

$Partitioned(A, i')$

$\equiv$ {transition $i' = 1$}

$1 : integer$

$1 \leq 1 \leq n$

$Sorted(A, 1, 0)$

$Partitioned(A, 1)$

$\equiv$ {assumptions}

T

Check entry transition

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$
\begin{array}{ll}
\bullet & i' : integer \\
 & 1 \leq i' \leq n \\
 & Sorted(A, 1, i' - 1) \\
 & Partitioned(A, i') \\
\equiv & \{\text{transition } i' = 1\} \\
 & 1 : integer \\
 & 1 \leq 1 \leq n \\
 & Sorted(A, 1, 0) \\
 & Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
 & T
\end{array}
$$

Check entry transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$
\begin{array}{ll}
\bullet & i' : integer \\
& 1 \leq i' \leq n \\
& Sorted(A, 1, i' - 1) \\
& Partitioned(A, i') \\
\equiv & \{\text{transition } i' = 1\} \\
& 1 : integer \\
& 1 \leq 1 \leq n \\
& Sorted(A, 1, 0) \\
& Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
& T
\end{array}
$$

Check entry transition

Assume

- $n : integer,$
- $A : \; array\, [1, n]\; of\; integer$
- $n \geq 1,$
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$
\begin{array}{ll}
\bullet & i' : integer \\
& 1 \leq i' \leq n \\
& Sorted(A, 1, i' - 1) \\
& Partitioned(A, i') \\
\equiv & \{\text{transition } \; i' = 1\} \\
& 1 : integer \\
& 1 \leq 1 \leq n \\
& Sorted(A, 1, 0) \\
& Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
& T
\end{array}
$$

Check entry transition

Assume

- $n : integer$,
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1$,
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$\begin{array}{ll}
\bullet & i' : integer \\
& 1 \leq i' \leq n \\
& Sorted(A, 1, i' - 1) \\
& Partitioned(A, i') \\
\equiv & \{\text{transition } i' = 1\} \\
& 1 : integer \\
& 1 \leq 1 \leq n \\
& Sorted(A, 1, 0) \\
& Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
& T
\end{array}$$

Check entry transition

Assume

- $n : integer,$
- $A : \ array\, [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$

Transition

- $i' = 1$

$$
\begin{array}{ll}
\bullet & i' : integer \\
 & 1 \leq i' \leq n \\
 & Sorted(A, 1, i' - 1) \\
 & Partitioned(A, i') \\
\equiv & \{\text{transition } \ i' = 1\} \\
 & 1 : integer \\
 & 1 \leq 1 \leq n \\
 & Sorted(A, 1, 0) \\
 & Partitioned(A, 1) \\
\equiv & \{\text{assumptions}\} \\
 & T
\end{array}
$$

Check exit transition

Assume

- $n : integer,$
- $A : \ array \ [1, n] \ of \ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

$Sorted(A, 1, i - 1)$

$Partitioned(A, i)$

Transition

- $i = n$

● T

$\Rightarrow$ {assumptions}

$Sorted(A, 1, i - 1) \wedge$
$Partitioned(A, i) \wedge i = n$

$\Rightarrow$ {substitution}

$Sorted(A, 1, n - 1) \wedge$
$Partitioned(A, n)$

$\Rightarrow$ {definition of Sorted and Partitioned}

$Sorted(A, 1, n)$

Check exit transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

$Sorted(A, 1, i - 1)$

$Partitioned(A, i)$

Transition

- $i = n$

$$
\begin{array}{ll}
\bullet & T \\
\Rightarrow & \{\text{assumptions}\} \\
 & Sorted(A, 1, i - 1) \wedge \\
 & Partitioned(A, i) \wedge i = n \\
\Rightarrow & \{\text{substitution}\} \\
 & Sorted(A, 1, n - 1) \wedge \\
 & Partitioned(A, n) \\
\Rightarrow & \{\text{definition of Sorted and} \\
 & \text{Partitioned}\} \\
 & Sorted(A, 1, n)
\end{array}
$$

Check exit transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$

Transition

- $i = n$

- T

$\Rightarrow$ {assumptions}

$Sorted(A, 1, i-1) \wedge$
$Partitioned(A, i) \wedge i = n$

$\Rightarrow$ {substitution}

$Sorted(A, 1, n-1) \wedge$
$Partitioned(A, n)$

$\Rightarrow$ {definition of Sorted and Partitioned}

$Sorted(A, 1, n)$

Check exit transition

Assume

- $n : integer,$
- $A :\ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

$Sorted(A, 1, i-1)$

$Partitioned(A, i)$

Transition

- $i = n$

$\bullet$ T

$\Rightarrow$ {assumptions}

$Sorted(A, 1, i-1) \wedge$
$Partitioned(A, i) \wedge i = n$

$\Rightarrow$ {substitution}

$Sorted(A, 1, n-1) \wedge$
$Partitioned(A, n)$

$\Rightarrow$ {definition of Sorted and Partitioned}

$Sorted(A, 1, n)$

Check exit transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

$Sorted(A, 1, i-1)$

$Partitioned(A, i)$

Transition

- $i = n$

- T

$\Rightarrow$ {assumptions}

$Sorted(A, 1, i-1) \wedge Partitioned(A, i) \wedge i = n$

$\Rightarrow$ {substitution}

$Sorted(A, 1, n-1) \wedge Partitioned(A, n)$

$\Rightarrow$ {definition of Sorted and Partitioned}

$Sorted(A, 1, n)$

Check exit transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

$Sorted(A, 1, i - 1)$

$Partitioned(A, i)$

Transition

- $i = n$

- T

$\Rightarrow$ {assumptions}

$Sorted(A, 1, i - 1) \wedge Partitioned(A, i) \wedge i = n$

$\Rightarrow$ {substitution}

$Sorted(A, 1, n - 1) \wedge Partitioned(A, n)$

$\Rightarrow$ {definition of Sorted and Partitioned}

$Sorted(A, 1, n)$

Check exit transition

Assume

- $n : integer,$
- $A :\ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$

Transition

- $i = n$

$\bullet$ T

$\Rightarrow$ {assumptions}

$Sorted(A, 1, i-1) \wedge Partitioned(A, i) \wedge i = n$

$\Rightarrow$ {substitution}

$Sorted(A, 1, n-1) \wedge Partitioned(A, n)$

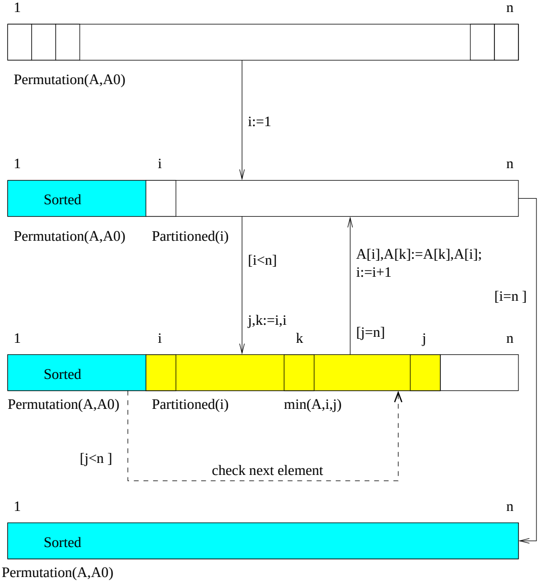
$\Rightarrow$ {definition of Sorted and Partitioned}

$Sorted(A, 1, n)$

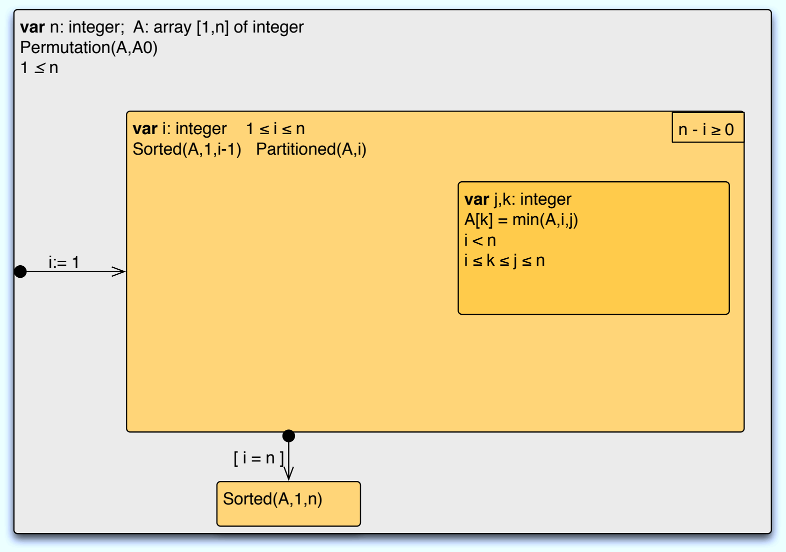
Identifying the smallest element

- To find the smallest remaining element, we need to scan over all the remaining elements
- We need a loop here also.
- We add a fourth situation, where part of the unsorted elements have already been scanned for the least element.

Scanning for smallest element



Scanning situation



var n: integer; A: array [1,n] of integer

Permutation(A,A0)

$1 \leq n$

var i: integer $1 \leq i \leq n$

Sorted(A,1,i-1) Partitioned(A,i)

$n - i \geq 0$

var j,k: integer

A[k] = min(A,i,j)

$i < n$

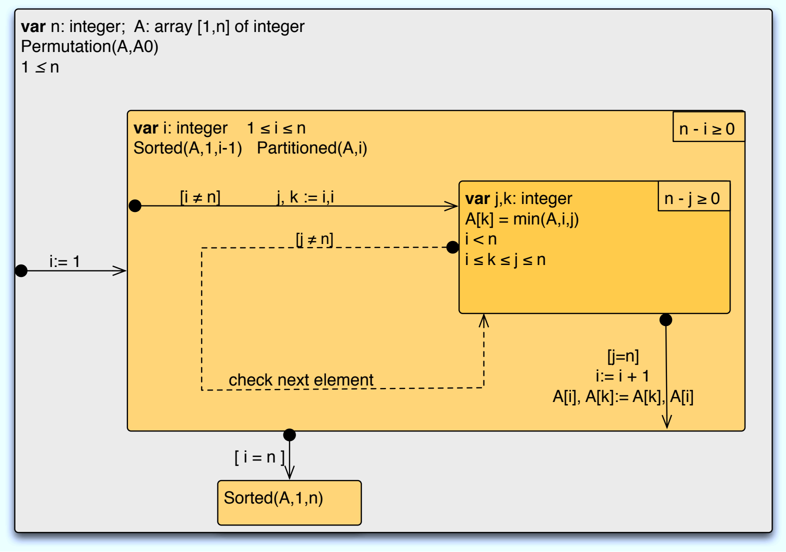
$i \leq k \leq j \leq n$

i:= 1

[i = n]

Sorted(A,1,n)

Scanning transitions



var n: integer; A: array [1,n] of integer

Permutation(A,A0)

1 ≤ n

i:= 1

var i: integer 1 ≤ i ≤ n

Sorted(A,1,i-1) Partitioned(A,i)

n - i ≥ 0

[i ≠ n] j, k := i,i

var j,k: integer

A[k] = min(A,i,j)

i < n

i ≤ k ≤ j ≤ n

n - j ≥ 0

[j ≠ n]

check next element

[j=n]

i:= i + 1

A[i], A[k]:= A[k], A[i]

[i = n]

Sorted(A,1,n)

Check entry transition

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$
- i : *integer*,
- $1 \leq i \leq n$

$Sorted(A, 1, i-1)$

$Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

- k', j' : *integer*

$i \leq k' \leq j' \leq n$

$A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

i, i : *integer*

$i \leq i \leq i \leq n$

$A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

T

Check entry transition

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$
- i : *integer*,
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

- k', j' : *integer*

  $i \leq k' \leq j' \leq n$

  $A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

  i, i : *integer*

  $i \leq i \leq i \leq n$

  $A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

  T

Check entry transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

$Sorted(A, 1, i-1)$

$Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

- $k', j' : integer$

  $i \leq k' \leq j' \leq n$

  $A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

$i, i : integer$

$i \leq i \leq i \leq n$

$A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

T

Check entry transition

Assume

- $n : integer$,
- $A : \ array \ [1, n] \ of \ integer$
- $n \geq 1$,
- $Permutation(A, A0)$
- $i : integer$,
- $1 \leq i \leq n$

$Sorted(A, 1, i-1)$

$Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

- $k', j' : integer$

$i \leq k' \leq j' \leq n$

$A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

$i, i : integer$

$i \leq i \leq i \leq n$

$A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

T

Check entry transition

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$
- i : *integer*,
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

- k', j' : *integer*

  $i \leq k' \leq j' \leq n$

  $A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

  i, i : *integer*

  $i \leq i \leq i \leq n$

  $A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

  T

Check entry transition

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$
- i : *integer*,
- $1 \leq i \leq n$

$Sorted(A, 1, i-1)$

$Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

$\bullet$ k', j' : *integer*

$i \leq k' \leq j' \leq n$

$A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

i, i : *integer*

$i \leq i \leq i \leq n$

$A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

T

Check entry transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

● $k', j' : integer$

$i \leq k' \leq j' \leq n$

$A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

$i, i : integer$

$i \leq i \leq i \leq n$

$A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

T

Check entry transition

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i - 1)$

  $Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

$\bullet$ $k', j' : integer$

$i \leq k' \leq j' \leq n$

$A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

$i, i : integer$

$i \leq i \leq i \leq n$

$A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

T

Check entry transition

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$
- i : *integer*,
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$

Transition

- $i \neq n$
- $j' = i$
- $k' = i$

$\bullet$ k', j' : *integer*

$i \leq k' \leq j' \leq n$

$A[k'] = min\{A[h] | i \leq h \leq j'\}$

$\equiv$ {substituting $j' = i$ and $k' = i$}

i, i : *integer*

$i \leq i \leq i \leq n$

$A[i] = min\{A[h] | i \leq h \leq i\}$

$\equiv$ {assumption $i \leq n$}

T

Check exit transition

We also check that if $j = n$, then

$$A[i], A[k] := A[k], A[i];\; i := i + 1$$

will establish the first intermediate situation, as indicated in the diagram. Need to check all constraints that involve A and i.

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

- $i' : integer \wedge 1 \leq i' \leq n$

  $Sorted(A', 1, i'-1)$

  $Partitioned(A', i')$

  $Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$
- i : *integer*,
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$
- $j.k$: *integer*
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

- i' : *integer* $\wedge 1 \leq i' \leq n$

  $Sorted(A', 1, i'-1)$

  $Partitioned(A', i')$

  $Permutation(A', A_0)$

$\equiv$ $\{ i' = i + 1\}$

$i+1$: *integer* $\wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i - 1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

- $i' : integer \wedge 1 \leq i' \leq n$

  $Sorted(A', 1, i' - 1)$

  $Partitioned(A', i')$

  $Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Assume

- $n : integer$,
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1$,
- $Permutation(A, A0)$
- $i : integer$,
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

- $i' : integer \wedge 1 \leq i' \leq n$

  $Sorted(A', 1, i'-1)$

  $Partitioned(A', i')$

  $Permutation(A', A_0)$

$\equiv$ $\{ i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i - 1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

● $i' : integer \wedge 1 \leq i' \leq n$

$Sorted(A', 1, i' - 1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Assume

- $n : integer$,
- $A : \; array\ [1, n]\ of\ integer$
- $n \geq 1$,
- $Permutation(A, A0)$
- $i : integer$,
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

- $i' : integer \wedge 1 \leq i' \leq n$

  $Sorted(A', 1, i'-1)$

  $Partitioned(A', i')$

  $Permutation(A', A_0)$

$\equiv$ $\{ \; i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$
- i : *integer*,
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$
- $j.k$: *integer*
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

• i' : *integer* $\wedge 1 \leq i' \leq n$

$Sorted(A', 1, i'-1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i+1$: *integer* $\wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Assume

- $n : integer,$
- $A : \; array \; [1, n] \; of \; integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

● $i' : integer \wedge 1 \leq i' \leq n$

$Sorted(A', 1, i'-1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{ \; i' = i + 1\}$

$i{+}1 : integer \wedge 1 \leq i{+}1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]\}$

T

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

● $i' : integer \wedge 1 \leq i' \leq n$

$Sorted(A', 1, i'-1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i - 1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

● $i' : integer \wedge 1 \leq i' \leq n$

$Sorted(A', 1, i' - 1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i{+}1 : integer \wedge 1 \leq i{+}1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

Assume

- n : *integer*,
- A : *array* $[1, n]$ *of integer*
- $n \geq 1$,
- $Permutation(A, A0)$
- i : *integer*,
- $1 \leq i \leq n$

  $Sorted(A, 1, i-1)$

  $Partitioned(A, i)$
- $j.k$: *integer*
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

● i' : *integer* $\wedge 1 \leq i' \leq n$

$Sorted(A', 1, i'-1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{ i' = i + 1\}$

$i+1$: *integer* $\wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i+1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

$\mathcal{T}$

Assume

- $n : integer,$
- $A : \ array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i - 1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

● $i' : integer \wedge 1 \leq i' \leq n$

$Sorted(A', 1, i' - 1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

Assume

- $n : integer$,
- $A :\ array\ [1, n]\ of\ integer$
- $n \geq 1$,
- $Permutation(A, A0)$
- $i : integer$,
- $1 \leq i \leq n$

  $Sorted(A, 1, i - 1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

• $i' : integer \wedge 1 \leq i' \leq n$

$Sorted(A', 1, i' - 1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

Assume

- $n : integer,$
- $A : \; array\ [1, n]\ of\ integer$
- $n \geq 1,$
- $Permutation(A, A0)$
- $i : integer,$
- $1 \leq i \leq n$

  $Sorted(A, 1, i - 1)$

  $Partitioned(A, i)$
- $j.k : integer$
- $A[k] = min(A, i, j)$
- $i < n$
- $i \leq k \leq j \leq n$

Transition

- $j = n$
- $i' = i + 1$
- $A' = A[i : A[k], k : A[i]]$

$\bullet$ $i' : integer \wedge 1 \leq i' \leq n$

$Sorted(A', 1, i' - 1)$

$Partitioned(A', i')$

$Permutation(A', A_0)$

$\equiv$ $\{\ i' = i + 1\}$

$i+1 : integer \wedge 1 \leq i+1 \leq n$

$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$Permutation(A', A_0)$

$\equiv$ {assumption $i < n$, swapping preserves permutation}

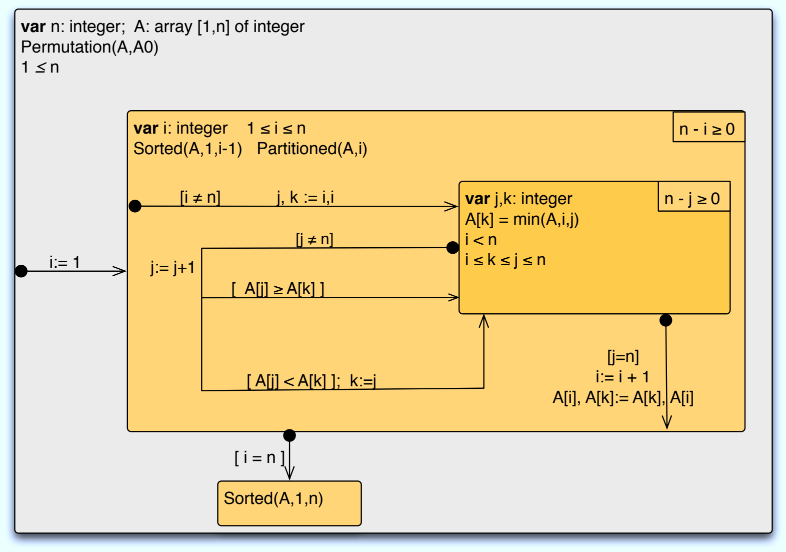
$Sorted(A', 1, i)$

$Partitioned(A', i + 1)$

$\equiv$ {assumption $A[k] = min(A, i, n)$ and $A' = A[i : A[k], k : A[i]]$}

T

Innermost loop



var n: integer; A: array [1,n] of integer
Permutation(A,A0)
1 ≤ n

i:= 1

var i: integer 1 ≤ i ≤ n
Sorted(A,1,i-1) Partitioned(A,i)

n - i ≥ 0

[i ≠ n] j, k := i,i

var j,k: integer
A[k] = min(A,i,j)
i < n
i ≤ k ≤ j ≤ n

n - j ≥ 0

[j ≠ n]

j:= j+1

[A[j] ≥ A[k]]

[A[j] < A[k]]; k:=j

[j=n]
i:= i + 1
A[i], A[k]:= A[k], A[i]

[i = n]

Sorted(A,1,n)

Preserving the invariant

- We need to check that the second invariant is preserved while making progress. We need to show that when $j \neq n$, the statement

$$j := j + 1; \text{if } A[j] < A[k] \;\text{ then } k := j \;\text{ fi}$$

preserves the second invariant.

- The inner loop will eventually terminate because $n - j$ is decreased but is bounded from below.
- The outer loop will terminate because $n - i$ is decreased and is bounded from below.
- Liveness not a problem, because all transitions from intermediate situations are mutually exhaustive

Preserving the invariant

- We need to check that the second invariant is preserved while making progress. We need to show that when $j \neq n$, the statement

$$j := j + 1; \text{if } A[j] < A[k] \ \text{ then } k := j \ \text{ fi}$$

preserves the second invariant.

- The inner loop will eventually terminate because $n - j$ is decreased but is bounded from below.
- The outer loop will terminate because $n - i$ is decreased and is bounded from below.
- Liveness not a problem, because all transitions from intermediate situations are mutually exhaustive

Preserving the invariant

- We need to check that the second invariant is preserved while making progress. We need to show that when $j \neq n$, the statement

$$j := j + 1; \text{if } A[j] < A[k] \text{ then } k := j \text{ fi}$$

preserves the second invariant.

- The inner loop will eventually terminate because $n - j$ is decreased but is bounded from below.
- The outer loop will terminate because $n - i$ is decreased and is bounded from below.
- Liveness not a problem, because all transitions from intermediate situations are mutually exhaustive

Preserving the invariant

- We need to check that the second invariant is preserved while making progress. We need to show that when $j \neq n$, the statement

$$j := j + 1; \text{if } A[j] < A[k] \ \text{ then } k := j \ \text{ fi}$$

preserves the second invariant.

- The inner loop will eventually terminate because $n - j$ is decreased but is bounded from below.
- The outer loop will terminate because $n - i$ is decreased and is bounded from below.
- Liveness not a problem, because all transitions from intermediate situations are mutually exhaustive

Final program

- The final program developed above has been proved correct
- It can be automatically compiled into executable program code, in any language
- Compiler does not need information about situations, only transitions are needed in order to generate code

Final program

- The final program developed above has been proved correct
- It can be automatically compiled into executable program code, in any language
- Compiler does not need information about situations, only transitions are needed in order to generate code

Final program

- The final program developed above has been proved correct
- It can be automatically compiled into executable program code, in any language
- Compiler does not need information about situations, only transitions are needed in order to generate code