Correction d'un algorithme

Jean-Marc Talbot et Emmanuel Beffara
emmanuel.beffara@univ-amu.fr

9 ou 16 avril 2019

Correction d'un algorithme ?

- un algorithme doit **terminer** sur toute entrée admise (terminaison)
- s'il termine alors la valeur calculée par l'algorithme doit être **correcte au regard de ce qu'on attend de lui** (correction partielle)

Correction d'un algorithme ?

- un algorithme doit **terminer** sur toute entrée admise (terminaison)
- s'il termine alors la valeur calculée par l'algorithme doit être **correcte au regard de ce qu'on attend de lui** (correction partielle)

$$\text{correction} = \text{correction partielle} + \text{terminaison}$$

Correction d'un algorithme ?

- un algorithme doit **terminer** sur toute entrée admise (terminaison)
- s'il termine alors la valeur calculée par l'algorithme doit être **correcte au regard de ce qu'on attend de lui** (correction partielle)

$$\text{correction} = \text{correction partielle} + \text{terminaison}$$

« Correct au regard de ce qu'on attend de lui » ? Spécification : propriété des sorties de l'algorithme

Exemple: la sortie est le quotient de a par b

Comment garantir la correction partielle d'un algorithme ?

- **Test**: on essaie différentes entrées possibles et on vérifie que la sortie est conforme à la spécification.
  - facile ☺
  - permet de trouver des bugs ☺
  - présuppose la terminaison 😐
  - formellement, ne garantit que les valeurs testées ☹

Comment garantir la correction partielle d'un algorithme ?

- **Test**: on essaie différentes entrées possibles et on vérifie que la sortie est conforme à la spécification.
  - facile ☺
  - permet de trouver des bugs ☺
  - présuppose la terminaison 😐
  - formellement, ne garantit que les valeurs testées ☹
- **Preuve**: on prouve formellement que pour **toute entrée**, la sortie est conforme à la spécification.
  - garantit formellement pour toutes les valeurs d'entrées possibles ☺
  - difficile ☹

Propriétés de quoi ?

La propriété désirée/spécification porte :

- sur la sortie de l'algorithme
- et donc, sur la valeur d'une ou plusieurs variables obtenue à la fin de l'algorithme.

Propriétés de quoi ?

La propriété désirée/spécification porte :

- sur la sortie de l'algorithme
- et donc, sur la valeur d'une ou plusieurs variables obtenue à la fin de l'algorithme.

Ces valeurs à cet endroit dépendent :

- de la valeur d'autres variables
- a priori à d'autres endroits

Propriétés de quoi ?

La propriété désirée/spécification porte :

- sur la sortie de l'algorithme
- et donc, sur la valeur d'une ou plusieurs variables obtenue à la fin de l'algorithme.

Ces valeurs à cet endroit dépendent :

- de la valeur d'autres variables
- a priori à d'autres endroits

État: associer à chaque variable une valeur ou *non-défini* "_"

$$(a \mapsto 17, b \mapsto 3, \text{résultat} \mapsto \_)$$

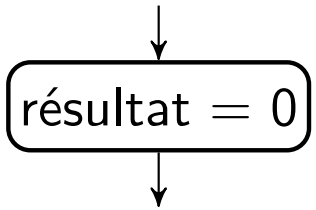
Spécification = propriété de l'état atteint à la fin (du déroulement) de l'algorithme

Déroulement de l'algorithme pas à pas

- L'exécution d'une instruction fait passer d'un état à un autre
- Pour une instruction ou un bloc d'instructions `Inst`, son exécution depuis un état e mène à l'état `Inst`(e).

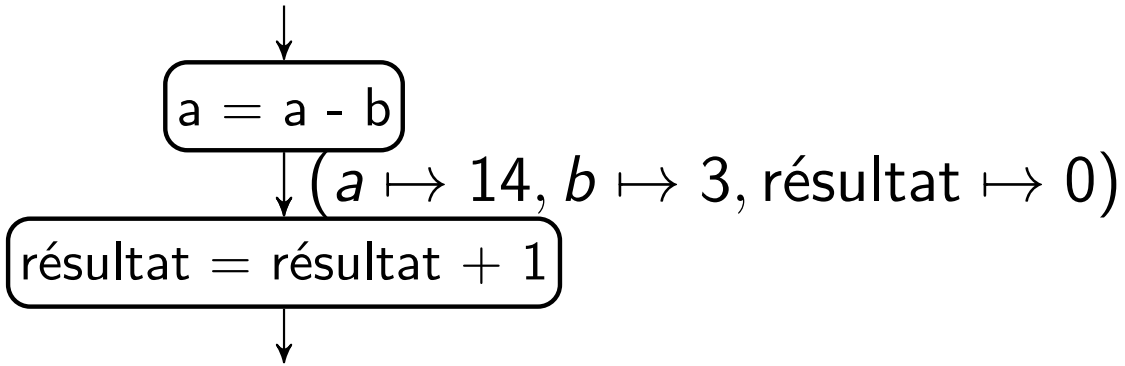
Déroulement de l'algorithme pas à pas

$(a \mapsto 17, b \mapsto 3, \text{résultat} \mapsto \_)$



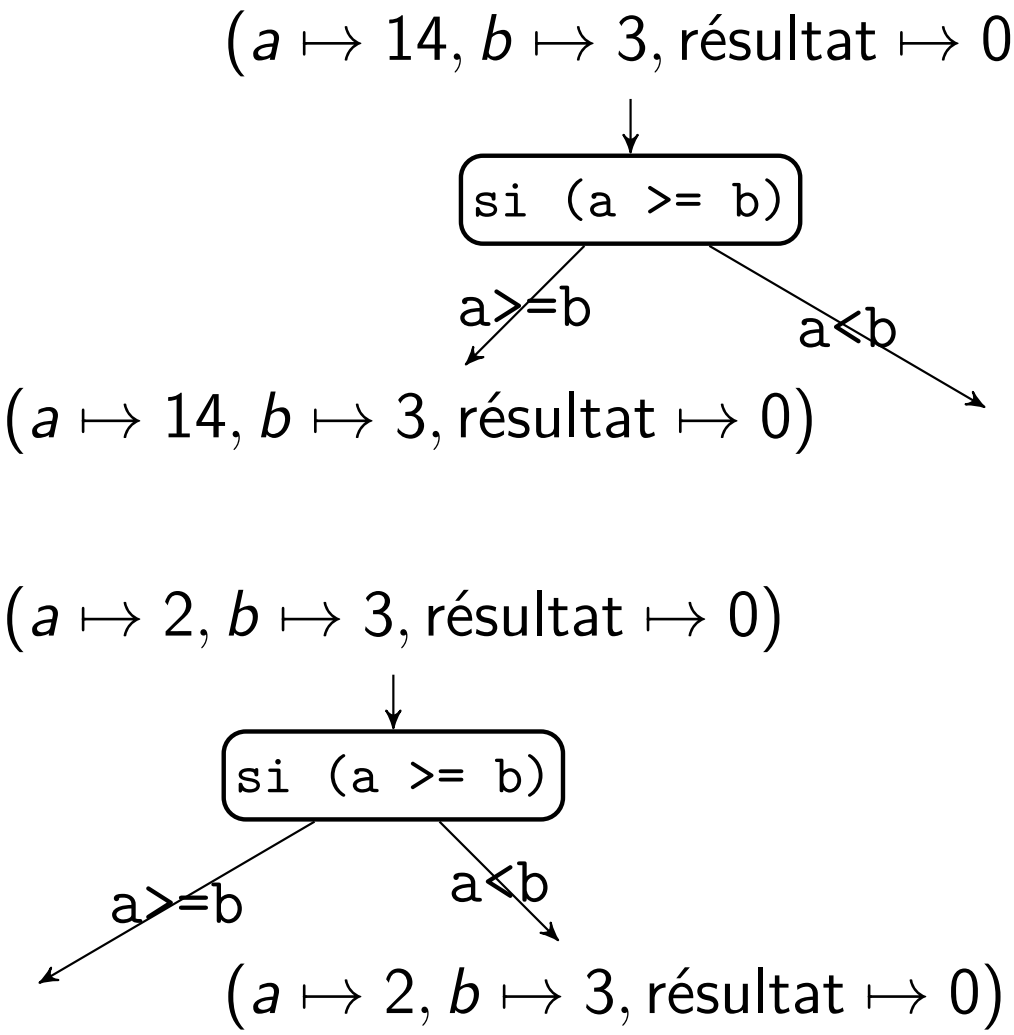
$(a \mapsto 17, b \mapsto 3, \text{résultat} \mapsto 0)$

$(a \mapsto 17,\ b \mapsto 3,\ \text{résultat} \mapsto 0)$



$(a \mapsto 14,\ b \mapsto 3,\ \text{résultat} \mapsto 1)$

Déroulement de l'algorithme pas à pas (II)



Tester une condition change le flot de contrôle mais **ne change pas l'état (des variables).**

Déroulement de l'algorithme et graphe de flot de contrôle

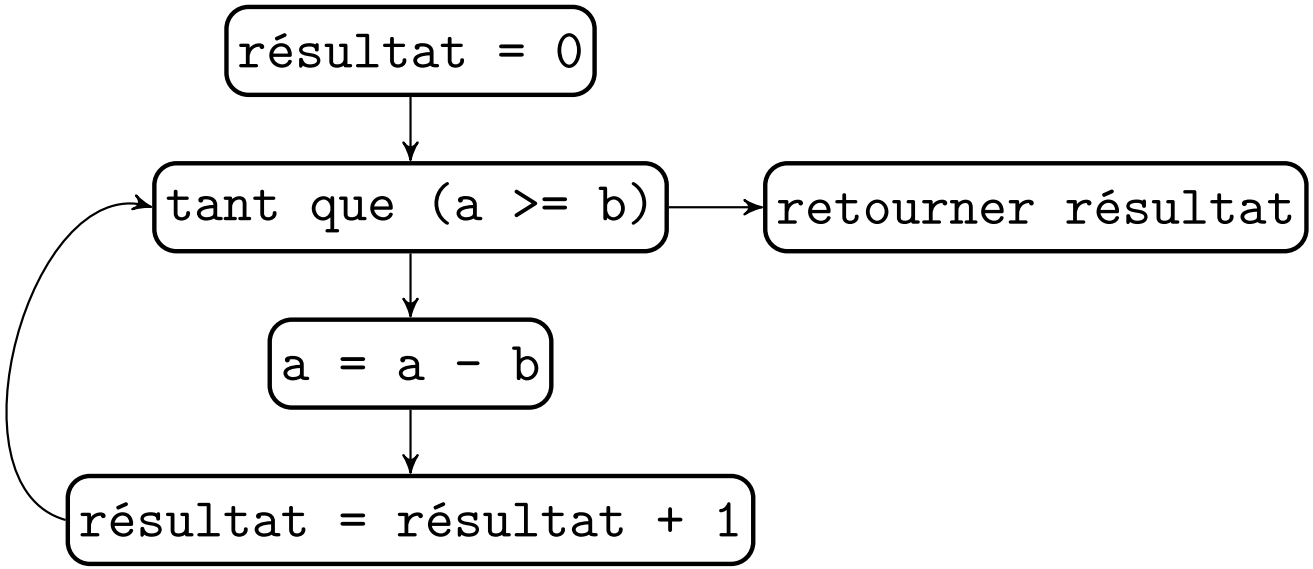
Algorithme de division euclidienne par soustraction

entrées :

a, un entier

b, un entier non-nul

sortie : le quotient de a par b



Etats et propriétés

Propriété d'un état :

- ne parle que des variables définies (de l'état) : `x, y, z, a, b`
- mais aussi éventuellement des variables initiales au bloc d'instructions en considération de l'algorithme (par exemple, les paramètres pour l'algorithme dans sa globalité) x_0, y_0,z_0,a_0,b_0, ...
- est soit vraie, soit fausse

Etats et propriétés

Propriété d'un état :

- ne parle que des variables définies (de l'état) : `x, y, z, a, b`
- mais aussi éventuellement des variables initiales au bloc d'instructions en considération de l'algorithme (par exemple, les paramètres pour l'algorithme dans sa globalité) x_0, y_0,z_0,a_0,b_0, ...
- est soit vraie, soit fausse

Les propriétés sont décrites / données par des **assertions** définies:

- textuellement de manière non-ambiguë : « x est strictement plus grand que y »
- par une formule logique : $x > y$

Etats et propriétés

$$A : (x > y \ \wedge y \geq y_0) \vee (z > 0)$$

Etats et propriétés

$$A : (x > y \ \wedge y \geq y_0) \vee (z > 0)$$

Pour une assertion A, $A(e)$ signifie que A est vraie pour l'état e.

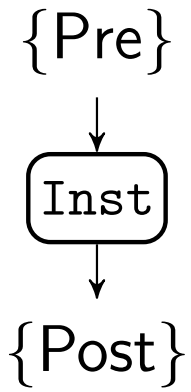
Si $y_0 = 0$, alors

- $A((x \mapsto 10, y \mapsto 4, z \mapsto 1))$,
- $A((x \mapsto 10, y \mapsto -1, z \mapsto 1))$,
- $A((x \mapsto 10, y \mapsto 4, z \mapsto -1))$
- mais pas $A((x \mapsto 10, y \mapsto -1, z \mapsto -1))$

Instructions et propriétés

Soit `Inst` une instruction (ou un bloc d'instructions), {Pre} et {Post} deux assertions

$\{Pre\}\texttt{Inst}\{Post\}$



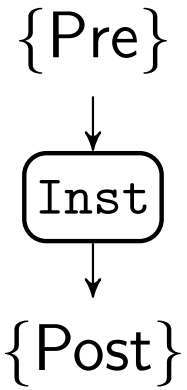
Sens

Pour tous les états e satisfaisant l'assertion {Pre}, après l'exécution de `Inst`, si celle-ci termine, `Inst`(e) satisfait l'assertion {Post}.

Instructions et propriétés

Soit `Inst` une instruction (ou un bloc d'instructions), {Pre} et {Post} deux assertions

$\{Pre\}\texttt{Inst}\{Post\}$



Sens

Pour tous les états e satisfaisant l'assertion {Pre}, après l'exécution de `Inst`, si celle-ci termine, `Inst`(e) satisfait l'assertion {Post}.

{Pre} est la **pré-condition**, {Post} est la **post-condition**

Correction : reformulation

En généralisant à un bloc d'instruction `BInst`,

```
Algo
entrées : x1,..., xn
sortie : y
début
  BInst;
fin
```

Correction partielle de `Algo`

Si Pre(`x1,...,xn`) et que l'exécution de `BInst` termine pour `x1,...,xn` alors Post(`y`).

Correction : reformulation

En généralisant à un bloc d'instruction `BInst`,

```
Algo
entrées : x1,..., xn
sortie : y
début
  BInst;
fin
```

Correction partielle de `Algo`

Si Pre(`x1,...,xn`) et que l'exécution de `BInst` termine pour `x1,...,xn` alors Post(`y`).
L'algorithme est correct sous la pré-condition Pre(`x1,...,xn`), si celui-ci termine et si Post(`y`) est (ou implique) la spécification de `Algo`.

Instructions et assertions/propriétés : affectation

$\{x \geq 0\}$ `x = x+1` $\{x > 0\}$

Instructions et assertions/propriétés : affectation

$\{x \geq 0\}$ `x = x+1` $\{x > 0\}$

$\{x \geq 0\}$ `x = x+1` $\{x \geq 0\}$

Instructions et assertions/propriétés : affectation

$\{x \geq 0\}$ `x = x+1` $\{x > 0\}$

$\{x \geq 0\}$ `x = x+1` $\{x \geq 0\}$

$\{x \geq 0\}$ `x = x+1` $\{x$ est un entier$\}$

Instructions et assertions/propriétés : affectation

$\{x \geq 0\}$ `x = x+1` $\{x > 0\}$

$\{x \geq 0\}$ `x = x+1` $\{x \geq 0\}$

$\{x \geq 0\}$ `x = x+1` $\{x$ est un entier$\}$

Noter que: $\{x > 0\}$ implique que $\{x \geq 0\}$ implique que $\{x$ est un entier$\}$

Rappel

A implique B: tout état qui satisfait A satisfait aussi B.

Instructions et assertions/propriétés : affectation (II)

$\{x \geq 0 \text{ et } y > 0\}$ `x = x+1` $\{x > 0 \text{ et } y > 0\}$

Instructions et assertions/propriétés : affectation (II)

$\{x \geq 0 \text{ et } y > 0\}$ `x = x+1` $\{x > 0 \text{ et } y > 0\}$

On n'a pas $\{x \geq 0 \text{ et } y > x\}$ `x = x+1` $\{x > 0 \text{ et } y > x\}$

Instructions et assertions/propriétés : affectation (II)

$\{x \geq 0 \text{ et } y > 0\}$ `x = x+1` $\{x > 0 \text{ et } y > 0\}$

On n'a pas $\{x \geq 0 \text{ et } y > x\}$ `x = x+1` $\{x > 0 \text{ et } y > x\}$

Mais $\{x \geq 0 \text{ et } y > x\}$ `x = x+1` $\{x > 0 \text{ et } y \geq x\}$

$\{x \geq 0 \text{ et } y \geq x\}$ `x = x+1` $\{x > 0 \text{ et } y \geq x - 1\}$

Instructions et assertions/propriétés : affectation (III)

De manière générale,

$\{A\}$ `x = expression` $\{B\}$

si $\{A\}$ implique $\{B[x \leftarrow \textit{expression}]\}$

Note: $\{B[x \leftarrow \textit{expression}]\}$ *est* $\{B\}$ *dans lequel tous les* x *ont été remplacés par expression*

Instructions et assertions/propriétés : affectation (III)

De manière générale,

$\{A\}$ `x = expression` $\{B\}$

si $\{A\}$ implique $\{B[x \leftarrow \textit{expression}]\}$

Note: $\{B[x \leftarrow \textit{expression}]\}$ *est* $\{B\}$ *dans lequel tous les* x *ont été remplacés par expression*

Exemple: $\{x \geq 0 \text{ et } y > x\}$ `x = x+1` $\{x > 0 \text{ et } y \geq x\}$

car $\{x > 0 \text{ et } y \geq x\}[x \leftarrow x + 1] = \{x + 1 > 0 \text{ et } y \geq x + 1\}$

qui est équivalent à (et donc, impliqué par) $\{x \geq 0 \text{ et } y > x\}$

Instructions et assertions/propriétés : affectation (IV)

$$\{2x = y\}$$
```
x = x + 1;
```
$$\{2x = y + 2\}$$
```
y = y + 2;
```
$$\{2x = y\}$$

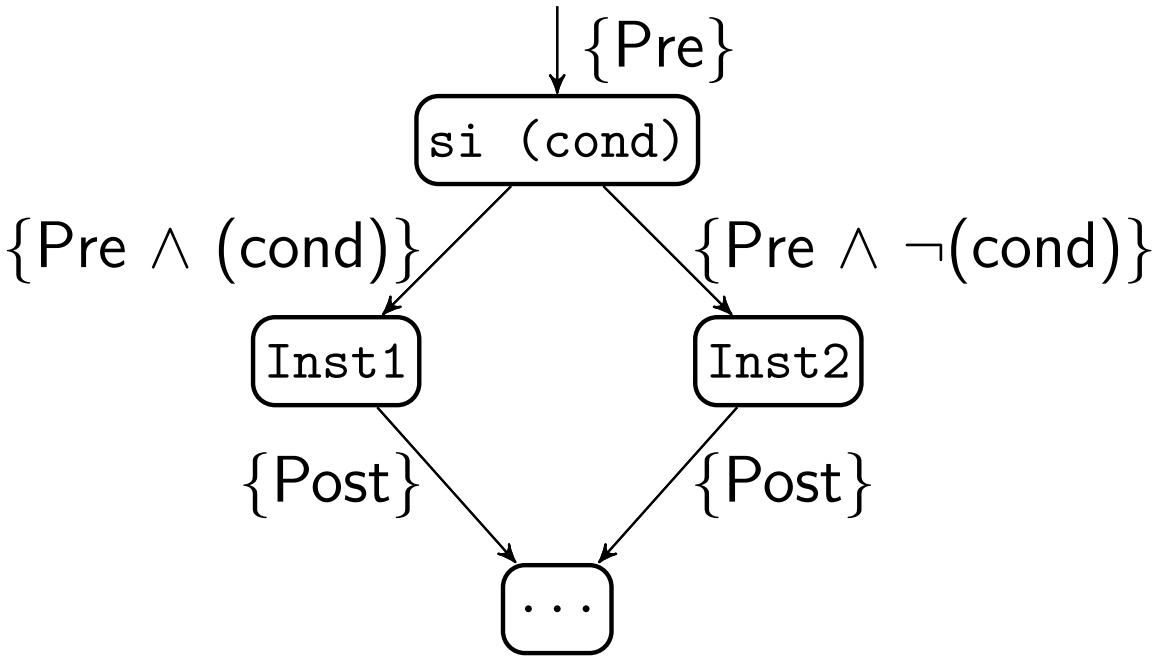
Une pré-condition calculée possible pour le bloc est alors $\{2(x + 1) = y + 2\}$,
qui est équivalent et donc impliqué par $\{2x = y\}$.

Instructions et assertions/propriétés : conditionnelle

{Pre} `si (cond) alors Inst1 sinon Inst2 finsi` {Post}

Instructions et assertions/propriétés : conditionnelle

$\{Pre\}$ `si (cond) alors Inst1 sinon Inst2 finsi` $\{Post\}$



Instructions et assertions/propriétés : conditionnelle (II)

La valeur absolue est un entier positif

$\{x$ est un entier$\}$
`si (x>=0) alors`
  $\{x \geq 0\}$
  `abs = x`
  $\{abs \geq 0\}$
`sinon`
  $\{x < 0\}$
  `abs = -x`
  $\{abs > 0\}$
`finsi`
$\{abs \geq 0\}$

OK car $\{abs > 0\}$ *implique* $\{abs \geq 0\}$

Instructions et assertion/propriétés : boucle

Définition (Invariant)

Un **invariant de boucle**

```
tant que (condition) faire
    BInst
fintantque;
```

est une assertion A telle que pour tout état e à l'entrée de la boucle, si $A(e) \wedge \texttt{condition}(e)$ est vraie et $\texttt{BInst}(e)$ termine alors $A(\texttt{BInst}(e))$

Invariants

Théorème de l'invariant

Considérons une boucle dont la condition est `(condition)` et A un invariant de cette boucle.

Si A est vraie avant l'exécution de la boucle alors l'assertion $A \wedge \neg$`(condition)` est vraie après l'exécution de la boucle.

Théorème de l'invariant : preuve

Montrons tout d'abord par récurrence sur $n \geq 0$, que si pour l'état e_0 à l'entrée de la boucle, on a $A(e_0)$ alors pour tout état e_n obtenu après n itérations de la boucle, on a $A(e_n)$:

Théorème de l'invariant : preuve

Montrons tout d'abord par récurrence sur $n \geq 0$, que si pour l'état e_0 à l'entrée de la boucle, on a $A(e_0)$ alors pour tout état e_n obtenu après n itérations de la boucle, on a $A(e_n)$:

- Pour $n = 0$, $A(e_n)$ est $A(e_0)$ et est vraie par hypothèse.

Théorème de l'invariant : preuve

Montrons tout d'abord par récurrence sur $n \geq 0$, que si pour l'état e_0 à l'entrée de la boucle, on a $A(e_0)$ alors pour tout état e_n obtenu après n itérations de la boucle, on a $A(e_n)$:

- Pour $n = 0$, $A(e_n)$ est $A(e_0)$ et est vraie par hypothèse.
- Considérons e_n un état obtenu après n itérations et on suppose l'existence d'une itération supplémentaire; ainsi, $\texttt{condition}(e_n)$ est vrai. Considérons $\texttt{BInst}(e_n)$ qui est, par définition, l'état obtenu après $n+1$ itérations, soit e_{n+1}. Par hypothèse de récurrence, on a $A(e_n)$. Puisque, A est un invariant, que $A(e_n)$ et $\texttt{condition}(e_n)$ sont vraies, alors on a $A(\texttt{BInst}(e_n))$, soit $A(e_{n+1})$.

Théorème de l'invariant : preuve

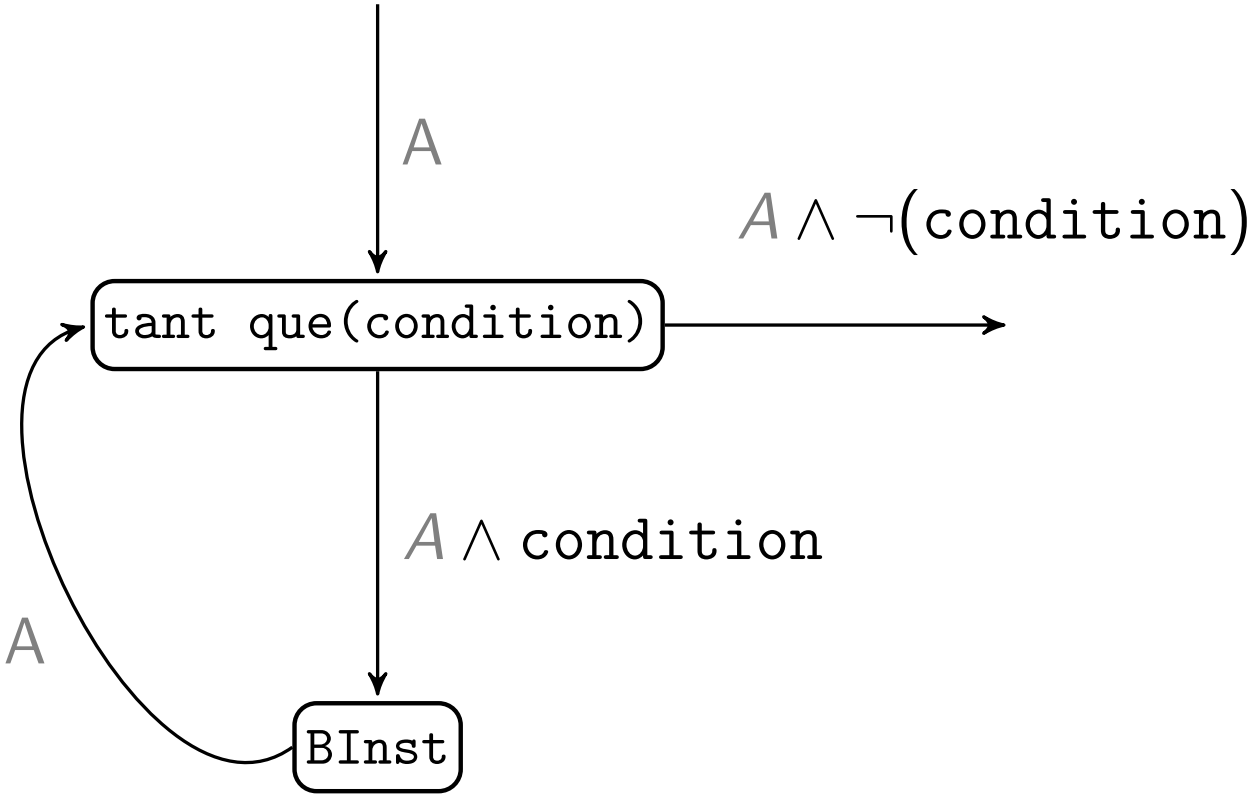
Montrons tout d'abord par récurrence sur $n \geq 0$, que si pour l'état e_0 à l'entrée de la boucle, on a $A(e_0)$ alors pour tout état e_n obtenu après n itérations de la boucle, on a $A(e_n)$:

- Pour $n = 0$, $A(e_n)$ est $A(e_0)$ et est vraie par hypothèse.
- Considérons e_n un état obtenu après n itérations et on suppose l'existence d'une itération supplémentaire; ainsi, $\texttt{condition}(e_n)$ est vrai. Considérons $\texttt{BInst}(e_n)$ qui est, par définition, l'état obtenu après $n+1$ itérations, soit e_{n+1}. Par hypothèse de récurrence, on a $A(e_n)$. Puisque, A est un invariant, que $A(e_n)$ et $\texttt{condition}(e_n)$ sont vraies, alors on a $A(\texttt{BInst}(e_n))$, soit $A(e_{n+1})$.

Lorsque la boucle termine (après un certain nombre m d'itérations) dans un état e, on a alors $A(e) \wedge \neg\texttt{condition(e)}$.

Invariants (II)

Graphiquement,



Correction partielle : méthodologie

Le but est d'annoter le graphe de flot de contrôle avec des assertions, vraies pour tous les états qui y peuvent apparaitre dans une exécution et cela, pour n'importe quelles entrées. L'assertion finale devra impliquer la spécification souhaitée.

- pour les affectations et les conditionnelles on peut calculer simplement la post-condition en fonction de la pré-condition.
- pour les boucles :
  - on choisit **de manière judicieuse** un invariant pour chaque boucle
  - on vérifie que cet invariant est impliqué par la pré-condition de la boucle
  - on vérifie que l'invariant est correct
  - on vérifie que l'invariant et la négation de la condition de la boucle implique la post-condition de la boucle