Validation d'algorithmes

Chapitre I

Preuve d'algorithmes : Variants et invariants.

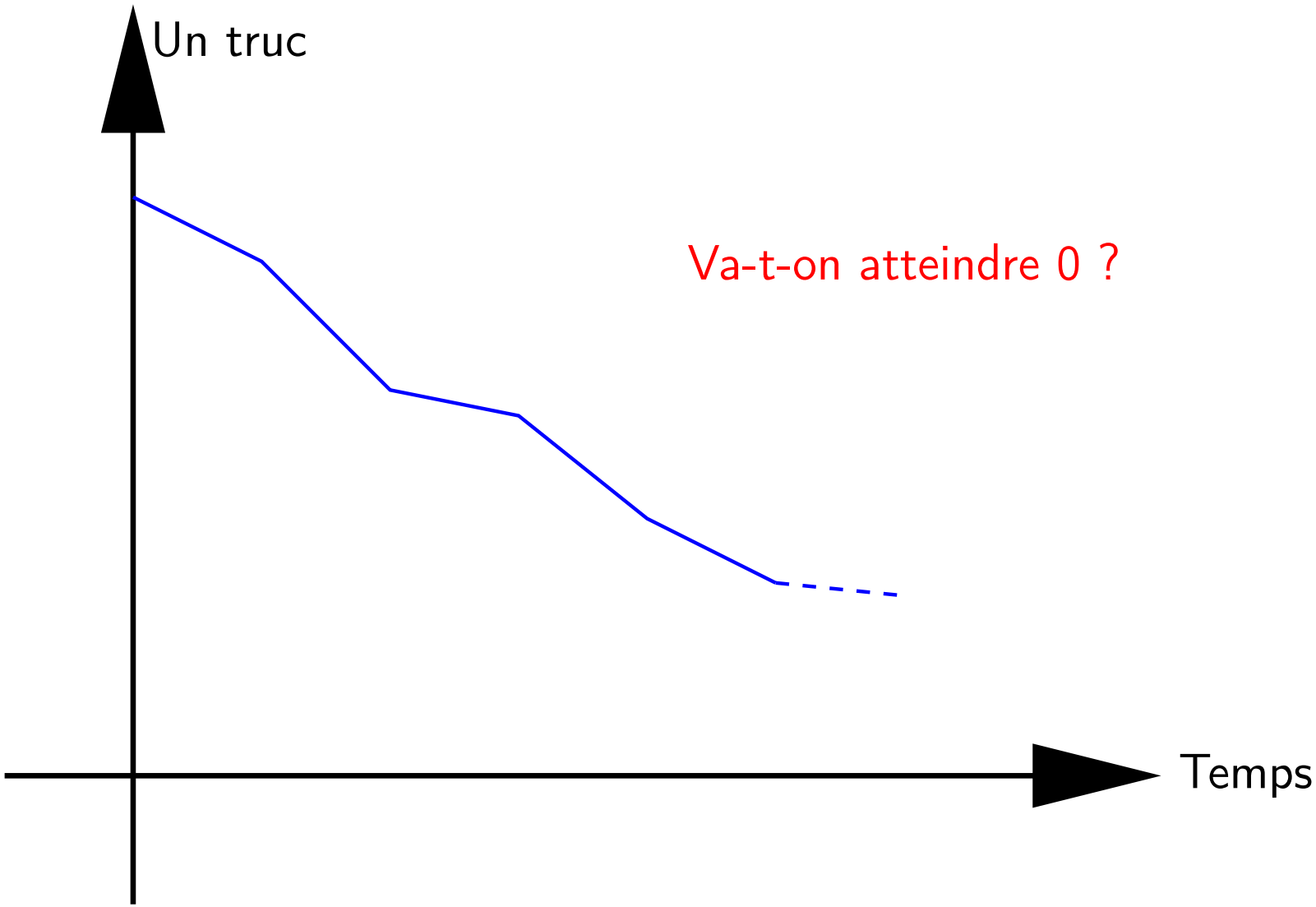
Raphaël Charrondière raphael.charrondiere@inria.fr

Dans ce premier chapitre :

- Comment faire une preuve ?
- Différencier : « ça termine » de « le résultat est correct »
- Gérer les boucles
- Les variants, les invariants
- Un peu de maths : ordre bien fondé.


Je me demande quand cette barre va atteindre les 100%

Ça descend



Dans un espace E (exemple $\mathbb{N}, \mathbb{R}$)

- Ordre ($\leqslant$)
  - réflexive $\forall x \in E,\ x \leqslant x$
  - antisymétrique $\forall x, y \in E,\ (x \leqslant y \wedge y \leqslant x) \implies x = y$
  - transitive $\forall x, y, z \in E,\ (x \leqslant y \wedge y \leqslant z) \implies x \leqslant z$
- Ordre total
  - $\forall x, y \in E,\ x \leqslant y \vee y \leqslant x$
- Ordre strict ($<$)
  - asymétrique $\forall x, y \in E,\ (x < y \vee x = y) \implies \neg(y < x)$
  - transitive $\forall x, y, z \in E,\ (x < y \wedge y < z) \implies x < z$

Relation d'ordre bien fondée

Une relation d'ordre bien fondée est une relation d'ordre tel qu'il n'existe pas de suite infinie décroissante $\neg\exists (x_n)_n \in E^{\mathbb{N}},\ \forall k \in \mathbb{N},\ x_{k+1} < x_k$

À vous de jouer

Exemples de relation d'ordre ?

Comment passer d'un ordre à un ordre strict ?

Pourquoi doit-on avoir un ordre bien fondé et pas seulement un ordre strict ?

Exemples et contre-exemples d'ordres bien fondés ?

Ici on confondra programme, algorithme même si en pratique on peut les différencier.
On note $\mathbb{M}$ l'ensemble des états mémoires.

Comment décrivez-vous un programme de manière générale ?

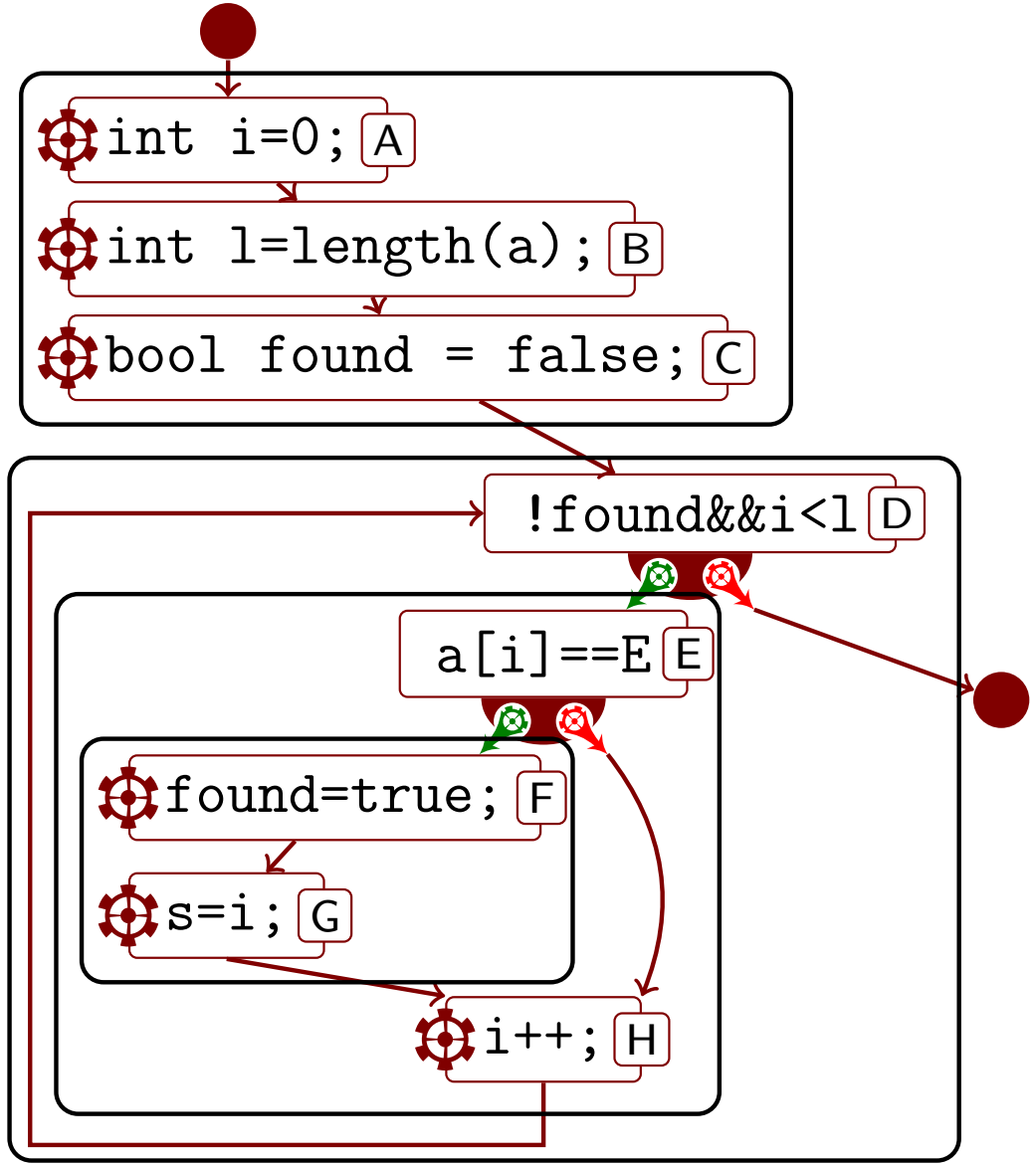
Un programme, un algorithme ?

On considère qu'un programme est constitué d'une séquence d'instructions élémentaires

- Des instructions élémentaires qui n'agissent que sur la mémoire, les entrées/sorties
- De sauts conditionnels

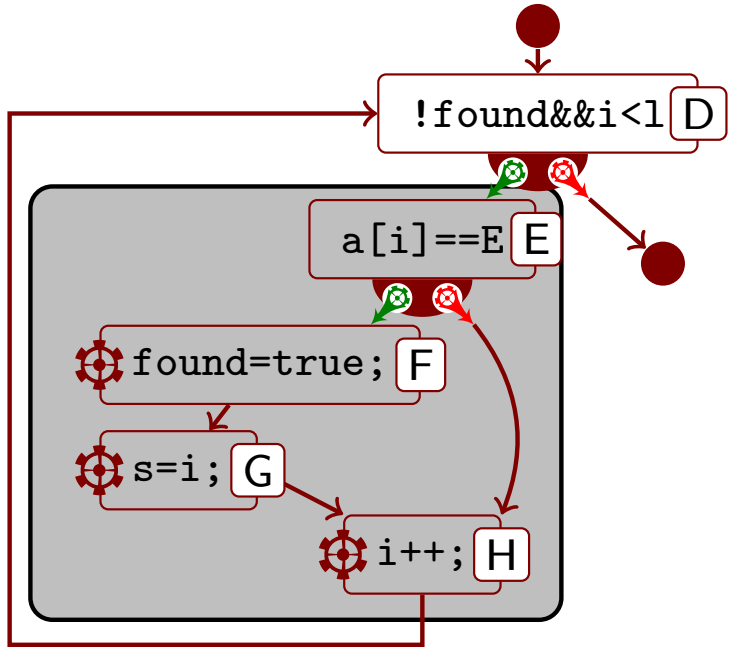
On appellera **bloc d'instructions** un ensemble d'instructions élémentaires autonomes (aucun saut conditionnel ne sort du bloc).

```
int i=0;
int l=length(a);
bool found=false;
while(!found&&i<l)
{
   if (a[i]==E) {
      found=true;
      s=i;
   }
   i++;
}
```



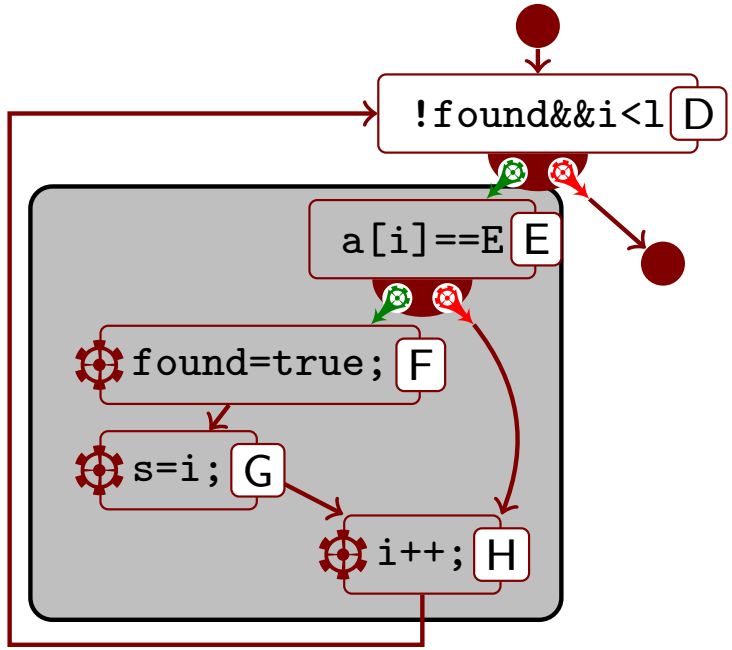
Un programme termine ssi pour tout état mémoire en entrée, l'exécution termine.
On décompose un programme en bloc d'instructions et sauts conditionnels.
Hypothèse : Chaque bloc termine.
Soit E un espace muni d'un ordre bien fondé, soit $\phi : \mathbb{M} \to E$ une fonction (nommée ... fonction potentielle ou variant).
Si l'on parvient à prouver que ϕ diminue (au sens de l'ordre bien fondé) entre le début et la fin de chaque bloc, alors le programme termine.

Hypothèse : Le bloc unique (gris) termine.



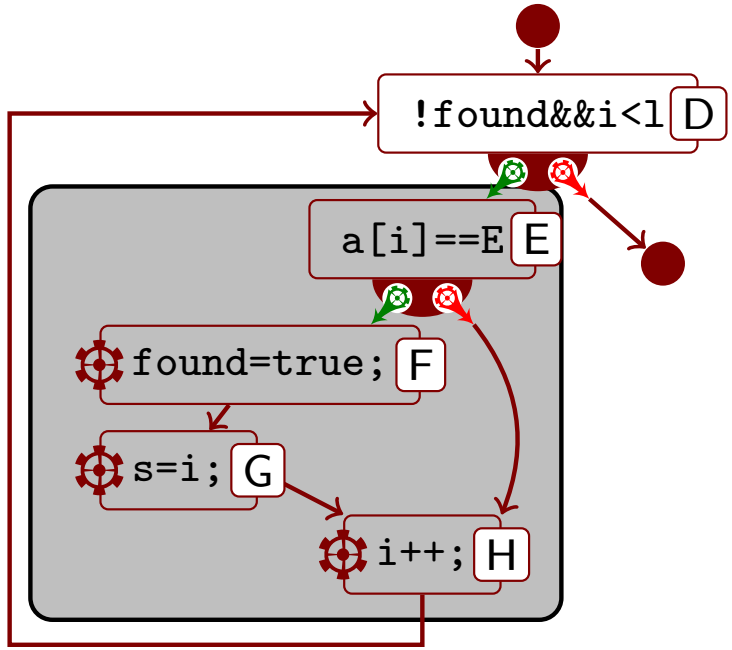
Hypothèse : Le bloc unique (gris) termine.

- On se place dans l'ensemble $\mathbb{N}$, et on utilise la relation $<$ classique.



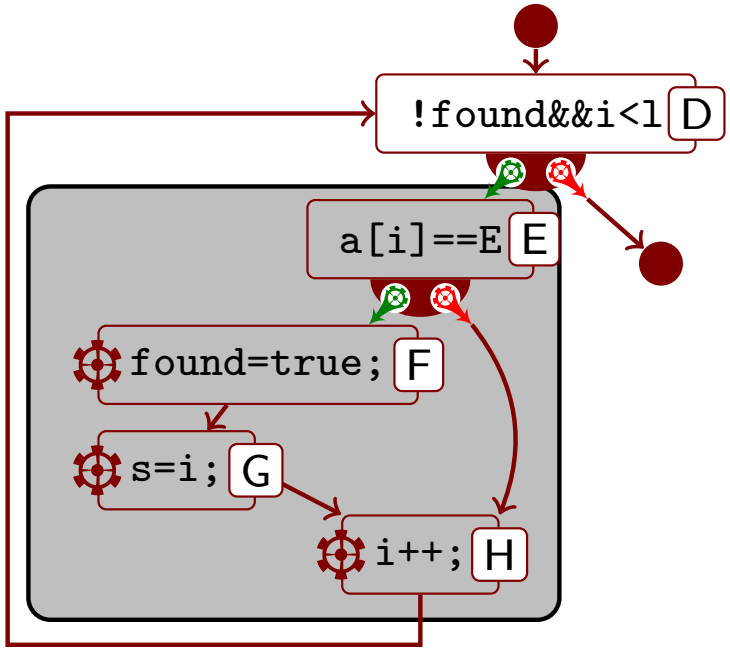
Hypothèse : Le bloc unique (gris) termine.

- On se place dans l'ensemble $\mathbb{N}$, et on utilise la relation $<$ classique.
- On définit $\phi(m) = m(l) - m(i)$. $\phi(m)$ est bien dans $\mathbb{N}$ en début de boucle.



Hypothèse : Le bloc unique (gris) termine.

- On se place dans l'ensemble $\mathbb{N}$, et on utilise la relation $<$ classique.
- On définit $\phi(m) = m(l) - m(i)$. $\phi(m)$ est bien dans $\mathbb{N}$ en début de boucle.
- Si on a m_0 en début de boucle, m_1 après l'exécution du bloc gris alors $\phi(m_1) = m_1(l) - m_1(i) = m_0(l) - (m_0(i) + 1) = \phi(m_0) - 1$ Ainsi, $\phi(m_1) < \phi(m_0)$, le "`while`" termine



Invariant

- On décompose un programme en bloc d'instructions et sauts conditionnels.

Invariant

- On décompose un programme en bloc d'instructions et sauts conditionnels.
- On note m_0 l'état mémoire avant l'exécution et m la mémoire à l'état courant.

Invariant

- On décompose un programme en bloc d'instructions et sauts conditionnels.
- On note m_0 l'état mémoire avant l'exécution et m la mémoire à l'état courant.
- Soit $P(m_0, m)$ une proposition logique.

Invariant

- On décompose un programme en bloc d'instructions et sauts conditionnels.
- On note m_0 l'état mémoire avant l'exécution et m la mémoire à l'état courant.
- Soit $P(m_0, m)$ une proposition logique.
- En supposant $P(m_0, m_0)$, P est un invariant si après l'exécution de chaque bloc on a $P(m_0, m)$

Invariant

- On décompose un programme en bloc d'instructions et sauts conditionnels.
- On note m_0 l'état mémoire avant l'exécution et m la mémoire à l'état courant.
- Soit $P(m_0, m)$ une proposition logique.
- En supposant $P(m_0, m_0)$, P est un invariant si après l'exécution de chaque bloc on a $P(m_0, m)$
- **En pratique :** Il faut montrer que si en début d'un bloc on a l'état mémoire m_i, en fin m_f $P(m_0, m_i) \implies P(m_0, m_f)$, c'est une sorte de récurrence.

Invariant

- On décompose un programme en bloc d'instructions et sauts conditionnels.
- On note m_0 l'état mémoire avant l'exécution et m la mémoire à l'état courant.
- Soit $P(m_0, m)$ une proposition logique.
- En supposant $P(m_0, m_0)$, P est un invariant si après l'exécution de chaque bloc on a $P(m_0, m)$
- **En pratique :** Il faut montrer que si en début d'un bloc on a l'état mémoire m_i, en fin m_f $P(m_0, m_i) \implies P(m_0, m_f)$, c'est une sorte de récurrence.
- Attention, il faut cependant être flexible, on peut vouloir ajouter des contraintes sur m_i, s'il y a une condition avant (mais il faut justifier)

Définition

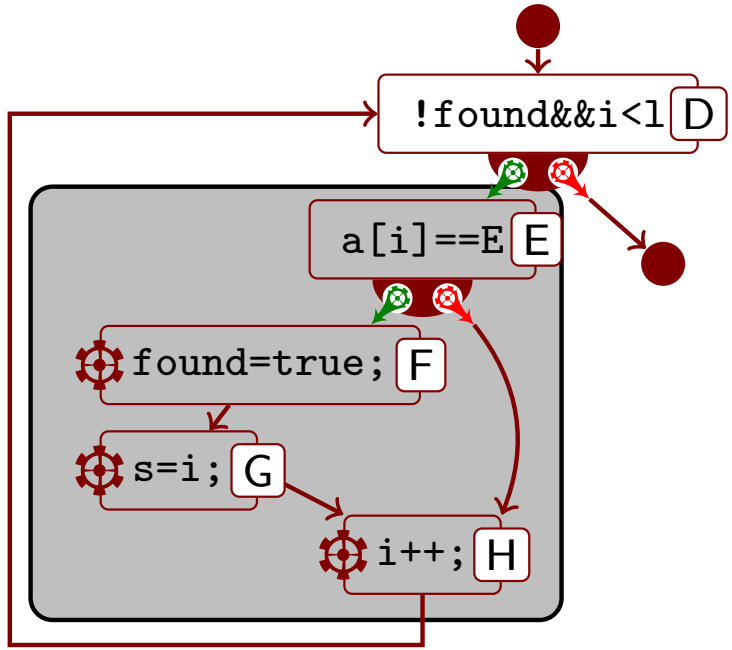
Une propriété sur l'ensemble des blocs : $P(m_0, m_f)$
"Si l'état mémoire en entré est m_0, et l'état mémoire en sortie est m_f, alors $P(m_0, m_f)$ est vrai".

- Un invariant sert à prouver quelque chose : par exemple qu'une fonction trie une liste.
- Un invariant peut se compléter en une propriété. Par exemple l'invariant sera "les éléments de la liste sont les même que ceux de la liste donnée en entrée" et la propriété "La sortie est la liste d'entrée dont les éléments ont été triés".

Exemple, on continue

- On définit l'invariant

$$P(m_0, m) \equiv$$
$$\Big(m(found) \wedge m_0(a)[m(s)] = E$$
$$\vee \forall j \in [\![m_0(i), m(i) - 1]\!],\ m_0(a)[j] \neq E\Big)$$
$$\wedge m_0(a) = m(a)$$

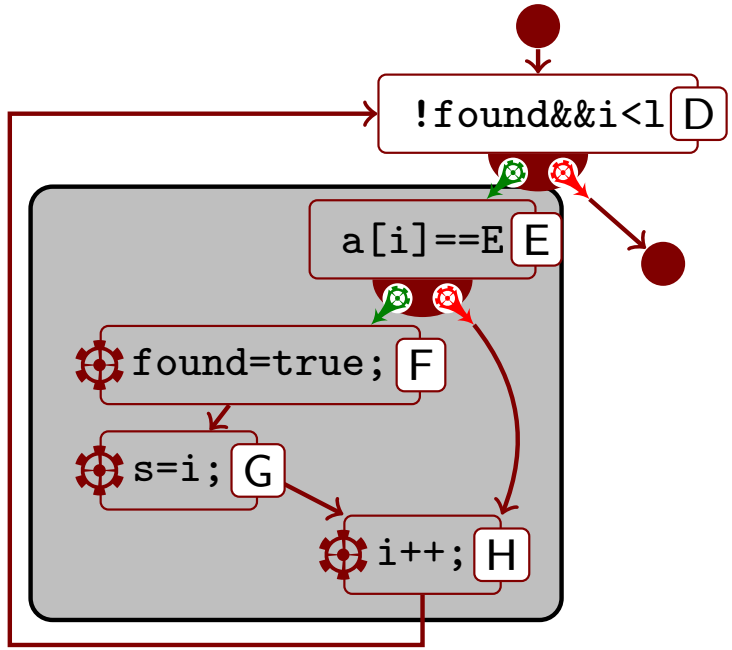


Exemple, on continue

- On définit l'invariant

$$P(m_0, m) \equiv$$
$$\Big(m(found) \wedge m_0(a)[m(s)] = E$$
$$\vee \forall j \in [\![m_0(i), m(i) - 1]\!],\ m_0(a)[j] \neq E\Big)$$
$$\wedge m_0(a) = m(a)$$

- « Soit on a trouvé, soit on n'a pas trouvé »

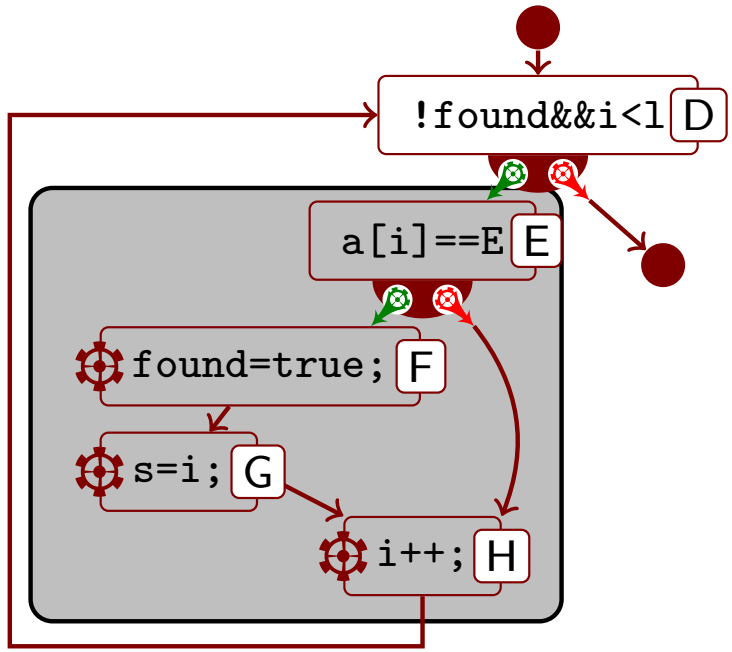


Exemple, on continue

- On définit l'invariant

$$P(m_0, m) \equiv$$
$$\Big(m(found) \wedge m_0(a)[m(s)] = E$$
$$\vee \forall j \in [\![m_0(i), m(i) - 1]\!],\ m_0(a)[j] \neq E\Big)$$
$$\wedge m_0(a) = m(a)$$

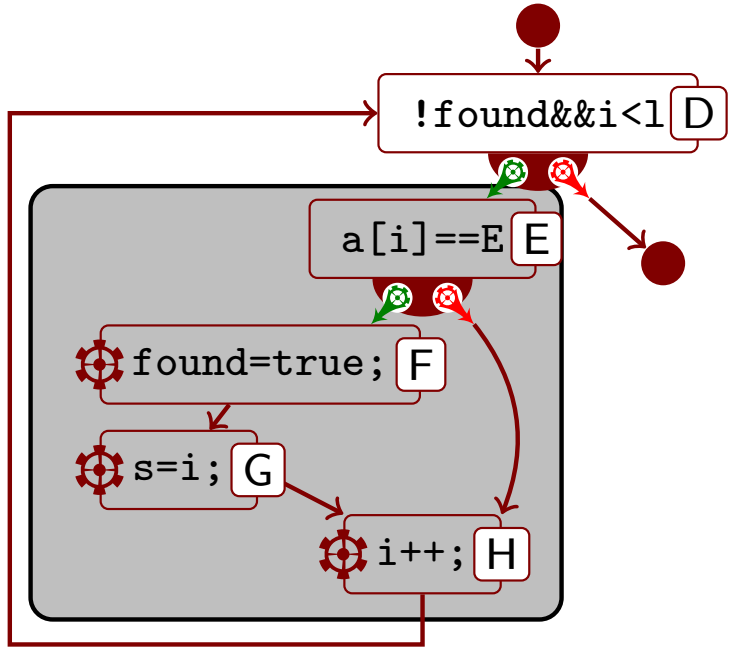
- « Soit on a trouvé, soit on n'a pas trouvé »
- « a reste identique »



- On définit l'invariant

$$P(m_0, m) \equiv$$
$$\Big(m(found) \wedge m_0(a)[m(s)] = E$$
$$\vee \forall j \in [\![m_0(i), m(i) - 1]\!],\ m_0(a)[j] \neq E\Big)$$
$$\wedge m_0(a) = m(a)$$

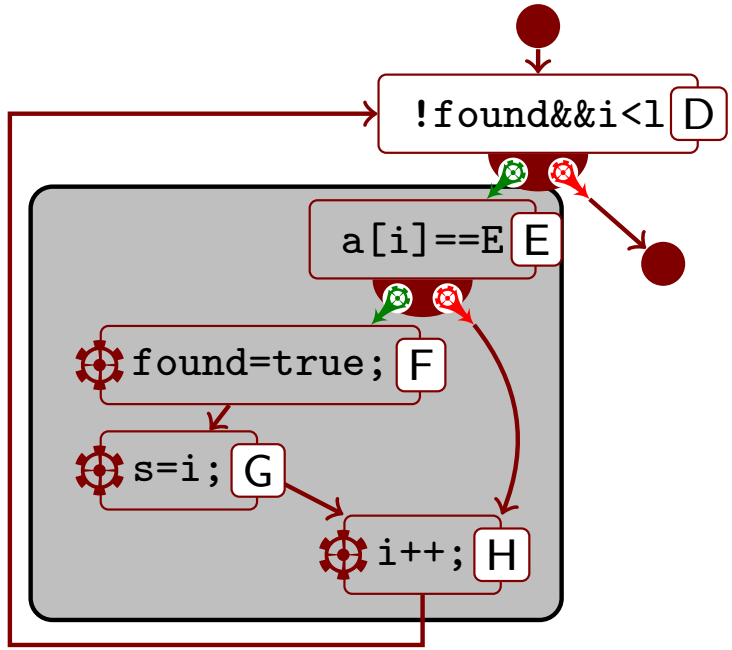
- « Soit on a trouvé, soit on n'a pas trouvé »
- « a reste identique »
- Pour le bloc gris on montre « $P(m_0, m_i) \implies P(m_0, m_f)$ » On peut rajouter $\neg m_i(found)$ (condition en D), la preuve est assez immédiate.



Exemple, on continue

- Et la propriété

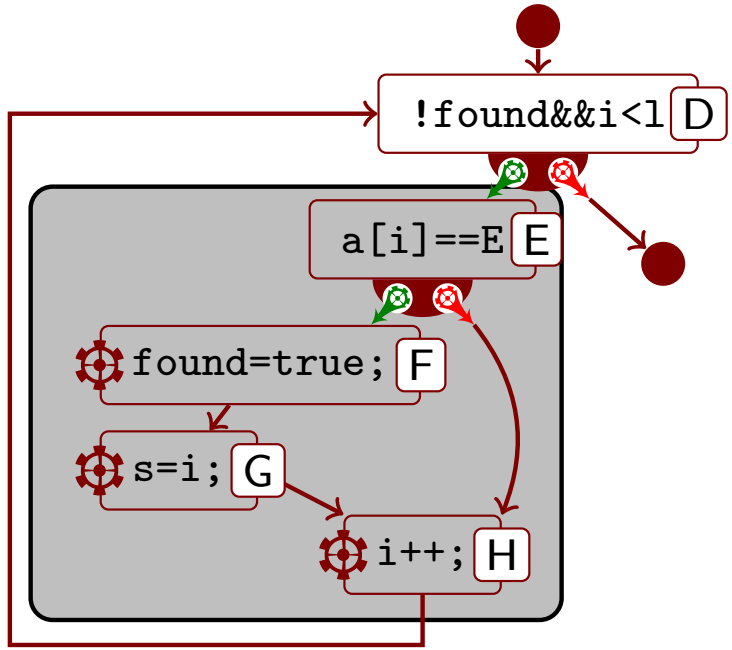
$$
\begin{aligned}
Q(m_0, m_f) \equiv \\
&(m_f(found) \wedge m_0(a)[m_f(s)] = E) \\
&\vee \Big(\neg m_f(found) \\
&\quad \wedge \forall j \in [\![m_0(i), m_0(l) - 1]\!],\ m_0(a)[j] \neq E\Big)
\end{aligned}
$$



- Et la propriété

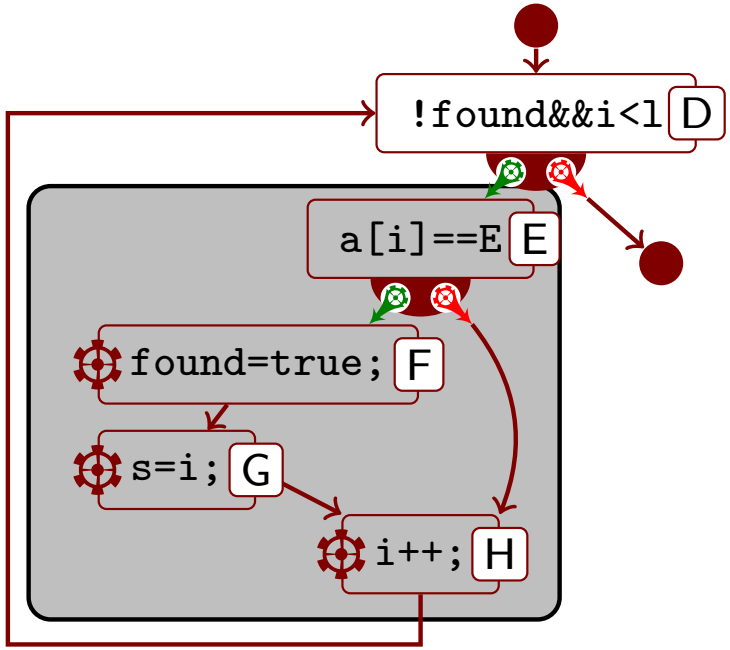
$$Q(m_0, m_f) \equiv$$
$$(m_f(found) \wedge m_0(a)[m_f(s)] = E)$$
$$\vee \left(\neg m_f(found)\right.$$
$$\left. \wedge \forall j \in [\![m_0(i), m_0(l) - 1]\!],\ m_0(a)[j] \neq E\right)$$

- On doit montrer qu'en sortie Q est vérifiée, c-a-d :



- Et la propriété

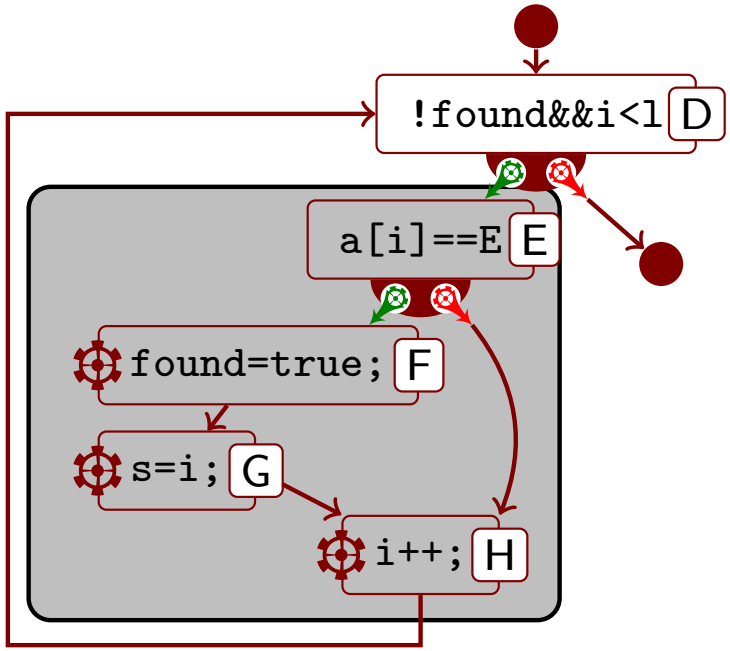
$$Q(m_0, m_f) \equiv$$
$$(m_f(found) \wedge m_0(a)[m_f(s)] = E)$$
$$\vee \left(\neg m_f(found)\right.$$
$$\left. \wedge \forall j \in [\![m_0(i), m_0(l) - 1]\!],\ m_0(a)[j] \neq E\right)$$



- On doit montrer qu'en sortie Q est vérifiée, c-a-d :
- $P(m_0, m_0) \wedge P(m_0, m_f) \wedge (m_f(found) \vee i \geqslant l) \implies Q(m_0, m_f)$

- Et la propriété

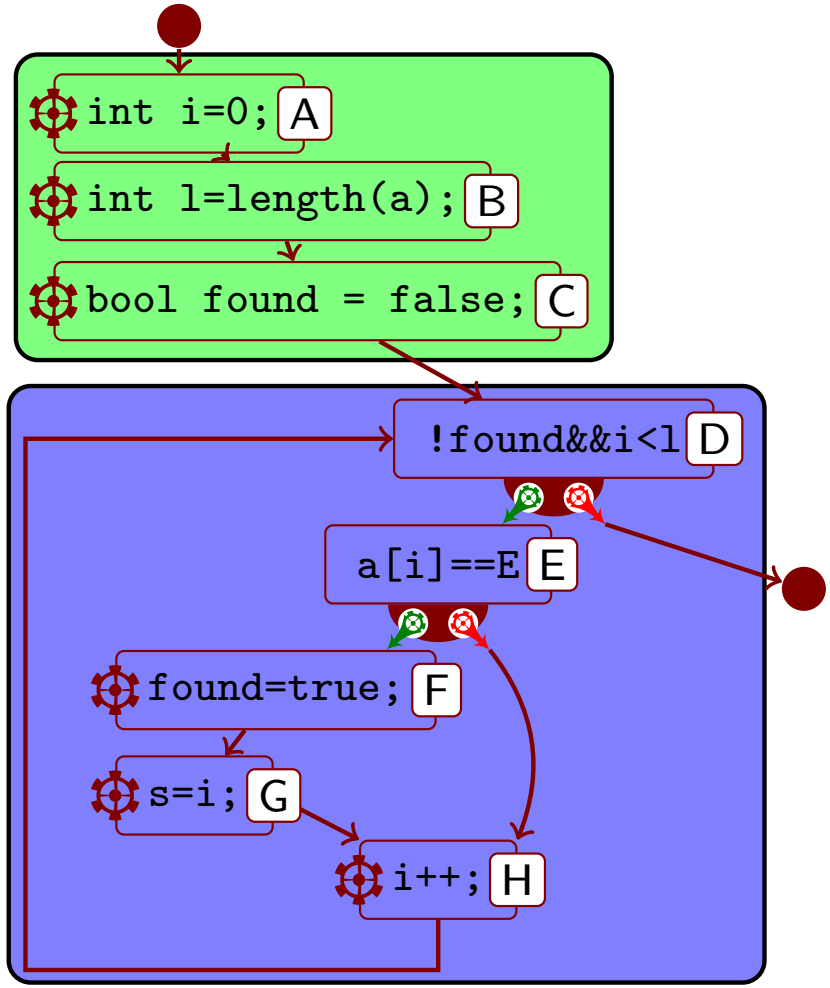
$$Q(m_0, m_f) \equiv$$
$$(m_f(found) \wedge m_0(a)[m_f(s)] = E)$$
$$\vee \ \Big(\neg m_f(found)$$
$$\wedge \forall j \in [\![m_0(i), m_0(l) - 1]\!], \ m_0(a)[j] \neq E\Big)$$



- On doit montrer qu'en sortie Q est vérifiée, c-a-d :
- $P(m_0, m_0) \wedge P(m_0, m_f) \wedge (m_f(found) \vee i \geqslant l) \implies Q(m_0, m_f)$
- Cette implication est triviale. Note : dans ce cas $P(m_0, m_0) \equiv true$, ce n'est pas très expressif

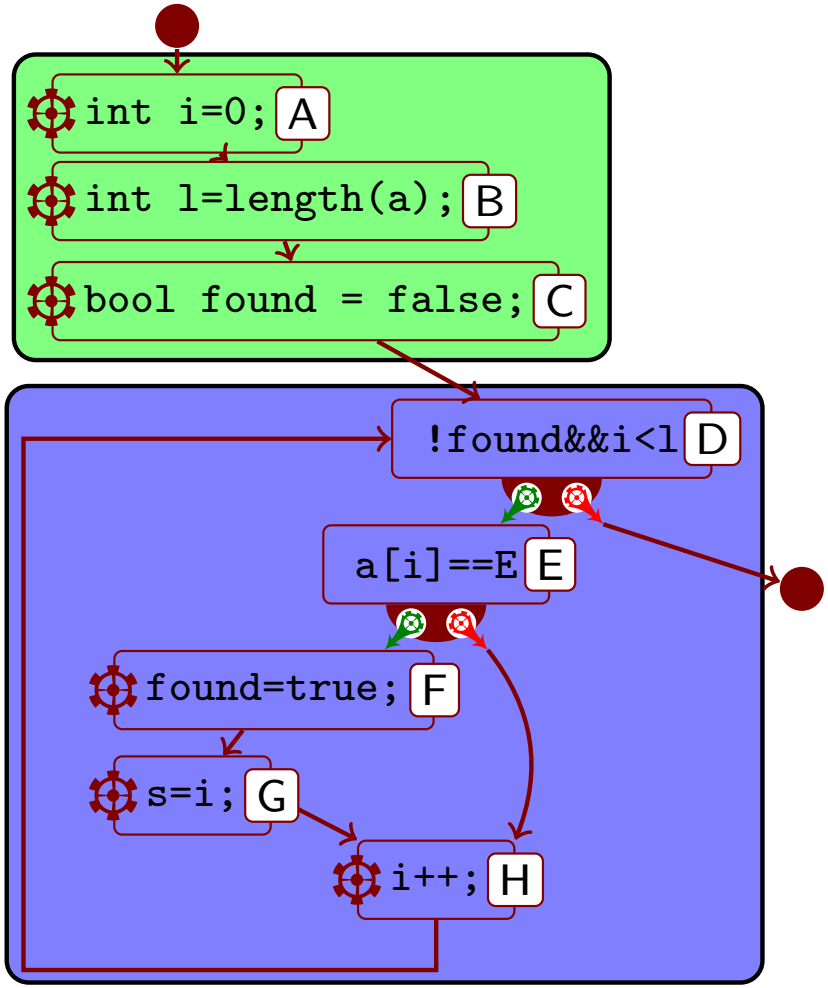
Exemple, fin

- Sur le bloc bleu nous avons la propriété Q, ajoutons en une sur le bloc vert



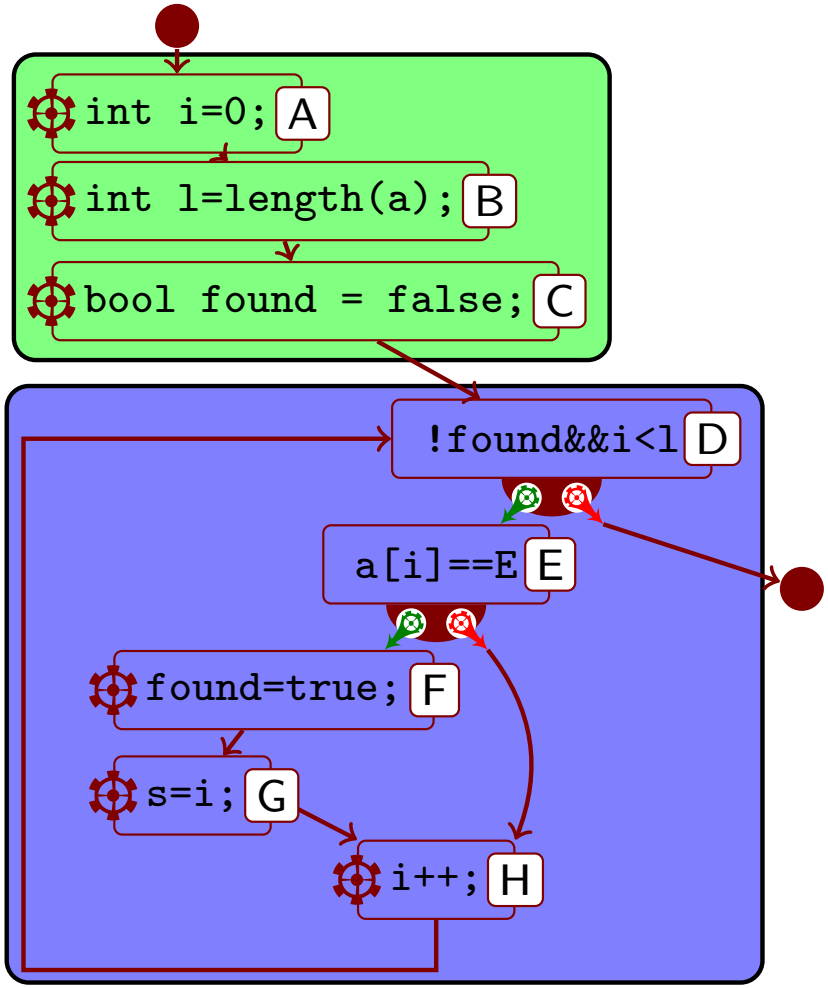
Exemple, fin

- Sur le bloc bleu nous avons la propriété Q, ajoutons en une sur le bloc vert
- $I(m_0, m_f) \equiv m_f(i) = 0 \wedge m_f(a) = m_0(a) \wedge m_f(found) = false$



Exemple, fin

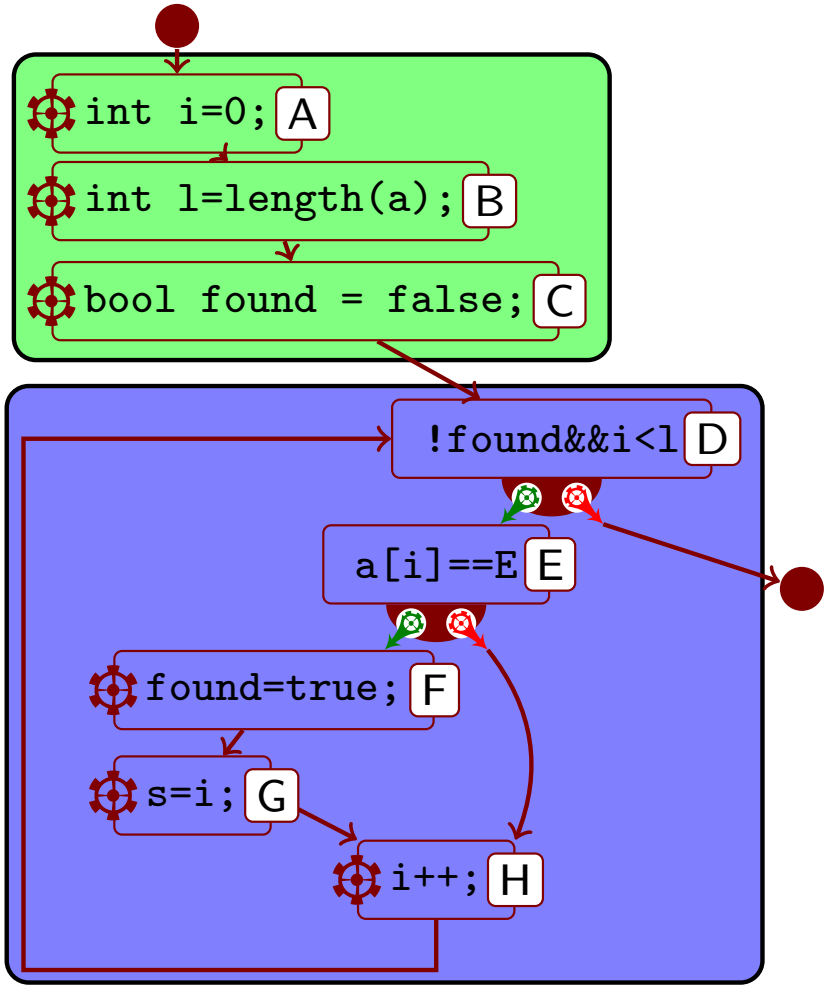
- Sur le bloc bleu nous avons la propriété Q, ajoutons en une sur le bloc vert
- $I(m_0, m_f) \equiv m_f(i) = 0 \wedge m_f(a) = m_0(a) \wedge m_f(found) = false$
- On peut combiner les propriétés, comme les blocs verts et bleus sont séquentiels :

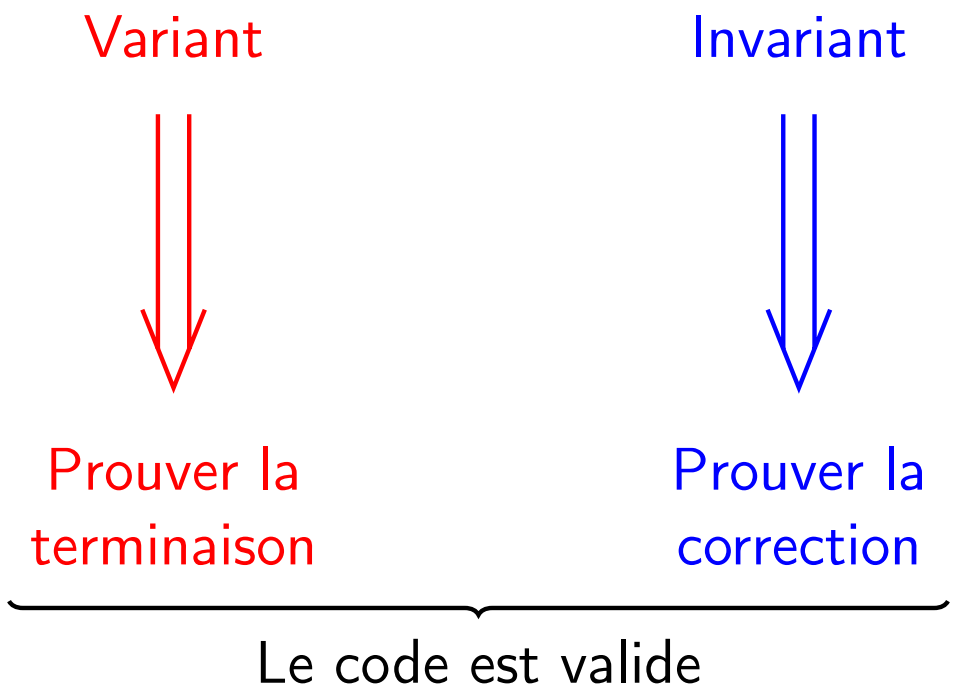


Exemple, fin

- Sur le bloc bleu nous avons la propriété Q, ajoutons en une sur le bloc vert
- $I(m_0, m_f) \equiv m_f(i) = 0 \wedge m_f(a) = m_0(a) \wedge m_f(found) = false$
- On peut combiner les propriétés, comme les blocs verts et bleus sont séquentiels :
- $I(m_0, m_1) \wedge Q(m_1, m_2) \implies F(m_0, m_2)$ avec

$$F(m_0, m_f) \equiv (m_f(found) \wedge m_0(a)[m_f(s)] = E) \vee (\neg m_f(found) \wedge \forall e \in m_0(a),\ e \neq E)$$

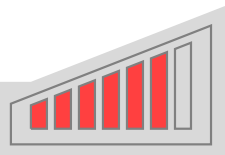



Variant
Invariant
Prouver la terminaison
Prouver la correction
Le code est valide

ATTENTION

- Si un exercice demande un invariant, l'invariant donné doit servir à justifier que la fonction est correcte. (Sinon il suffirait de répondre que l'invariant est "VRAI")
- Il n'y a pas de recette magique pour trouver un (in)variant
- Le passage invariant à "le programme est correct" n'est pas forcément évident.

- On a des blocs d'instructions c_i
- Pour un bloc c_i, on peut avoir une propriété $P_i(m_0, m_f)$ "Si l'état mémoire en entrant dans le bloc est m_0, et l'état mémoire en sortie est m_f, alors $P_i(m_0, m_f)$ est vrai".
- On suppose que chaque bloc termine (c'est soit évident, soit on l'a prouvé)
- Pour une séquence $c_1; c_2$ la terminaison est évidente et une propriété est une simplification de
  $P(m_0, m_2) \equiv \exists m_1 \in \mathbb{M},\ P_1(m_0, m_1) \wedge P_2(m_1, m_2)$
- Pour une boucle `while(cond(m))` c_1 ; il faut trouver le variant et l'invariant
  - On doit s'aider de la propriété P_1 pour trouver le variant et l'invariant I.
  - On aura la propriété $P(m_1, m_2) \equiv I(m_1, m_2) \wedge \neg \text{cond}(m_2)$



Notation

$\sigma[k/X]$ est l'état mémoire $\begin{array}{l} Y \mapsto \sigma(Y) \quad si\ X \neq Y \\ X \mapsto k \end{array}$

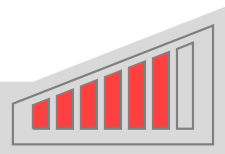
Si on note un état mémoire m_i, on notera a_i pour $m_i(a)$

Exemple simple 1

`x=f(x,y);` où f est un calcul mathématique
Terminaison : évidente
Correction : on a la propriété $P(m_0, m_1) \equiv m_1 = m_0[f(x_0, y_0)/x]$

`x=f(x,y);` où f est un calcul mathématique
Terminaison : évidente
Correction : on a la propriété $P(m_0, m_1) \equiv m_1 = m_0[f(x_0, y_0)/x]$

Note : si `f` est un appel à une fonction, il faudrait se servir d'une propriété existante Q sur cette fonction, on aurait alors la propriété $P(m_0, m_1) \equiv \exists m_t, Q(m_0, m_t) m_1 = m_t[f(x_0, y_0)/x]$, qui est plus digeste une fois simplifiée.



`while(N>0){r++;N--;}`

Terminaison : On prend le variant $\phi(m) = m(N)$ "Ok, mais là le résultat n'est pas dans $\mathbb{Z}$? Non, car la condition impose $N > 0$ en entrée de boucle"

Correction :

- Invariant $I(m_0, m_1) \equiv r_0 = 0 \wedge N_1 \geqslant 0 \wedge N_1 + r_1 = N_0$
- Propriété $P(m_0, m_1) \equiv r_0 = 0 \wedge m_1 = m_0[0/N][N_0/r_1]$

`while(N>0){r++;N--;}`

Terminaison : On prend le variant $\phi(m) = m(N)$ "Ok, mais là le résultat n'est pas dans $\mathbb{Z}$? Non, car la condition impose $N > 0$ en entrée de boucle"

Correction :

- Invariant $I(m_0, m_1) \equiv r_0 = 0 \wedge N_1 \geqslant 0 \wedge N_1 + r_1 = N_0$
- Propriété $P(m_0, m_1) \equiv r_0 = 0 \wedge m_1 = m_0[0/N][N_0/r_1]$

Correction (variante) :

- Invariant $I(m_0, m_1) \equiv N_1 \geqslant 0 \wedge N_1 + r_1 = N_0$
- Propriété $P(m_0, m_1) \equiv \wedge m_1 = m_0[0/N][N_0 + r0/r_1]$