IF107 - Preuve de programmes itératifs

Frédéric Herbreteau
frederic.herbreteau@bordeaux-inp.fr

10 décembre 2024

1. Spécification

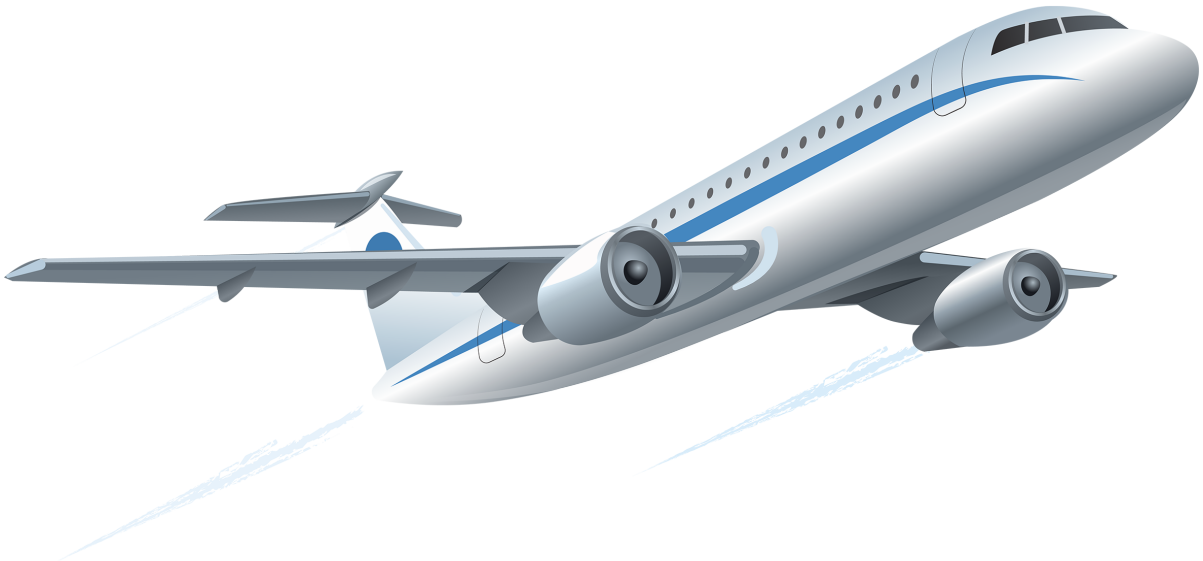
Preuve de programmes transformationnels

```
function factorial (n : int) returns (f : int)
    f := 1;
    i := 1;
    while i <= n:
        f := f * i
        i := i + 1
```

- On souhaite que la fonction `factorial` :
  - **termine** pour toute entrée `n`
  - retourne **correctement** la valeur `n`!
- Objectif : **techniques de preuve** pour la **terminaison** et la **correction**
- Langage de programmation : variables, affectation, **if**, **while**, for, **function**, etc.

Ce n'est pas le cas de tous les programmes

Système réactif :



On souhaite que le pilote automatique :

- **NE termine PAS**, quelle que soit l'entrée
- produise les **commandes correctes** en fonction des valeurs captées

Contexte du cours

Dans ce cours :

- Programmes **transformationnels**
- Hypothèses simplificatrices : entiers "parfaits", pas de problèmes mémoire, etc.
- Preuve de **terminaison**
- Preuve de **correction vis-à-vis d'une spécification**

Un exemple de spécification

```
function factorial (n : int) returns (f : int)
    f := 1;
    i := 1;
    while i <= n:
        f := f * i
        i := i + 1
```

PRECOND : `n` est un entier naturel

$$\mathtt{n} \in \mathbb{N}$$

POSTCOND : `f` est égale à la factorielle de `n`

$$\mathtt{f} = \mathtt{n}!$$

La spécification forme un contrat

```
function factorial (n : int) returns (f : int)
    f := 1;
    i := 1;
    while i <= n:
        f := f * i
        i := i + 1
```

La **spécification (PRECOND, POSTCOND)** forme un **contrat**

PRECOND :

POSTCOND :

Sous réserve que **l'utilisateur respecte la PRECOND**, la fonction **s'engage à satisfaire la POSTCOND**

Exemples de spécifications

function search(l : List , x : int) **returns** (found : **bool**)

PRECOND : *l* est triée

POSTCOND : $\texttt{found} = \mathit{true} \iff x \in l$

function abs(x : int) **returns** (a : int);

PRECOND : *true*

POSTCOND : $\texttt{a} = |\texttt{x}|$

function pow(x : real , int n) **returns** (p : real)

PRECOND : $\texttt{n} \in \mathbb{N}$

POSTCOND : $\texttt{p} = \texttt{x}^{\texttt{n}}$

Ces spécifications sont réalisées par de **multiples** programmes

Spécification et programmation

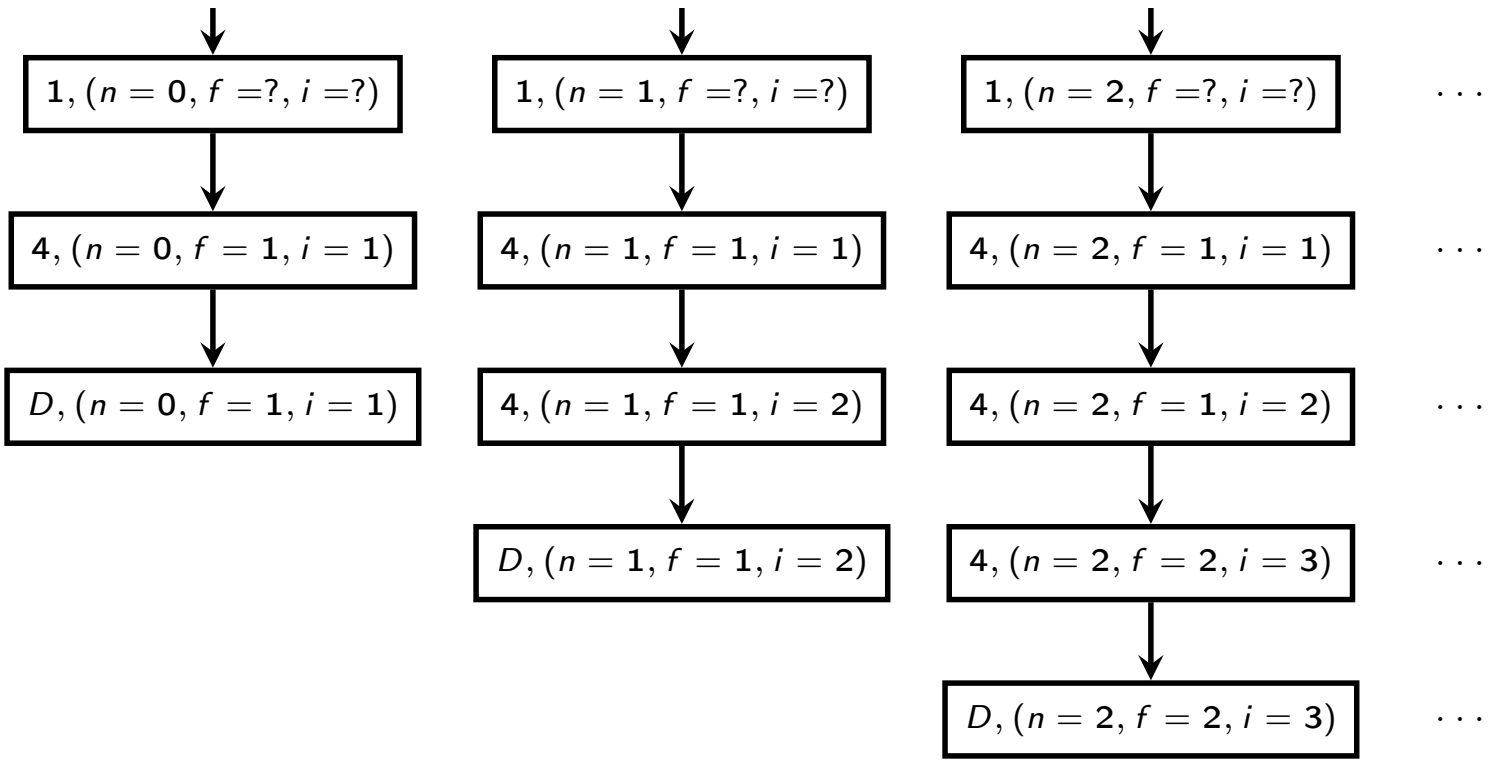
```
// Calcul de la factorielle
// PRECOND: n >= 0
// POSTCOND: f = n!
function factorial (n : int) returns (f : int)
    assert n >= 0;
    f := 1;
    i := 1;
    while i <= n:
        f := f * i
        i := i + 1
```

- **documenter la spécification (PRECOND, POSTCOND)** est utile pour rendre votre code (ré)utilisable
- la spécification est également utile pour générer des **cas de test** (vérification de la POSTCOND)
- la PRECOND n'est pas vérifiée par la fonction, mais on peut **détecter la violation**

2. Machine à états, terminaison, correction

Exemple : machine à états de `factorial` (1/2)

```
1  function factorial (n : int) returns (f : int)
2      f := 1;
3      i := 1;
4      while i <= n:
5          f := f * i
6          i := i + 1
```



Exemple : machine à états de `factorial` (2/2)

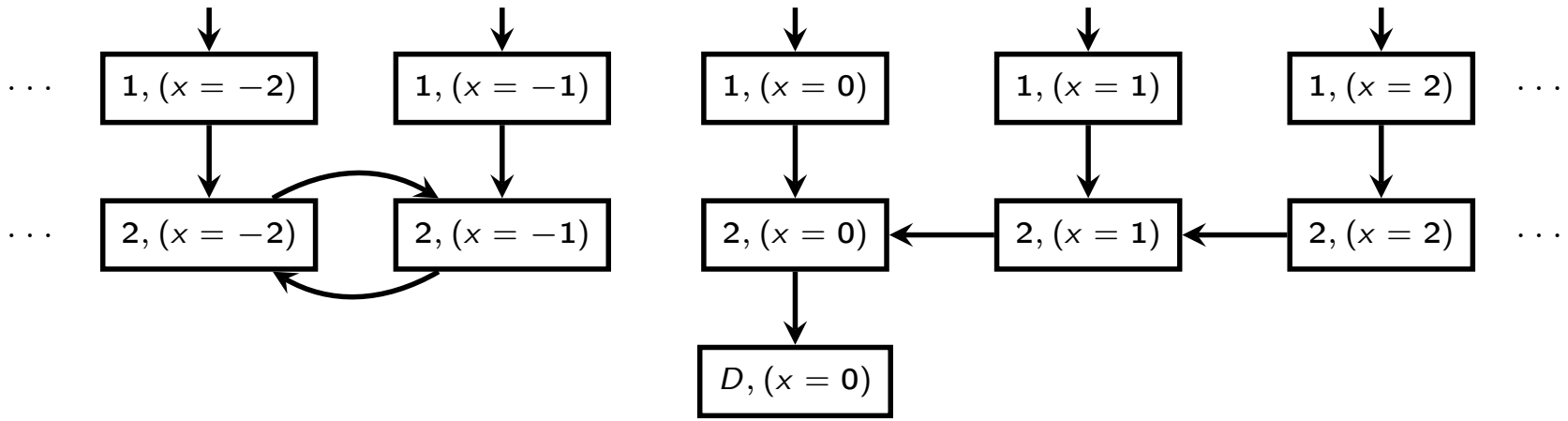
```
1  function factorial (n : int) returns (f : int)
2      f := 1;
3      i := 1;
4      while i <= n:
5          f := f * i
6          i := i + 1
```

$M_{fact} = (S, \text{Init}, \text{Next})$ où :

- $S = \{1, 4, D\} \times \mathbb{N} \times (\mathbb{N} \cup \{?\}) \times (\mathbb{N} \cup \{?\})$
- $\text{Init} = \{(1, n, ?, ?) \mid n \in \mathbb{N}\}$
- Next est l'ensemble des transitions :
  - $(1, n, f, i) \to (4, n, 1, 1)$
  - $(4, n, f, i) \to (4, n, f \times i, i + 1)$ si $i \leq n$
  - $(4, n, f, i) \to (D, n, f, i)$ si $i > n$

Exemple : machine à états de f

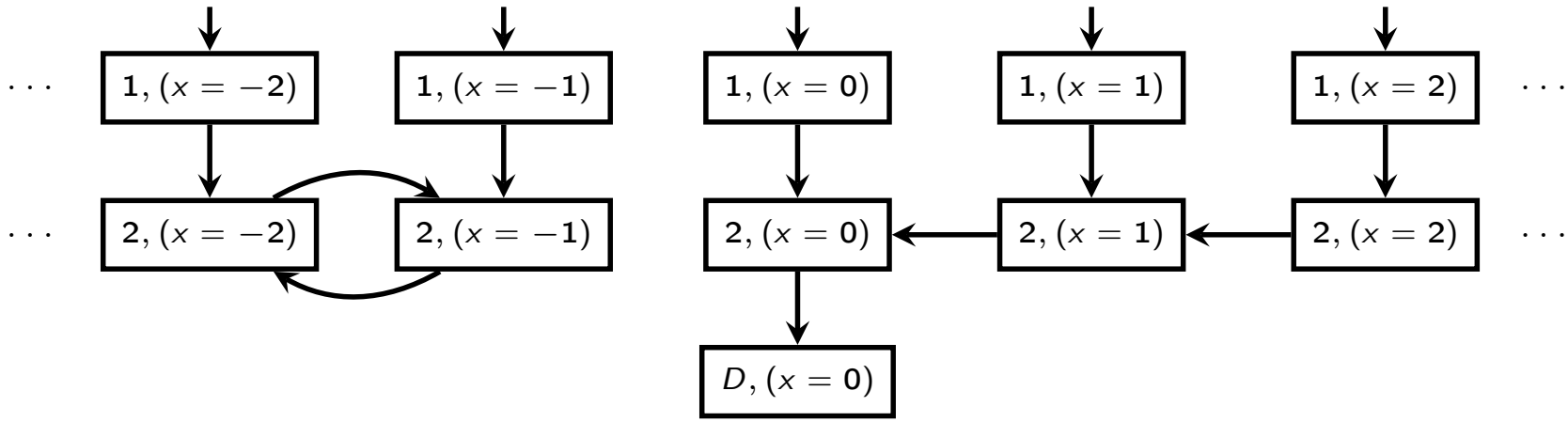
```
1  function f(x : int)
2      while x != 0:
3          if x < -1:
4              x = 0;
5          x = x - 1;
```



$M_f = (S, \text{Init}, \text{Next})$ où :

- $S = \{1, 2, D\} \times \mathbb{Z}$
- $\text{Init} = \{(1, x) \mid x \in \mathbb{Z}\}$
- $(1, x) \to (2, x)$
- $(2, x) \to (D, x)$ si $x = 0$
- $(2, x) \to (2, -1)$ si $x < -1$
- $(2, c) \to (2, x - 1)$ sinon

Rappel : exécutions finies et infinies



Définition

Une **exécution (ou trajectoire)** d'une machine à états $(S, \text{Init}, \text{Next})$ est une séquence d'états $s_0, s_1, s_2, \ldots$ possiblement infinie telle que $s_0 \in \text{Init}$ et $(s_i, s_{i+1}) \in \text{Next}$ pour tout i.

Exemple

Donner une exécution finie et une exécution infinie de la machine à états ci-dessus.

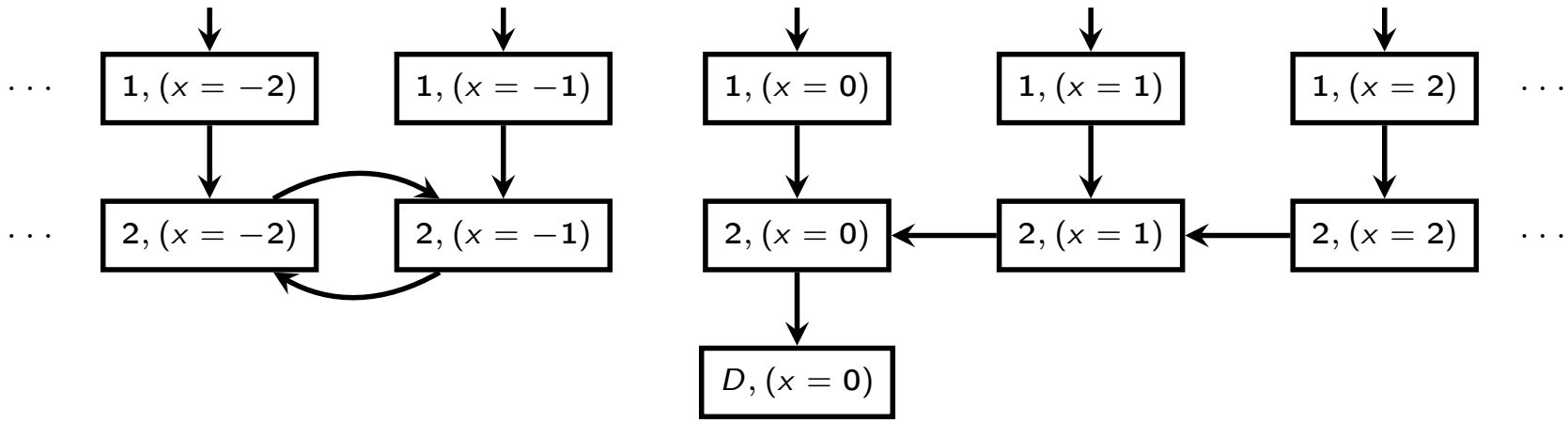
Terminaison de programme

Définition

Un programme **termine** si chacune des exécutions de sa machine à états est finie.

Exemple

La fonction `f` ne termine pas.



Théorème

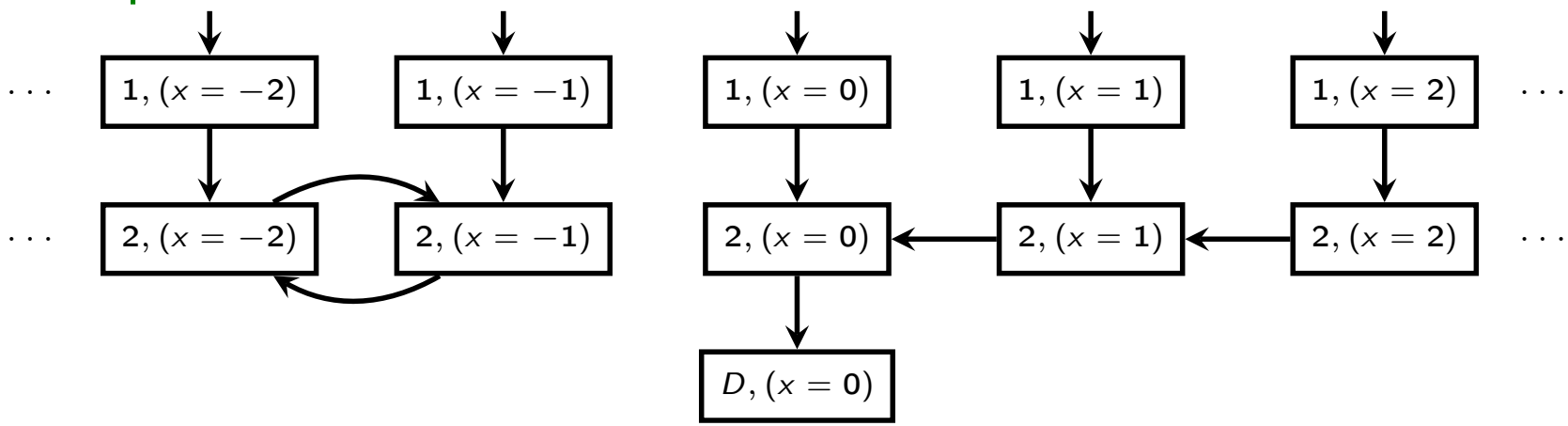
*Le problème qui consiste à décider si un programme P donné termine est **indécidable**.*

Correction des programmes

Définition

Un programme est **correct** pour une spécification (PRE, POST) si **pour chaque exécution finie** $s_0 s_1 \cdots s_n$, si PRE est vraie en s_0, alors POST est vraie en s_n.

Exemple



PRECOND : *true* POSTCOND : $x = 0$

Théorème

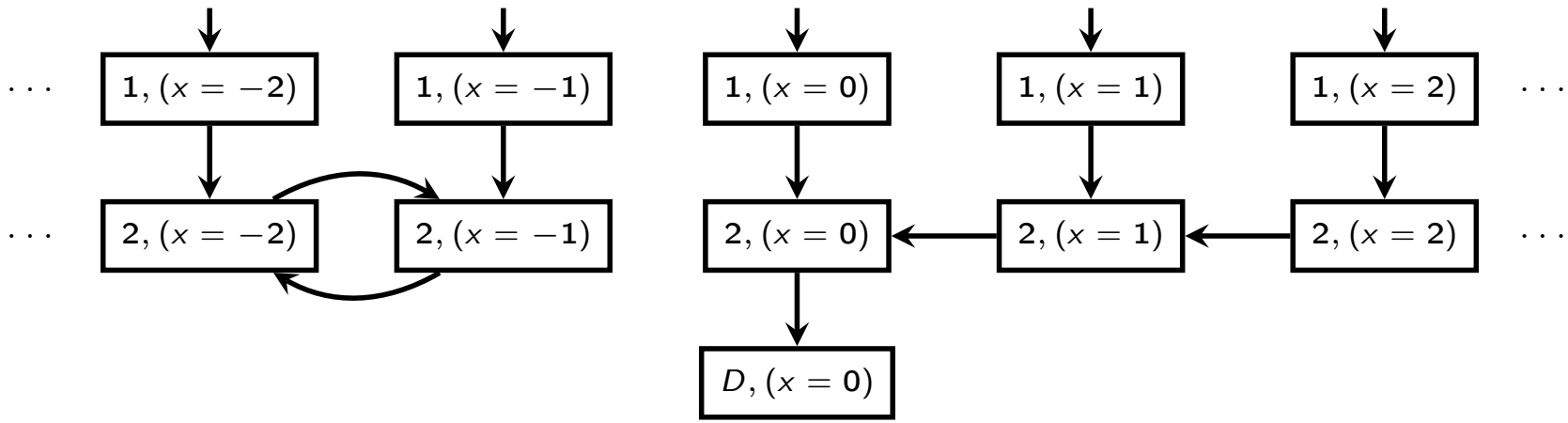
Le problème qui consiste à décider si un programme P est correct pour spécification (PRECOND, POSTCOND) est ***indécidable****.*

Sûreté et progrès

Exemple

Donner un programme qui est correct pour toute spécification

- la terminaison traduit une notion de **progrès**
- la correction assure que le résultat du calcul est **sûr**



- la terminaison et la correction étant **indécidables**, nous allons étudier des techniques qui ne sont pas automatisables, ou qui ne permettent pas de prouver tous les programmes.

3. Preuve de terminaison

Terminaison et convergence

Terminaison des fonctions `foo`, `bar` et `buz` ?

```
function foo (x : nat)
    while x > 0:
        x := x / 2   // quotient
```

```
function bar (x : real)
    while x > 0:
        x := x / 2
```

```
function buz (x : nat, y : nat)
    while x > y:
        x := x / 2   // quotient
```

Rappels : ensembles bien ordonnés

Définition

Un ensemble ordonné $(E, \preccurlyeq)$ est **bien ordonné** si toute partie non-vide de E admet un élément minimal pour $\preccurlyeq$.

Théorème

Toute suite strictement décroissante dans un ensemble bien ordonné est nécessairement finie.

Exemple

- $(\mathbb{N}, \leq)$ est bien ordonné
- l'ensemble List des listes chaînées muni de l'ordre $\leq_{\mathsf{List}}$ est bien ordonné
- l'ensemble Tree des arbres binaires ordonnés par $\leq_{\mathsf{Tree}}$ est bien ordonné

Exemple : terminaison de `foo`

```
1  function foo (x : nat)
2      while x > 0:
3          x := x / 2   // quotient
```

$M_{foo} = (S, \text{Init}, \text{Next})$ où :

- $S = \{1, 2, D\} \times \mathbb{N}$
- $\text{Init} = \{(1, x) \mid n \in \mathbb{N}\}$

- $(1, x) \to (2, x)$
- $(2, x) \to (2, x/2)$ si $x > 0$
- $(2, x) \to (D, x)$ si $x \leq 0$

Exemple

On se concentre sur la boucle : $(2, x) \to (2, x/2)$ si $x > 0$

- la valeur de `x` est **strictement décroissante** par $\to$:
- `x` est **à valeurs dans** $\mathbb{N}$:
- donc, toute exécution de M est finie et `foo` **termine**

Méthodologie : preuve de termnaison

On veut montrer que **toute exécution d'une machine à états M est finie (ou termine)** :

1. trouver un **variant**, c'est à dire une expression formée à partir des variables qui décrivent l'état de M
2. montrer que :
   - 2.1 le variant est **strictement décroissant** par exécution de M
   - 2.2 le variant est **à valeur dans un ensemble bien ordonné**

   $\rightarrow$ les valeurs du variant forment donc une **suite strictement decroissante dans un ensemble bien ordonné** le long de **toute exécution de M**

   $\rightarrow$ cette suite est donc nécessairement **finie** (théorème des ensembles bien ordonnés)
3. en conclure que **toute exécution de M est finie**

Exemple : terminaison de `factorial`

```
1  function factorial (n : int) returns (f : int)
2      f := 1;
3      i := 1;
4      while i <= n:
5          f := f * i
6          i := i + 1
```

PRECOND : $n \geq 0$ POSTCOND : $f = n!$

$$(4, n, f, i) \rightarrow (4, n, f \times i, i + 1) \qquad \text{si } i \leq n$$

Exemple

1. **variant** :
2. montrer que :
   - 2.1 le variant est **strictement décroissant** :
   - 2.2 le variant est **à valeur dans un ensemble bien ordonné** :
3. donc toute exécution de `factorial` est finie : la fonction `factorial` **termine** pour tout entrée `n`

Exemple : terminaison de search

```
1  function search (l : List, x : int) returns (found : bool)
2      l' := l
3      found := false
4      while l' ≠ ⟨⟩ and not(found):
5          if x = head(l'):
6              found := true
7          l' := tail(l')
```

PRECOND : *true* POSTCOND : $\texttt{found} = \mathit{true} \iff x \in l$

$(4, l, x, l', \mathit{false}) \to (4, l, x, \texttt{tail}(l'), \mathit{true})$ si $l' \neq \langle\rangle$ et $x = \texttt{head}(l')$
$(4, l, x, l', \mathit{false}) \to (4, l, x, \texttt{tail}(l'), \mathit{false})$ si $l' \neq \langle\rangle$ et $x \neq \texttt{head}(l')$

Exemple

1. **variant** :
2. montrer que :
   - 2.1 le variant est **strictement décroissant** :
   - 2.2 le variant est **à valeur dans un ensemble bien ordonné** :
3. donc la fonction `search` **termine** pour tout entrée `l`, `x`

4. Preuve de correction

Correction de factorial

```
1  function factorial (n : int) returns (f : int)
2      f := 1;
3      i := 1;
4      while i <= n:
5          f := f * i
6          i := i + 1
```

PRECOND : $n \geq 0$ POSTCOND : $f = n!$

$M_{fact} = (S, \text{Init}, \text{Next})$ où :

- $S = \{1, 4, D\} \times \mathbb{N} \times (\mathbb{N} \cup \{?\}) \times (\mathbb{N} \cup \{?\})$
- $\text{Init} = \{(1, n, ?, ?) \mid n \in \mathbb{N}\}$
- $(1, n, f, i) \rightarrow (4, n, 1, 1)$
- $(4, n, f, i) \rightarrow (4, n, f \times i, i + 1)$ si $i \leq n$
- $(4, n, f, i) \rightarrow (D, n, f, i)$ si $i > n$

Objectif : montrer que toute exécution finie depuis un état $(1, n, ?, ?) \in \text{Init}$ aboutit dans un état (D, n, f, i) avec $f = n!$

Rappels : invariant et invariant inductif

Définition

Un **invariant** d'une machine à états M est un prédicat $P(s)$ qui vrai pour tout état $s \in \text{Reach}(M)$.

Définition

Un **invariant inductif** est un prédicat $P(s)$ tel que :

- $P(s)$ est vrai pour tout etat $s \in \text{Init}$
- et, pour tout $(s, s') \in \text{Next}$, $P(s)$ implique $P(s')$.

Théorème

Si $P(s)$ est un invariant inductif, alors $P(s)$ est un invariant

Correction de factorial

```
1  function factorial (n : int) returns (f : int)
2      f := 1;
3      i := 1;
4      while i <= n:
5          f := f * i
6          i := i + 1
```

PRECOND : $n \geq 0$ POSTCOND : $f = n!$

NB : la POSTCOND $f = n!$ n'est généralement **pas un invariant**

Objectif : identifier un **invariant** I tel que $I \implies POSTCOND$ en sortie de boucle

Méthodologie : preuve de correction

```
while C:
        P
```

1. identifier un **invariant de boucle** I
2. montrer que :
   - 2.1 I est vrai **avant la boucle**
   - 2.2 I est **préservé par la boucle**

   $\rightarrow$ alors I est un invariant inductif de la boucle
3. en déduire que I est vrai **après la boucle**
4. montrer que "" I and not$(C) \implies$ *POSTCOND*"

Exemple : correction de `factorial`

```
1  function factorial (n : int) returns (f : int)
2      f := 1;
3      i := 1;
4      while i <= n:
5          f := f * i
6          i := i + 1
```

PRECOND : $n \geq 0$ POSTCOND : $f = n!$

$(1, n, ?, ?) \rightarrow (4, n, 1, 1)$

$(4, n, f, i) \rightarrow (4, n, f \times i, i + 1)$ si $i \leq n$

Exemple

1. **invariant de boucle** I
2. montrer que :
   - 2.1 I est vrai **avant la boucle**
   - 2.2 I est **préservé par la boucle**
3. I est donc vrai après la boucle
4. montrer que "I and not$(i \leq n) \implies f = n!$"

Exemple : correction de search

```
1  function search (l : List, x : int) returns (found : bool)
2      l' := l
3      found := false
4      while l' ≠ ⟨⟩ and not(found):
5          if x = head(l'):
6              found := true
7          l' := tail(l')
```

PRECOND : *true* POSTCOND : $\texttt{found} = \mathit{true} \iff x \in l$

$(1, l, x, ?, ?) \rightarrow (4, l, x, l, \mathit{false})$
$(4, l, x, l', \mathit{false}) \rightarrow (4, l, x, \texttt{tail}(l'), \mathit{true})$ si $l' \neq \langle\rangle$ et $x = \texttt{head}(l')$
$(4, l, x, l', \mathit{false}) \rightarrow (4, l, x, \texttt{tail}(l'), \mathit{false})$ si $l' \neq \langle\rangle$ et $x \neq \texttt{head}(l')$

Exemple

1. **invariant de boucle *I***
2. montrer que :
   - 2.1 *I* est vrai **avant la boucle**
   - 2.2 *I* est **préservé par la boucle**
3. *I* est donc vrai après la boucle
4. montrer que "*I* and not$(i \leq n) \implies$ *POSTCOND*"