

---

# Cours Modélisation et Vérification des Systèmes Informatiques (MVSI)

## Modélisation, spécification et vérification CM4

---

Dominique Méry  
Telecom Nancy, Université de Lorraine

---

**Année universitaire 2024-2025**

- ① Exemples de correction partielle (affectation simple)
- ② Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- ③ Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- ④ Vérification d'un contrat avec un solveur Z3
- ⑤ Conclusion et limites

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites

Le modèle relationnel  $M(P)$  pour le programme  $P$  annoté est donc défini comme suit :

$$M(P) \stackrel{def}{=} (Th(s, c), (pc, v), \text{LOCATIONS} \times \text{MEMORY}, \text{Init}(\ell, v), \{r_{\ell, \ell'} \mid \ell, \ell' \in \text{LOCATIONS} \wedge \ell \longrightarrow \ell'\}).$$

La définition de  $\text{Init}(x)$  est dépendante de la précondition de  $P$  :

$$\text{Init}(x) \stackrel{def}{=} .x = (\ell_0, v) \wedge \mathbf{pre}(P)(v).$$

### Conditions initiales

Les deux propriétés suivantes sont équivalentes :

- ▶  $\forall x_0 \in \text{VALS} : \text{Init}(x_0) \Rightarrow J(x_0, x_0)$
- ▶  $\forall v \in \text{MEMORY}. \mathbf{pre}(P)(v) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$

- ▶ Les relations  $r_i$  correspondent aux transitions satisfaisant  $\ell \longrightarrow \ell'$  et on associe à chaque  $r_i$  la relation  $r_{\ell,\ell'}$ 
  - ▶  $x \ r_{\ell,\ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell')$
- ▶  $J(x_0, x) \stackrel{def}{=} \exists v_0, \ell, v. (\ell \in \text{LOCATIONS} \wedge v_0, v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v_0, v))$
- ▶  $P_\ell(v_0, v) \stackrel{def}{=} \exists x_0, x. (x_0, x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

## Pas d'induction

Les deux propriétés suivantes sont équivalentes :

- ▶  $\forall i \in \{0, \dots, n\} : J(x_0, x) \wedge x \ r_i \ x' \Rightarrow J(x_0, x')$
- ▶  $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

## Pas d'induction (explication)

►  $J(x_0 n x) \wedge x \ r_{\ell, \ell'} \ x' \Rightarrow J(x_0, x')$

- ▶  $J(x_0, n x) \wedge x \ r_{\ell, \ell'} \ x' \Rightarrow J(x_0, x')$
- ▶  $x \ r_{\ell, \ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$

- ▶  $J(x_0 n x) \wedge x \ r_{\ell, \ell'} \ x' \Rightarrow J(x_0, x')$
- ▶  $x \ r_{\ell, \ell'} \ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶  $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$



- ▶  $J(x_0, x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- ▶  $x \text{ r}_{\ell, \ell'} x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶  $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- ▶  $P_\ell(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$



- ▶  $J(x_0 n x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- ▶  $x \text{ r}_{\ell, \ell'} x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶  $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- ▶  $P_\ell(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- ▶  $J(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- ▶  $J(x_0, x') \equiv pc = \ell' \wedge P_{\ell'}(v_0, v')$

- ▶  $J(x_0 n x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- ▶  $x \text{ r}_{\ell, \ell'} x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶  $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- ▶  $P_\ell(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- ▶  $J(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- ▶  $J(x_0, x') \equiv pc = \ell' \wedge P_{\ell'}(v_0, v')$
- ▶  $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell') \Rightarrow pc = \ell' \wedge P_{\ell'}(v_0, v')$

- $J(x_0 n x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- $x \text{ r}_{\ell, \ell'} x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $J(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $J(x_0, x') \equiv pc = \ell' \wedge P_{\ell'}(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \wedge P_{\ell'}(v_0, v'))$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \text{ (Tautologie)})$

- $J(x_0 n x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- $x \text{ r}_{\ell, \ell'} x' \stackrel{\text{def}}{=} (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- $I(x) \stackrel{\text{def}}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{\text{def}}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $J(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $J(x_0, x') \equiv pc = \ell' \wedge P_{\ell'}(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \wedge P_{\ell'}(v_0, v'))$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \text{ (Tautologie)})$
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge \text{cond}_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow P_{\ell'}(v_0, v'))$

- ▶  $J(x_0 n x) \wedge x \text{ r}_{\ell, \ell'} x' \Rightarrow J(x_0, x')$
- ▶  $x \text{ r}_{\ell, \ell'} x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell')$
- ▶  $I(x) \stackrel{def}{=} \exists \ell, v. (\ell \in \text{LOCATIONS} \wedge v \in \text{MEMORY} \wedge x = (\ell, v) \wedge P_{\ell}(v))$
- ▶  $P_{\ell}(v_0, v) \stackrel{def}{=} \exists x. (x \in \text{VALS} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- ▶  $J(x_0 n x) \equiv pc = \ell \wedge P_{\ell}(v_0, v)$
- ▶  $J(x_0, x') \equiv pc = \ell' \wedge P_{\ell'}(v_0, v')$
- ▶  $pc = \ell \wedge P_{\ell}(v_0, v) \wedge (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \wedge P_{\ell'}(v_0, v'))$
- ▶  $pc = \ell \wedge P_{\ell}(v_0, v) \wedge (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \text{ (Tautologie)})$
- ▶  $pc = \ell \wedge P_{\ell}(v) \wedge (pc = \ell \wedge cond_{\ell, \ell'}(v) \wedge \wedge v' = f_{\ell, \ell'}(v) \wedge pc' = \ell' \Rightarrow P_{\ell'}(v_0, v'))$
- ▶  $P_{\ell}(v_0, v) \wedge cond_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

- ▶  $J(x_0, x) \stackrel{def}{=} \exists \ell, v, v_0. (\ell \in \text{LOCATIONS} \wedge v, v_0 \in \text{MEMORY} \wedge x = (\ell, v) \text{wedge} x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v))$
- ▶  $P_\ell(v_0, v) \stackrel{def}{=} \exists x. (x, x_0 \in \text{VALS} \wedge x = (\ell, v) \text{wedge} x_0 = (\ell_0, v_0) \wedge J(x_0), x)$
- ▶  $J(x_0, x) \Rightarrow A(x_0, x)$
- ▶  $\exists \ell, v, v_0. (\ell \in \text{LOCATIONS} \wedge v, v_0 \in \text{MEMORY} \wedge x = (\ell, v) \text{wedge} x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow A(x_0, x)$
- ▶  $\forall \ell, v, v_0. (\ell \in \text{LOCATIONS} \wedge v, v_0 \in \text{MEMORY} \wedge x = (\ell, v) \text{wedge} x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow A(x_0, x)$
- ▶  $\forall \ell \in \text{LOCATIONS}, v, v_0 \in \text{MEMORY}. P_\ell(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v)$

## Conclusion

Les deux propriétés suivantes sont équivalentes :

- ▶  $J(x_0, x) \Rightarrow A(x_0, x)$
- ▶  $\forall \ell \in \text{LOCATIONS}, v, v_0 \in \text{MEMORY}. P_\ell(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v)$



Les conditions de vérification suivantes sont équivalentes :

►  $\forall x_0, x, x' \in \text{LOCATIONS} \times \text{MEMORY}$  :

$$\left\{ \begin{array}{l} (1) \text{ Init}(x_0) \Rightarrow J(x_0, x_0) \\ (2) J(x_0, x) \Rightarrow A(x_0, x) \\ (3) \forall i \in \{0, \dots, n\} : J(x_0, x) \wedge x \text{ r}_i x' \Rightarrow J(x_0, x') \end{array} \right.$$

►  $\forall v_0, v, v' \in \text{MEMORY}$  :

$$\left\{ \begin{array}{l} (1) \text{ pre(P)}(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v) \\ (2) \forall \ell \in \text{LOCATIONS}. P_{\ell}(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v) \\ (3) \forall \ell, \ell' \in \text{LOCATIONS} : \\ \ell \longrightarrow \ell' \Rightarrow P_{\ell}(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v') \end{array} \right.$$

- ▶  $A(\ell, v)$  est l'énoncé de la propriété de sûreté à vérifier.

## Méthode relationnelle de correction de propriétés de sûreté

Soit  $A(\ell_0, v_0, \ell, v)$  une propriété d'un programme  $P$ . Soit une famille d'annotations famille de propriétés  $\{P_\ell(v_0, v) : \ell \in \text{LOCATIONS}\}$  pour ce programme. Si les conditions suivantes sont vérifiées :  
alors  $A(\ell_0, v_0, \ell, v)$  est une propriété de sûreté pour le programme  $P$ .

### Definition Condition de vérification

L'expression  $P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$  où  $\ell, \ell'$  sont deux étiquettes liées par la relation  $\longrightarrow$ , est appelée une condition de vérification.

## Floyd and Hoare

- ▶  $\forall v_0, v, v' \in \text{MEMORY}. \forall \ell, \ell' \in \text{LOCATIONS}. \ell \longrightarrow \ell' \Rightarrow$   
 $P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$  est équivalent à  
 $\forall \ell, \ell' \in \text{LOCATIONS}. \ell \longrightarrow \ell' \Rightarrow \forall v' \in$   
 $\text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell, \ell'}(v))$
- ▶  $\forall v_0, v, v' \in \text{MEMORY}. \forall \ell, \ell' \in \text{LOCATIONS}. \ell \longrightarrow \ell' \Rightarrow$   
 $P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$  est équivalent à  
 $\forall \ell, \ell' \in \text{LOCATIONS}. \ell \longrightarrow \ell' \Rightarrow \forall v' \in$   
 $\text{MEMORY}. (\exists v \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v)) \Rightarrow$   
 $P_{\ell'}(v_0, v')$

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

$$\blacktriangleright \forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$$

$$\begin{array}{l} \ell : P_\ell(v_0, v) \\ V := f_{\ell, \ell'}(V) \\ \ell' : P_{\ell'}(v_0, v) \end{array}$$

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

- ▶  $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶  $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{TRUE} \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$$\begin{aligned} \ell &: P_\ell(v_0, v) \\ V &:= f_{\ell, \ell'}(V) \\ \ell' &: P_{\ell'}(v_0, v) \end{aligned}$$

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

- ▶  $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶  $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{TRUE} \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶  $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

$$\begin{aligned} \ell &: P_\ell(v_0, v) \\ V &:= f_{\ell, \ell'}(V) \\ \ell' &: P_{\ell'}(v_0, v) \end{aligned}$$

Nous pouvons resumer les deux formes possibles de l'affectation suivante :

- ▶  $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶  $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge \text{TRUE} \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶  $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- ▶  $\forall v \in \text{MEMORY}. P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell, \ell'}(v))$   
(l'axiomatique de Hoare).

$$\begin{aligned} \ell &: P_\ell(v_0, v) \\ V &:= f_{\ell, \ell'}(V) \\ \ell' &: P_{\ell'}(v_0, v) \end{aligned}$$





```
 $\ell_1 : P_{\ell_1}(v_0, v)$   
WHILE  $B(v)$  DO  
   $\ell_2 : P_{\ell_2}(v_0, v)$   
  ...  
   $\ell_3 : P_{\ell_3}(v_0, v)$   
END  
 $\ell_4 : P_{\ell_4}(v_0, v)$ 
```

Pour la structure d'itération, les conditions de vérification sont les suivantes :

- ▶  $P_{\ell_1}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- ▶  $P_{\ell_1}(v_0, v) \wedge \neg B(v) \Rightarrow P_{\ell_4}(v_0, v)$
- ▶  $P_{\ell_3}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- ▶  $P_{\ell_3}(v_0, v) \wedge \neg B(v) \Rightarrow P_{\ell_4}(v_0, v)$

```
 $\ell_1 : P_{\ell_1}(v_0, v)$   
IF  $B(v)$  THEN  
   $\ell_2 : P_{\ell_2}(v_0, v)$   
  ...  
   $\ell_3 : P_{\ell_3}(v_0, v)$   
ELSE  
   $m_2 : P_{\ell_2}(v_0, v)$   
  ...  
   $m_3 : P_{\ell_3}(v_0, v)$   
FI  
 $\ell_4 : P_{\ell_4}(v_0, v)$ 
```

Pour la structure de conditionnelle, les conditions suivantes :

- ▶  $P_{\ell_1}(v_0, v) \wedge B(v) \Rightarrow P_{\ell_2}(v_0, v)$
- ▶  $P_{\ell_3}(v_0, v) \Rightarrow P_{\ell_4}(v_0, v)$
- ▶  $P_{\ell_1}(v_0, v) \wedge \neg B(v) \Rightarrow P_{m_2}(v_0, v)$
- ▶  $P_{m_3}(v_0, v) \Rightarrow P_{\ell_4}(v_0, v)$



Si les conditions suivantes sont vérifiées :

- ▶  $\forall v_0, v \in \text{MEMORY} : \mathbf{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- ▶  $\forall v_0, v \in \text{MEMORY} : P_{\ell_f}(v_0, v) \Rightarrow \mathbf{post}(P)(v_0, v)$
- ▶  $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in \text{MEMORY}. (P_{\ell}(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$ ,

alors le programme  $P$  est partiellement correct par rapport à  $\mathbf{pre}(P)(v_0)$  et  $\mathbf{post}(P)(v_0, v)$ .

- ▶ La correction partielle indique que si le programme termine normalement, alors la postcondition est vérifiée par les variables courantes.
- ▶ La sémantique du contrat est donc assez simple à donner :

- ▶  $\text{pre}(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow \text{post}(v_0, v_f)$   
(expression de la correction partielle)
- ▶  $pc_0 = \ell_0 \wedge \text{pre}(v_0) \wedge (pc_0, v_0) \xrightarrow{\text{Next}^*} (pc, v) \wedge pc = \ell_f \Rightarrow \text{post}(v_0, v_f)$   
(big-step semantics et small-step semantics equivalence)
- ▶  $pc_0 = \ell_0 \wedge \text{pre}(v_0) \wedge (pc_0, v_0) \xrightarrow{\text{Next}^*} (pc, v) \Rightarrow (pc = \ell_f \Rightarrow \text{post}(v_0, v_f))$   
(implication and conjunction property)
- ▶  $\text{Init}(x_0) \wedge x_0 \xrightarrow{\text{Next}^*} x \Rightarrow \text{PC}(x_0, x)$

$$\begin{aligned} \text{Init}(x_0) &\stackrel{\text{def}}{=} pc_0 = \ell_0 \wedge \text{pre}(v_0) \\ x_0 &\stackrel{\text{def}}{=} (\ell_0, v_0) \text{ and } x \stackrel{\text{def}}{=} (pc, v) \end{aligned}$$

$$\text{PC}(x_0, x) \stackrel{\text{def}}{=} x_0 = (\ell_0, v_0) \wedge x = (\ell_f, v) \Rightarrow (pc = \ell_f \Rightarrow \text{post}(v_0, v))$$



**Partial correctness is a safety property and the relational method for safety properties is applied.**

Un programme  $P$  *remplit* un contrat  $(pre, post)$  :

- ▶  $P$  transforme une variable  $v$  à partir d'une valeur initiale  $v_0$  et produisant une valeur finale  $v_f$  :  $v_0 \xrightarrow{P} v_f$
- ▶  $v_0$  satisfait  $pre$  :  $pre(v_0)$  and  $v_f$  satisfait  $post$  :  $post(v_0, v_f)$
- ▶  $pre(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow post(v_0, v_f)$

requires  $pre(v_0)$

ensures  $post(v_0, v_f)$

variables  $V$

begin

$0 : P_0(v_0, v)$

instruction<sub>0</sub>

...

$i : P_i(v_0, v)$

...

instruction <sub>$f-1$</sub>

$f : P_f(v_0, v)$

end

▶  $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$

▶  $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$

▶ Pour toute paire d'étiquettes  $\ell, \ell'$  telle que  $\ell \longrightarrow \ell'$ , on vérifie que, pour toutes valeurs

$v, v' \in \text{MEMORY}$

$$\left( \begin{array}{c} P_\ell(v_0, v) \\ \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$

### An Early Program Proof by Alan Turing

Turing, A. M. 1949. "Checking a Large Routine." In Report of a Conference on High Speed Automatic Calculating Machines, Univ. Math. Lab., Cambridge, pp. 67-69.

- ▶ Turing se pose une question fondamentale de la correction des routines ou programmes en 1949.
- ▶ Il s'agit sans doute (Jones!) de la méthode d'annotation et d'induction sur les programmes qui sera finalisée par Floyd en 1967.

### Méthode de Floyd

- ▶ Au point 0,  $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- ▶ Annotations : au point  $i$ , l'assertion  $P_i(x_0, x)$  est vraie.
- ▶ Au point final  $f$ ,  $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$

- ▶ La transition à exécuter est celle allant de  $\ell$  à  $\ell'$  et caractérisée par la condition ou garde  $cond_{\ell,\ell'}(v)$  sur  $v$  et une transformation de la variable  $v$ ,  $v' = f_{\ell,\ell'}(v)$ .
- ▶ Une condition d'absence d'erreur est définie par  $\mathbf{DOM}(\ell, \ell')(v)$  pour la transition considérée.  $\mathbf{DOM}(\ell, \ell')(v)$  signifie que la transition  $\ell \longrightarrow \ell'$  est possible et ne conduit pas à une erreur.
- ▶ Une erreur est un débordement arithmétique, une référence à un élément de tableau qui n'existe pas, une référence à un pointeur nul, ...

### exemple

- 1 La transition correspond à une affectation de la forme  $x := x+y$  ou  $y := x+y$  :  
$$\mathbf{DOM}(x+y)(x, y) \stackrel{def}{=} \mathbf{DOM}(x)(x, y) \wedge \mathbf{DOM}(y)(x, y) \wedge x+y \in int$$
- 2 La transition correspond à une affectation de la forme  $x := x+1$  ou  $y := x+1$  :  
$$\mathbf{DOM}(x+1)(x, y) \stackrel{def}{=} \mathbf{DOM}(x)(x, y) \wedge x+2 \in int$$



### Définition RTE

L'absence d'erreurs à l'exécution vise à établir qu'un programme  $P$  ne va pas produire des erreurs durant son exécution par rapport à sa précondition et à sa postcondition.

- ▶ la spécification des données de  $P$  **pre**( $P$ )( $v$ )
- ▶ la spécification des résultats de  $P$  **post**( $P$ )( $v_0, v$ )
- ▶ une famille d'annotations de propriétés  $\{P_\ell(v) : \ell \in \text{LOCATIONS}\}$  pour ce programme.
- ▶ une propriété de sûreté définissant l'absence d'erreurs à l'exécution :

$$\bigwedge_{\ell \in \text{LOCATIONS} - \{\text{output}\}, n \in \text{LOCATIONS}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$$

.....

#### ☒ Definition

Le programme  $P$  ne produira pas d'erreurs à l'exécution par rapport à **pre**( $P$ )( $v$ ) et **post**( $P$ )( $v_0, v$ ), si la propriété

$\bigwedge_{\ell \in \text{LOCATIONS} - \{\text{output}\}, n \in \text{LOCATIONS}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$  est une propriété de sûreté pour ce programme.

### RTE = Run Time Error

Si les conditions suivantes sont vérifiées :

- ▶  $\forall v_0, v \in \text{MEMORY} : \mathbf{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$
- ▶  $\forall m \in \text{LOCATIONS} - \{\ell_f\}, n \in \text{LOCATIONS}, \forall v_0, v, v' \in \text{MEMORY} : m \longrightarrow n : P_m(v_0, v) \Rightarrow \mathbf{DOM}(m, n)(v)$
- ▶  $\forall \ell, \ell' \in \text{LOCATIONS} : \ell \longrightarrow \ell' : \forall v_0, v, v' \in \text{MEMORY}. (P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$ ,

alors le programme  $P$  ne produira pas d'erreurs à l'exécution par rapport à  $\mathbf{pre}(P)(v_0)$  et  $\mathbf{post}(P)(v_0, v)$ .

- ▶ On doit d'abord vérifier la correction partielle puis renforcer les assertions de la correction partielle par des conditions de domaine.
- ▶ On peut donc en déduire un contrat qui intègre aussi la vérification de l'absence d'erreurs à l'exécution.

Un programme  $P$  *remplit* un contrat (pre,post) :

- ▶  $P$  transforme une variable  $v$  à partir d'une valeur initiale  $v_0$  et produisant une valeur finale  $v_f$  :  $v_0 \xrightarrow{P} v_f$
- ▶  $v_0$  satisfait pre :  $\text{pre}(v_0)$  and  $v_f$  satisfait post :  $\text{post}(v_0, v_f)$
- ▶  $\text{pre}(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow \text{post}(v_0, v_f)$
- ▶  $\mathbb{D}$  est le domaine RTE de  $V$

requires  $\text{pre}(v_0)$   
 ensures  $\text{post}(v_0, v_f)$   
 variables  $V$

```
begin
  0 :  $P_0(v_0, v)$ 
  instruction0
  ...
  i :  $P_i(v_0, v)$ 
  ...
  instructionf-1
  f :  $P_f(v_0, v)$ 
end
```

- ▶  $\text{pre}(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- ▶  $\text{pre}(v_0) \wedge P_f(v_0, v) \Rightarrow \text{post}(v_0, v)$
- ▶ Pour toute paire d'étiquettes  $\ell, \ell'$  telle que  $\ell \longrightarrow \ell'$ , on vérifie que, pour toutes valeurs  $v, v' \in \text{MEMORY}$ 

$$\left( \begin{array}{c} P_\ell(v_0, v) \\ \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \\ \Rightarrow P_{\ell'}(v_0, v') \end{array} \right),$$
- ▶  $\forall m \in \text{LOCATIONS} - \{\ell_f\}, n \in \text{LOCATIONS}, \forall v_0, v, v' \in \text{MEMORY} :$   
 $m \longrightarrow n : P_m(v_0, v) \Rightarrow \text{DOM}(m, n)(v)$

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites

$$v = v_0 \wedge pre(v_0) \wedge v_f = f(v) \Rightarrow post(v_0, v_f) \quad (I)$$

```
requires  $pre(v_0)$   
ensures  $post(v_0, v_f)$   
variables  $V$   
[ begin  
   $0 : P_0(v_0, v)$   
   $V := f(V)$   
   $f : P_f(v_0, v)$   
end
```

Liste des conditions à vérifier pour prouver (I)

- ▶  $v = v_0 \wedge pre(v_0) \Rightarrow P_0(v_0, v)$
- ▶  $pre(v_0) \wedge P_0(v_0, v) \wedge v' = f(v) \Rightarrow P_f(v_0, v')$
- ▶  $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- ▶ (I) et (II) sont équivalents et (II) est la définition de l'invariance de  $A(x_0, x) \stackrel{def}{=} (x = (f, v) \Rightarrow post(v_0, v))$ .

$$x_0 = (0, v_0) \wedge pre(v_0) \wedge x_0 \xrightarrow{[V := f(V)]} x \Rightarrow (x = (f, v) \Rightarrow post(v_0, v)) \quad (II)$$

$$v = v_0 \wedge pre(v_0) \wedge v_f = g(f(v)) \Rightarrow post(v_0, v_f) \text{ (I)}$$

requires  $pre(v_0)$   
ensures  $post(v_0, v_f)$   
variables  $V$

```
begin  
  0 :  $P_0(v_0, v)$   
   $V := f(V)$   
  1 :  $P_1(v_0, v)$   
   $V := g(V)$   
   $f : P_f(v_0, v)$   
end
```

Liste des conditions à vérifier pour prouver  
(I)

- ▶  $v = v_0 \wedge pre(v_0) \Rightarrow P_0(v_0, v)$
- ▶  $pre(v_0) \wedge P_0(v_0, v) \wedge v' = f(v) \Rightarrow P_1(v_0, v')$
- ▶  $pre(v_0) \wedge P_1(v_0, v) \wedge v' = g(v) \Rightarrow P_f(v_0, v')$
- ▶  $pre(v_0) \wedge P_f(v_0, v) \Rightarrow post(v_0, v)$
- ▶ (I) et (II) sont équivalents et (II) est la définition de l'invariance de  $A(x_0, x) \stackrel{def}{=} (x = (f, v) \Rightarrow post(v_0, v))$ .

$$x_0 = (0, v_0) \wedge pre(v_0) \wedge x_0 \xrightarrow{[V := f(V); V := g(V)]} x \Rightarrow (x = (f, v) \Rightarrow post(v_0, v)) \text{ (II)}$$

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites

## Méthode de correction de propriétés de sûreté

Soit  $A(\ell_0, v_0, \ell, v)$  une propriété d'un programme  $P$ . Soit une famille d'annotations famille de propriétés  $\{P_\ell(v_0, v) : \ell \in \text{LOCATIONS}\}$  pour ce programme. Si les conditions suivantes sont vérifiées :

$\forall v_0, v, v' \in \text{MEMORY} :$

$$\left\{ \begin{array}{l} (1) \text{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v) \\ (2) \forall \ell \in \text{LOCATIONS}. P_\ell(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v) \\ (3) \forall \ell, \ell' \in \text{LOCATIONS} : \\ \quad \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v') \end{array} \right. ,$$

alors  $A(\ell_0, v_0, \ell, v)$  est une propriété de sûreté pour le programme  $P$ .

- 1 Définir la précondition  $\text{pre}(P)(v_0, v)$
- 2 Annoter le programme avec des prédicats  $P_\ell(v_0, v)$  où  $\ell \in \text{LOCATIONS}$
- 3 Vérifier que  $\text{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_\ell(v)$  où  $\ell \in \text{INPUTS}$  (ensemble des points d'entrée.
- 4 Vérifier que  $P_\ell(v_0, v) \Rightarrow A(\ell, v)$  où  $\ell \in \text{LOCATIONS}$
- 5 Pour chaque paire de points de contrôle  $(\ell, \ell')$  telle que  $\ell \longrightarrow \ell'$  (successifs), vérifier que  $(P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$ .



- 1 Vérifier que  $\mathbf{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$  où  $\ell \in \text{INPUTS}$  (ensemble des points d'entrée).
- 2 Vérifier que  $P_\ell(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v)$  où  $\ell \in \text{LOCATIONS}$
- 3 Pour chaque paire de points de contrôle  $(\ell, \ell')$  telle que  $\ell \longrightarrow \ell'$  (successifs), vérifier que  $(P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$ .

- 1 Vérifier que  $\mathbf{pre}(P)(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$  où  $\ell \in \text{INPUTS}$  (ensemble des points d'entrée).
- 2 Vérifier que  $P_\ell(v_0, v) \Rightarrow A(\ell_0, v_0, \ell, v)$  où  $\ell \in \text{LOCATIONS}$
- 3 Pour chaque paire de points de contrôle  $(\ell, \ell')$  telle que  $\ell \longrightarrow \ell'$  (successifs), vérifier que  $(P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$ .

### Exemples de propriétés de sûreté

- ▶ Correction partielle :  $A_1(\ell_0, v_0, \ell, v) \stackrel{def}{=} \ell = \ell_f \Rightarrow \mathbf{post}(P)(v_0, v)$
- ▶ Absence d'erreurs à l'exécution :  $A_2(\ell_0, v_0, \ell, v) \stackrel{def}{=} \wedge_{\ell', \ell \rightarrow \ell'} \mathbf{DOM}(\ell, \ell')(v)$

- ▶ Les vérifications sont longues et nombreuses
- ▶ Les vérifications sont parfois élémentaires et assez faciles à prouver
- ▶ Approche par vérification algorithmique via TLA et ses outils
- ▶ Approche par mécanisation du raisonnement symbolique via Event-B et ses outils

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites

$$\begin{array}{l} l0 : v = 3 \\ v := v+2; \\ l1 : v = 5 \end{array}$$

$$\begin{array}{l} l0 : v = 3 \\ v := v+2; \\ l1 : v = 5 \end{array}$$

- ▶ Annotation du code
- ▶ Traduction de l'invariant à vérifier
- ▶ Expression de la propriété de correction partielle
- ▶ Vérification de la propriété

$l0 : v = 3$   
 $v := v + 2;$   
 $l1 : v = 5$

- ▶ Annotation du code
- ▶ Traduction de l'invariant à vérifier
- ▶ Expression de la propriété de correction partielle
- ▶ Vérification de la propriété

```
-----MODULE an0-----  
EXTENDS Integers, TLC  
-----  
CONSTANTS v0,pc0  
VARIABLES v,pc  
-----  
(* extra definitions *)  
min == -2^{31}  
max == 2^{31}-1  
D == min..max  
-----  
(* precondition pre(x0,y0,z0,pc0) *)  
pre(fv) == fv=3  
ASSUME pre(v0)  
-----  
(* initial conditions *)  
Init == pc = "l0" /\ v=3  
-----  
(* actions *)  
skip == UNCHANGED <<pc,v>>  
a10l1 == pc="l0" /\ TRUE /\ pc'="l1" /\ v'=v+2  
-----  
(* next relation *)  
Next == skip \/ a10l1  
-----  
(* invariant properties *)  
i ==  
  /\ pc \in {"l0","l1"}  
  /\ pc="l0" => v=3  
  /\ pc="l1" => v=5  
-----  
(* safety properties *)  
suretecorrectionpartielle == pc="l1" => v=5  
sureteabsencederreurs == v \in D /\ v+2 \in D  
-----  
tocheck == i  
=====
```

- ▶ Le programme ou l'algorithme est annoté à des points de contrôle  $\ell \in \text{LOCATIONS}$  et à chaque point de contrôle  $\ell$  se trouve une assertion  $P_\ell(v_0, v)$ .



- ▶ Le programme ou l'algorithme est annoté à des points de contrôle  $\ell \in \text{LOCATIONS}$  et à chaque point de contrôle  $\ell$  se trouve une assertion  $P_\ell(v_0, v)$ .
- ▶ Si les deux points de contrôle  $\ell, \ell'$  définissent un calcul élémentaire, alors on définit une action  $\mathcal{E}(\ell, \ell')$  comme suit :

$$\begin{aligned}\mathcal{E}(\ell, \ell') &\triangleq \\ &\wedge c = \ell \\ &\wedge \text{cond}_{\ell, \ell'}(v) \\ &\wedge c' = \ell' \\ &\wedge v' = f_{\ell, \ell'}(v)\end{aligned}$$

- ▶ Le programme ou l'algorithme est annoté à des points de contrôle  $\ell \in \text{LOCATIONS}$  et à chaque point de contrôle  $\ell$  se trouve une assertion  $P_\ell(v_0, v)$ .
- ▶ Si les deux points de contrôle  $\ell, \ell'$  définissent un calcul élémentaire, alors on définit une action  $\mathcal{E}(\ell, \ell')$  comme suit :

$$\begin{aligned}\mathcal{E}(\ell, \ell') &\triangleq \\ &\wedge c = \ell \\ &\wedge \text{cond}_{\ell, \ell'}(v) \\ &\wedge c' = \ell' \\ &\wedge v' = f_{\ell, \ell'}(v)\end{aligned}$$

- $v$  est la variable de l'état mémoire ou la liste des variables de l'état mémoire ;  $v$  inclut les variables locales et les variables résultat.
- $c$  est une nouvelle variable qui modélise le flot de contrôle de type  $\text{LOCATIONS}$ .
- $\mathcal{E}(\ell, \ell')$  simule le calcul débutant en  $\ell$  et terminant en  $\ell'$  ;  $v$  est mise à jour.

$$\begin{aligned} i &\triangleq \\ &\quad \wedge c \in \text{LOCATIONS} \\ &\quad \wedge v \in \text{Type} \\ &\dots \\ &\quad \wedge c = \ell \Rightarrow P_\ell(v_0, v) \\ &\quad \wedge c = \ell' \Rightarrow P_{\ell'}(v_0, v) \\ &\dots \\ \text{safty} &\triangleq S(c, v_0, v) \end{aligned}$$

$$\begin{aligned} i &\triangleq \\ &\quad \wedge c \in \text{LOCATIONS} \\ &\quad \wedge v \in \text{Type} \\ \dots \\ &\quad \wedge c = \ell \Rightarrow P_\ell(v_0, v) \\ &\quad \wedge c = \ell' \Rightarrow P_{\ell'}(v_0, v) \\ \dots \\ \text{safety} &\triangleq S(c, v_0, v) \end{aligned}$$

- ▶  $\text{Type}$  est le type des variables  $v$  et est un ensemble de valeurs possibles.
- ▶ L'annotation donne gratuitement les conditions satisfaites par  $v$  quand le contrôle est en  $\ell$ , (resp. en  $\ell'$ ).
- ▶  $S(c, v_0, v)$  est une propriété de sûreté à vérifier et est un théorème dans le cas de *Event-B*.

# Méthode de vérification exhaustive ou algorithmique

---

- La relation de transition  $Next$  est définie par :

$$Next \triangleq \dots \vee \mathcal{E}(\ell, \ell') \vee \dots$$

- ▶ La relation de transition  $Next$  est définie par :

$$Next \triangleq \dots \vee \mathcal{E}(\ell, \ell') \vee \dots$$

- ▶ Les conditions initiales des variables sont à définir par un prédicat  $Init$

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites



- ▶ Définition d'un langage algorithmique simple.
- ▶ Commentaire spécifique dans entre (\* et \*)  
--algorithm nom { definitions }
- ▶ Génération d'une spécification TLA<sup>+</sup> avec introduction d'une nouvelle variable pc modélisant le contrôle.
- ▶ L'outil ToolBox dispose d'une fonctionnalité de traduction.

## Exemple (I)

---

```
----- MODULE exemple -----
EXTENDS Naturals, Integers, TLC
CONSTANTS x0,y0,z0,min,max,undef
-----

(* precondition *)
ASSUME x0 = y0 + 3*z0
-----

(*
--algorithm ex {
    variables x=x0,
               y = y0,
               z=z0;

{
10: assert x = y + 3*z /\ /\ y=y0 /\ z=z0 ;
    x := y+3*z;
11: assert x = y0+3*z0 /\ y=y0 /\ z=z0 ;
}
}
)
```

### Example (II)

```
----- MODULE exemple
```

$$\text{ISDEF}(X,Y) == X \# \text{undef} \Rightarrow X \setminus \text{in } Y$$
$$\text{DD}(X) == X \# \text{undef} \Rightarrow X \text{ \textit{in} min..max}$$

i ==

```
/\ pc \in {"10","11","Done"}
```

$$\wedge \text{ISDEF}(x, \text{Int}) \quad \wedge \text{ISDEF}(x, \text{Int}) \quad \wedge \text{ISDEF}(z, \text{Int})$$
$$\wedge \text{ pc} = "10" \Rightarrow x = y + 3 * z$$
$$\wedge \text{ pc} = \text{"11"} \Rightarrow x+y+z \geq y$$

```
post ==      x = y0+3*z0 /\ y=y0 /\ z=z0
```

$$\text{safetyrte} == \text{DD}(x) \wedge \text{DD}(y) \wedge \text{DD}(z)$$

```
safetypc == pc="Done" => post
```

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites

## General form for processes

---

—— MODULE module\_name ——

\\* TLA+ code

(\* ——algorithm algorithm\_name  
variables global\_variables

process p\_name = ident  
variables local\_variables  
begin  
  \\* pluscal code  
end process

process p\_group \in set  
variables local\_variables  
begin  
  \\* pluscal code  
end process

end algorithm; \*)

## Example 1

---

```
process pro = "test"  
begin  
    print<<"test">>;  
end process
```

- ▶ A multiprocess algorithm contains one or more processes.
- ▶ A process begins in one of two ways :
  - defining a set of processes : `process ( ProcName  $\in$  IdSet )`
  - defining one process with an identifier `process ( ProcName = Id )`
- ▶ `self` designates the current process

A process S sends a message to a process R

```

—algorithm ex_process {
    variables
        input = <<>>, output = <<>>,
        msgChan = <<>>, ackChan = <<>>,
        newChan = <<>>;
    /* defining macros
    process (Sender = "S")
    {

    }; /* end Sender process block
    process (Receiver = "R")
    {

    }; /* end Receiver process block
} /* end algorithm

```



```
—algorithm ex_process {  
  variables  
    input = <<>>, output = <<>>,  
    msgChan = <<>>, ackChan = <<>>,  
    newChan = <<>>;  
  macro Send(m, chan) {  
    chan := Append(chan, m);  
  }  
  macro Recv(v, chan) {  
    await chan # <<>>;  
    v := Head(chan);  
    chan := Tail(chan);  
  }  
  
  * Processes S and R  
  
} \* end algorithm
```

```
—algorithm ex_process {  
  variables  
    input = <<>>, output = <<>>,  
    msgChan = <<>>, ackChan = <<>>,  
    newChan = <<>>;  
  /* defining macros  
  process (Sender = "S")  
  variables msg;  
  {  
    sending:  Send("Hello", msgChan);  
    printing: print <<"Sender", input>>;  
  }; /* end Sender process block  
  process (Receiver = "R")  
  {  
    waiting: Recv(msg, msgChan);  
    adding:  output := Append(output, msg);  
    printing: print <<"Receiver", output>>;  
  }; /* end Receiver process block  
} /* end algorithm
```

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites

```
macro Name(var1, ...)  
begin  
  \* something to write  
end macro;
```

```
procedure Name(arg1, ...)  
variables var1 = ... \* not \in, only =  
begin  
  Label:  
  \* something  
  return;  
end procedure;
```

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites

requires  $x0 \geq 0$ ;  
ensures  $x_f = x0+2$ ;  
variables  $X$

```
begin  
  int  $X = x0$ ;  
  0 :  $x = x0$   
   $X = X+2$ ;  
  1 :  $x = x0+2$   
end
```

- ▶  $x0 \geq 0 \wedge x = x_0 \Rightarrow x = x0$
- ▶  $x = x0+2 \Rightarrow x = x0+2$
- ▶ conditions de vérification  $0 \longrightarrow 1$  :  
 $x = x0 \wedge x' = x+2 \Rightarrow x' = x0+2$
- ▶  $(x0 \geq 0, x == x0, x! = x0)$
- ▶  $(x == x0+2, x! = x0+2)$
- ▶  $(x == x0, xp == x+2, xp! = x0+2)$

### Listing 1 – z3 en Python

```
from numbers import Real  
from z3 import *  
x = Real('x')  
xp = Real('xp')  
x0 = Real('x0')  
s = Solver()  
s.add(x0 >= 0, x == x0, x != x0)  
print(s.check())  
s.add(x == x0+2, x != x0+2)  
print(s.check())  
s.add(x == x0, xp == x + 2, xp != x0+2)  
print(s.check())
```

- 1 Exemples de correction partielle (affectation simple)
- 2 Annotation et vérification outillée avec TLA/TLA<sup>+</sup>  
Vérification avec TLA et ses outils
- 3 Le langage PlusCal  
Defining processes in PlusCal  
Macros and Procedures
- 4 Vérification d'un contrat avec un solveur Z3
- 5 Conclusion et limites





- ▶  $\mathcal{R}$  : exigences du système.

- ▶  $\mathcal{R}$  : exigences du système.
- ▶  $\mathcal{D}$  : domaine du problème.

- ▶  $\mathcal{R}$  : exigences du système.
- ▶  $\mathcal{D}$  : domaine du problème.
- ▶  $\mathcal{S}$  : système répondant aux spécifications.

- 9 1 1

- ▶  $\mathcal{R}$  : exigences du système.
- ▶  $\mathcal{D}$  : domaine du problème.
- ▶  $\mathcal{S}$  : système répondant aux spécifications.

$$\mathcal{D}, \mathcal{S} \quad \text{SATISFAIT} \quad \mathcal{R}$$

- ▶  $\mathcal{R}$  : pre/post.
- ▶  $\mathcal{D}$  : entiers, réels, ...
- ▶  $\mathcal{S}$  : code, procédure, programme, ...

$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \left\{ \begin{array}{l} \mathbf{pre}(\text{ALG})(v) \\ \mathbf{post}(\text{ALG})(v_0, v) \end{array} \right.$$

$\mathcal{D}$
<hr/>
$\mathbf{pre}(\text{ALG})(v)$
$\mathbf{post}(\text{ALG})(v_0, v)$
<hr/>
ALG

$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \left\{ \begin{array}{l} \mathbf{pre}(\text{ALG})(v) \\ \mathbf{post}(\text{ALG})(v_0, v) \end{array} \right.$$



Vérification de conditions de vérification

$\mathcal{D}$
<hr/>
$\mathbf{pre}(\text{ALG})(v)$
$\mathbf{post}(\text{ALG})(v_0, v)$
<hr/>
ALG

$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \left\{ \begin{array}{l} \text{pre}(\text{ALG})(v) \\ \text{post}(\text{ALG})(v_0, v) \end{array} \right.$$



### Vérification de conditions de vérification

$\mathcal{D}$
$\text{pre}(\text{ALG})(v)$
$\text{post}(\text{ALG})(v_0, v)$
ALG

- Vérification des conditions de vérification avec un model-checker par exploration de tous les états.



$$\mathcal{D}, \text{ALG} \quad \text{SATISFAIT} \quad \left\{ \begin{array}{l} \text{pre}(\text{ALG})(v) \\ \text{post}(\text{ALG})(v_0, v) \end{array} \right.$$



### Vérification de conditions de vérification

$\mathcal{D}$
$\text{pre}(\text{ALG})(v)$
$\text{post}(\text{ALG})(v_0, v)$
ALG

- ▶ Vérification des conditions de vérification avec un model-checker par exploration de tous les états.
- ▶ Vérification des conditions de vérification avec un outil de preuve formelle.



- ▶ Vérifier les énoncés de la forme  $\Gamma \vdash P$  (séquents)

- ▶ Vérifier les énoncés de la forme  $\Gamma \vdash P$  (séquents)
- ▶ Enoncer ou calculer les invariants d'un modèle :  $\text{REACHABLE}(M)$ .

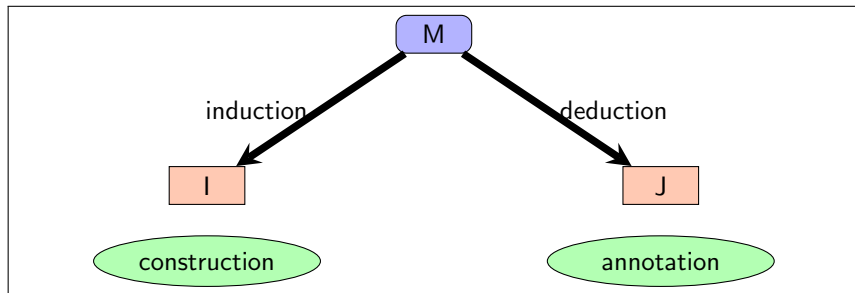
- ▶ Vérifier les énoncés de la forme  $\Gamma \vdash P$  (séquents)
- ▶ Énoncer ou calculer les invariants d'un modèle :  $\text{REACHABLE}(M)$ .
- ▶  $\text{TLA}^+$  versus Event-B
  - Plate-formes :  $\text{TLA}^+$  avec TLAPS et Toolbox, Event-B avec Rodin
  - Langage de la théorie des ensembles avec quelques différences
  - Fonctionnalités des outils
    - ▶ Éditeurs de modèles :  $\text{TLA}^+$  et Event-B
    - ▶ Model-Checking :  $\text{TLA}^+$  et Event-B
    - ▶ Assistant de preuve : Event-B
    - ▶ Animateur et Model-Checker ProB

- ▶ Vérifier les énoncés de la forme  $\Gamma \vdash P$  (séquents)
- ▶ Énoncer ou calculer les invariants d'un modèle :  $\text{REACHABLE}(M)$ .
- ▶  $\text{TLA}^+$  versus Event-B
  - Plate-formes :  $\text{TLA}^+$  avec TLAPS et Toolbox, Event-B avec Rodin
  - Langage de la théorie des ensembles avec quelques différences
  - Fonctionnalités des outils
    - ▶ Éditeurs de modèles :  $\text{TLA}^+$  et Event-B
    - ▶ Model-Checking :  $\text{TLA}^+$  et Event-B
    - ▶ Assistant de preuve : Event-B
    - ▶ Animateur et Model-Checker ProB
- ▶ Développement d'outils symboliques comme les solveurs SMT ou des procédures de décision

- ▶ TLA<sup>+</sup> et TLA Toolbox : logique temporelle, théorie des ensembles, calcul des prédicats, model-checker
- ▶ Event-B et Rodin : théorie des ensembles, assistant de preuve, model-checker, animateur
- ▶ B et Event-B et ProB : théorie des ensembles, model-checker, animateur, validation
- ▶ Promela et SPIN : logique temporelle, model-checking
- ▶ C et Frama-C : analyse sémantique des programmes, assistants de preuve, solveurs SMT.
- ▶ Spec# et Rise4fun : pre/post, contrats
- ▶ PAT : cadre générique pour créer son propre model-checker (classique, temps réel, probabiliste, stochastique)
- ▶ C et cppcheck : analyse statique de programmes C ou C++

## Vérification à faire mais comment automatiquement ?

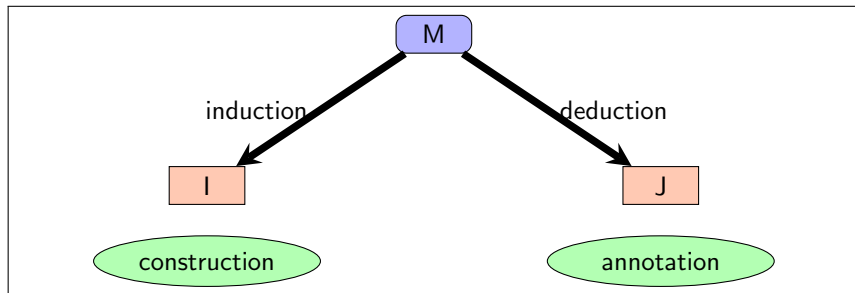
- Application de la correction du principe d'induction : si on vérifie les trois propriétés, alors  $A$  est une propriété de sûreté pour le modèle en question : outil de déduction.





## Vérification à faire mais comment automatiquement ?

- ▶ Application de la correction du principe d'induction : si on vérifie les trois propriétés, alors  $A$  est une propriété de sûreté pour le modèle en question : outil de déduction.
- ▶ Si on veut montrer que  $A$  est une propriété de sûreté, alors on doit utiliser l'invariant pour induire des annotations pour le modèle : outil d'induction.



- ▶  $\forall x_0, x \in \text{VALS}. \text{Init}(x_0) \wedge \text{Next}^*(x_0, x) \Rightarrow A(x)$
- ▶  $\forall x \in \text{VALS}. (\exists x_0. x_0 \in \text{VALS} \wedge \text{Init}(x_0) \wedge \text{Next}^*(x_0, x)) \Rightarrow A(x).$
- ▶  $\text{REACHABLE}(M) = \{u \mid u \in \text{VALS} \wedge (\exists x_0. x_0 \in \text{VALS} \wedge \text{Init}(x_0) \wedge \text{Next}^*(x_0, u))\}$  est l'ensemble des états accessibles à partir des états initiaux.
- ▶ Model Checking : on doit montrer l'inclusion  $\text{REACHABLE}(M) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}.$
- ▶ Preuves : définir un invariant  $I(\ell, v) \equiv \bigvee_{\ell \in \text{LOCATIONS}} \left( \bigvee_{v \in \text{MEMORY}} P_\ell(v) \right)$  avec la famille d'annotations  $\{P_\ell(v) : \ell \in \text{LOCATIONS}\}$  et démontrer les conditions de vérification.
- ▶ Analyse automatique :
  - Mécaniser la vérification des conditions de vérification
  - Calculer  $\text{REACHABLE}(M)$
  - Calculer une valeur approchée de  $\text{REACHABLE}(M)$

$$(\mathcal{P}(\text{VALS}), \subseteq) \xleftrightarrow[\alpha]{\gamma} (D, \sqsubseteq)$$

$$\alpha(\text{REACHABLE}(M)) \sqsubseteq A \text{ ssi } \text{REACHABLE}(M) \subseteq \gamma(A)$$

Si  $\gamma(A) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}$ , alors

$$\text{REACHABLE}(M) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}$$

- ▶ Mécaniser la vérification des conditions de vérification
- ▶ Calculer  $\text{REACHABLE}(M)$  comme un point-fixe.
- ▶ Calculer une valeur approchée de  $\text{REACHABLE}(M)$

$$(\mathcal{P}(\text{VALS}), \subseteq) \xleftrightarrow[\alpha]{\gamma} (D, \sqsubseteq)$$
$$\alpha(\text{REACHABLE}(M)) \sqsubseteq A \text{ ssi } \text{REACHABLE}(M) \subseteq \gamma(A)$$

Si  $A$  vérifie  $\gamma(A) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}$ , alors  
 $\text{REACHABLE}(M) \subseteq \{u \mid u \in \text{VALS} \wedge A(u)\}$

## Method for verifying program properties

correctness and Run Time Errors

A program  $P$  satisfies a (pre,post) contract :

- ▶  $P$  transforms a variable  $v$  from initial values  $v_0$  and produces a final value  $v_f : v_0 \xrightarrow{P} v_f$
- ▶  $v_0$  satisfies pre :  $\text{pre}(v_0)$  and  $v_f$  satisfies post :  $\text{post}(v_0, v_f)$
- ▶  $\text{pre}(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow \text{post}(v_0, v_f)$
- ▶  $\mathbb{D}$  est le domaine RTE de  $V$

requires  $\text{pre}(v_0)$   
ensures  $\text{post}(v_0, v_f)$   
variables  $V$

```
begin  
  0 :  $P_0(v_0, v)$   
  instruction0  
  ...  
  i :  $P_i(v_0, v)$   
  ...  
  instructionf-1  
  f :  $P_f(v_0, v)$   
end
```

▶  $\text{pre}(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$

▶  $P_f(v_0, v) \Rightarrow \text{post}(v_0, v)$

▶ For any pair of labels  $\ell, \ell'$   
such that  $\ell \rightarrow \ell'$ , one verifies that, pour  
any values  $v, v' \in \text{MEMORY}$

$$\left( \begin{array}{l} P_\ell(v_0, v) \wedge \text{cond}_{\ell, \ell'}(v) \\ \wedge v' = f_{\ell, \ell'}(v) \end{array} \right) \Rightarrow P_{\ell'}(v_0, v')$$

▶ For any pair of labels  $m, n$   
such that  $m \rightarrow n$ , one verifies that,  
 $\forall v, v' \in \text{MEMORY} : P_m(v_0, v) \Rightarrow$

**DOM**( $m, n$ )( $v$ )

## Summary of concepts

