UNIVERSITÉ DE LORRAINE

TELECOM nancy
Ingénieurs du numérique • Inspiring your digital future

# Cours Modélisation et Vérification des Systèmes Informatiques (MVSI)

# Modélisation, spécification et vérification CM4

Dominique Méry
Telecom Nancy, Université de Lorraine

**Année universitaire 2024-2025**

# Plan

# Sommaire

Le modèle relationnel $M(P)$ pour le programme P annoté est donc défini comme suit :
$M(P) \stackrel{def}{=}$
$(Th(s,c),(pc,v),\textsc{Locations}\times\textsc{Memory},Init(\ell,v),\{r_{\ell,\ell'}|\ell,\ell' \in \textsc{Locations} \wedge \ell \longrightarrow \ell'\})$.
La définition de $Init(x)$ est dépendante de la précondition de P :

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

## Conditions initiales

Les deux propriétés suivantes sont équivalentes :

- $\forall x_0 \in \textsc{Vals} : \mathit{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0)$
- $\forall v \in \textsc{Memory}.\textbf{pre}(\mathrm{P})(v) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v)$

## Pas d'induction

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

## Pas d'induction

Les deux propriétés suivantes sont équivalentes :

- $\forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x\ r_i\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $\forall \ell, \ell' \in \textsc{Locations} : \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

# Pas d'induction (explication)

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

# Pas d'induction (explication)

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

# Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$

# Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$

# Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$

# Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$

# Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \Rightarrow pc = \ell' \wedge P_\ell(v_0, v')$

# Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell'$ (Tautologie)

# Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell'$ (Tautologie)
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ P_{\ell'}(v_0, v')$

# Pas d'induction (explication)

- $\mathrm{J}(x_0 n x) \wedge x\ r_{\ell,\ell'}\ x' \Rightarrow \mathrm{J}(x_0, x')$
- $x\ r_{\ell,\ell'}\ x' \stackrel{def}{=} (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell'$
- $I(x) \stackrel{def}{=} \exists \ell, v.(\ell \in \textsc{Locations} \wedge v \in \textsc{Memory} \wedge x = (\ell, v) \wedge P_\ell(v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x \in \textsc{Vals} \wedge x = (\ell, v) \wedge x_0 = (\ell_0, v_0) \wedge J(x_0, x))$
- $\mathrm{J}(x_0 n x) \equiv pc = \ell \wedge P_\ell(v_0, v)$
- $\mathrm{J}(x_0, x') \equiv pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell' \wedge P_\ell(v_0, v')$
- $pc = \ell \wedge P_\ell(v_0, v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ pc = \ell'$ (Tautologie)
- $pc = \ell \wedge P_\ell(v) \wedge (pc = \ell \wedge cond_{\ell,\ell'}(v) \wedge \wedge v' = f_{\ell,\ell'}(v) \wedge pc' = \ell' \ \Rightarrow\ P_{\ell'}(v_0, v')$
- $P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

# Conclusion

- $J(x_0, x) \stackrel{def}{=} \exists \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v))$
- $P_\ell(v_0, v) \stackrel{def}{=} \exists x.(x, x_0 \in \textsc{Vals} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge J(x_0), x)$
- $\mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x)$
- $\exists \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell, v, v_0.(\ell \in \textsc{Locations} \wedge v, v_0 \in \textsc{Memory} \wedge x = (\ell, v) wedge x_0 = (\ell_0, v_0) \wedge P_\ell(v_0, v)) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell \in \textsc{Locations}, v, v_0 \in \textsc{Memory}.P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$

## Conclusion

Les deux propriétés suivantes sont équivalentes :

- $\mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x)$
- $\forall \ell \in \textsc{Locations}, v, v_0 \in \textsc{Memory}.P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$

# Traduction des conditions de vérifications

Les conditions de vérification suivantes sont équivalentes :

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

$$\begin{cases} (1) & \mathsf{Init}(x_0) \Rightarrow \mathrm{J}(x_0, x_0) \\ (2) & \mathrm{J}(x_0, x) \Rightarrow \mathrm{A}(x_0, x) \\ (3) & \forall i \in \{0, \ldots, n\} : \mathrm{J}(x_0, x) \wedge x \; r_i \; x' \Rightarrow \mathrm{J}(x_0, x') \end{cases}$$

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

$$\begin{cases} (1) & \mathbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_{\ell_0}(v_0, v) \\ (2) & \forall \ell \in \text{LOCATIONS}. P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v) \\ (3) & \forall \ell, \ell' \in \text{LOCATIONS} : \\ & \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v') \end{cases}$$

- Le programme est annoté.
- Les annotations définissent un invariant à vérifier selon les conditions de vérification.
- $A(\ell, v)$ est l'énoncé de la propriété de sûreté à vérifier.

## Méthode relationnelle de correction de propriétés de sûreté

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

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

## Floyd and Hoare

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

# Condition de vérification pour l'affectation

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

- $\forall v, v' \in \textsc{Memory}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

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

# Condition de vérification pour l'affectation

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

- $\forall v, v' \in \textsc{Memory}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

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

# Condition de vérification pour l'affectation

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

- $\forall v, v' \in \textsc{Memory}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \textsc{Memory}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$

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

# Condition de vérification pour l'affectation

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

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

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

# Condition de vérification pour l'affectation

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

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

- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge TRUE \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v, v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')$
- $\forall v \in \text{MEMORY}. P_\ell(v_0, v) \Rightarrow P_{\ell'}(v_0, v \mapsto f_{\ell,\ell'}(v))$ (l'axiomatique de Hoare).
- $\forall v \in \text{MEMORY}. (\exists v' \in \text{MEMORY}. P_\ell(v_0, v) \wedge v' = f_{\ell,\ell'}(v)) \Rightarrow P_{\ell'}(v_0, v')$ correspond à la règle d'affectation de Floyd.

# Conditions de vérification pour l'itération

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

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

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

# Conditions de vérification pour la conditionnelle

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

Pour la structure de conditionnelle, les conditions suivantes :

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

# Correction partielle d'un programme

La correction partielle vise à établir qu'un programme $\mathrm{P}$ est partiellement correct par rapport à sa précondition et à sa postcondition.

- la spécification des données de $\mathrm{P}$ **pre**$(\mathrm{P})(\mathrm{v}_0)$
- la spécification des résultats de $\mathrm{P}$ **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$
- une famille d'annotations de propriétés $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ pour ce programme.
- une propriété de sûreté définissant la correction partielle $pc = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v_f)$ où $\ell_f$ est l'étiquette marquant la fin du programme $\mathrm{P}$

⊠ Definition

Le programme $\mathrm{P}$ est partiellement correct par rapport à **pre**$(\mathrm{P})(\mathrm{v}_0)$ et **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$, si la propriété $pc = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v)$ est une propriété de sûreté pour ce programme.

# Correction partielle d'un programme

Si les conditions suivantes sont vérifiées :

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

alors le programme P est partiellement correct par rapport à **pre**(P)($v_0$) et **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 :

# Reformulation du calcul

- $\mathsf{pre}(v_0) \wedge v_0 \xrightarrow{\mathsf{P}} v_f \Rightarrow \mathsf{post}(v_0, v_f)$

(expression de la correction partielle)

- $pc_0 = \ell_0 \wedge \mathsf{pre}(v_0) \wedge (pc_0, v_0) \xrightarrow{\mathsf{Next}^\star} (pc, v) \wedge pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f)$

(big-step semantics et small-step semantics equivalence)

- $pc_0 = \ell_0 \wedge \mathsf{pre}(v_0) \wedge (pc_0, v_0) \xrightarrow{\mathsf{Next}^\star} (pc, v) \Rightarrow (pc = \ell_f \Rightarrow \mathsf{post}(v_0, v_f))$

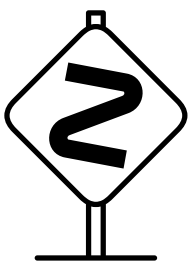
(implication and conjunction property)

- $Init(x_0) \wedge x_0 \xrightarrow{\mathsf{Next}^\star} x \Rightarrow \mathsf{PC}(x_0, x)$

$(Init(x_0) \stackrel{def}{=} pc_0 = \ell_0 \wedge \mathsf{pre}(v_0)$

$x_0 \stackrel{def}{=} (\ell_0, v_0)$ and $x \stackrel{def}{=} (pc, v)$

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



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

# Vérification du contrat

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
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
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 $v, v' \in \textsc{Memory}$

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

## 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 $f$, $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$

## Absence d'erreurs à l'exécution

- 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 '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 $\mathrm{P}$ ne va pas produire des erreurs durant son exécution par rapport à sa précondition et à sa postcondition.

- la spécification des données de $\mathrm{P}$ **pre**$(\mathrm{P})(\mathrm{v})$
- la spécification des résultats de $\mathrm{P}$ **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$
- une famille d'annotations de propriétés $\{P_\ell(v) : \ell \in \textsc{Locations}\}$ pour ce programme.
- une propriété de sûreté définissant l'absence d'erreurs à l'exécution :

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

☒ Definition

Le programme $\mathrm{P}$ ne produira pas d'erreurs à l'exécution par rapport à **pre**$(\mathrm{P})(\mathrm{v})$ et **post**$(\mathrm{P})(\mathrm{v}_0,\mathrm{v})$, si la propriété $\bigwedge_{\ell \in \textsc{Locations} - \{output\}, n \in \textsc{Locations}, \ell \longrightarrow n} (\mathbf{DOM}(\ell, n)(v))$ est une propriété de sûreté pour ce programme.

# Principe d'induction pour RTE

## RTE = Run Time Error

Si les conditions suivantes sont vérifiées :

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

alors le programme P ne produira pas d'erreurs à l'exécution par rapport à **pre**(P)($v_0$) et **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.

# Méthode relationnelle de vérification pour RTE

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$(v_0, v_f)$
- $\mathbb{D}$ est le domaine RTE de V

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables $V$
begin
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
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 $v, v' \in \textsc{Memory}$

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

- $\forall m \in \textsc{Locations} - \{\ell_f\}, n \in \textsc{Locations}, \forall v_0, v, v' \in \textsc{Memory} :$ $m \longrightarrow n : P_m(v_0, v) \Rightarrow$ **DOM**$(m, n)(v)$

# Current Summary

# Simple contrat (I)

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

requires $pre(v_0)$
ensures $post(v_0, v_f)$
variables $V$
begin
$0 : P_0(v_0, v)$
$V := f(V)$
$f : P_f(v_0, v)$
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 $A(x_0, x) \stackrel{def}{=} (x = (f, v) \Rightarrow post(v_0, v))$.

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

# Simple contrat (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 $A(x_0, x) \stackrel{def}{=} (x = (f, v) \Rightarrow post(v_0, v))$.

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

# Current Summary

## Méthode de correction de propriétés de sûreté

Soit $\mathrm{A}(\ell_0, v_9, \ell, v)$ une propriété d'un programme $\mathrm{P}$. Soit une famille d'annotations famille de propriétés $\{P_\ell(v_0, v) : \ell \in \textsc{Locations}\}$ pour ce programme. Si les conditions suivantes sont vérifiées :

$\forall v0, v, v' \in \textsc{Memory}$ :

$$\begin{cases} (1) \quad \textbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v) \\ (2) \quad \forall \ell \in \textsc{Locations}. P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v) \\ (3) \quad \forall \ell, \ell' \in \textsc{Locations} : \\ \ell \longrightarrow \ell' \Rightarrow P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v') \end{cases},$$

alors $\mathrm{A}(\ell_0, v_9, \ell, v)$ est une propriété de sûreté pour le programme $\mathrm{P}$.

1. Définir la précondition $\textbf{pre}(\mathrm{P})(v_0, v)$
2. Annoter le programme avec des prédicats $P_\ell(v_0, v)$ où $\ell \in \textsc{Locations}$
3. Vérifier que $\textbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v)$ où $\ell \in \textsc{Inputs}$ (ensemble des points d'entrée.
4. Vérifier que $P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell, v)$ où $\ell \in \textsc{Locations}$
5. Pour chaque paire de points de contrôle $(\ell, \ell')$ telle que $\ell \longrightarrow \ell'$ (successifs), vérifier que $\big(P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')\big)$.

# Conditions de vérification

1. Vérifier que $\mathbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$ où $\ell \in \textsc{Inputs}$ (ensemble des points d'entrée.
2. Vérifier que $P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$ où $\ell \in \textsc{Locations}$
3. Pour chaque paire de points de contrôle $(\ell, \ell')$ telle que $\ell \longrightarrow \ell'$ (successifs), vérifier que $(P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v'))$.

# Conditions de vérification

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

## Exemples de propriétés de sûreté

- Correction partielle : $\mathrm{A}_1(\ell_0, v_0, \ell, v) \stackrel{def}{=} \ell = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v)$
- Absence d'erreurs à l'exécution : $\mathrm{A}_2(\ell_0, v_0, \ell, v) \stackrel{def}{=} \wedge_{\ell', \ell \to \ell'}$ **DOM**$(\ell, \ell')(v)$

# Mécanisation et automatisation de la vérification

- 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

# Current Subsection Summary

# Traduction des annotations

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

# Traduction des annotations

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

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

# Traduction des annotations

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

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

```
---------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>>
al0l1 == pc="l0" /\ TRUE /\ pc'="l1" /\ v'=v+2
------------------------------
(* next relation *)
Next == skip \/ al0l1
------------------------------
(* 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 :

$$
\begin{array}{l}
\mathcal{E}(\ell, \ell') \triangleq \\
\qquad \wedge\ c = \ell \\
\qquad \wedge\ cond_{\ell,\ell'}(v) \\
\qquad \wedge\ c' = \ell' \\
\qquad \wedge\ v' = f_{\ell,\ell'}(v)
\end{array}
$$

- 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 :

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

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

$$
\begin{array}{l}
i \triangleq \\
\quad \wedge\ c \in \textsc{Locations} \\
\quad \wedge\ v \in Type \\
\ldots \\
\quad \wedge\ c = \ell \Rightarrow P_\ell(v_0, v) \\
\quad \wedge\ c = \ell' \Rightarrow P_{\ell'}(v_0, v) \\
\ldots \\
safety \triangleq\ S(c, v_0, v)
\end{array}
$$

$$
\begin{array}{l}
i \triangleq \\
\quad \wedge\ c \in \textsc{Locations} \\
\quad \wedge\ v \in Type \\
\ldots \\
\quad \wedge\ c = \ell \Rightarrow P_\ell(v_0, v) \\
\quad \wedge\ c = \ell' \Rightarrow P_{\ell'}(v_0, v) \\
\ldots \\
safety \triangleq\ S(c, v_0, v)
\end{array}
$$

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

# Méthode de vérification exhaustive ou algorithmique

# Méthode de vérification exhaustive ou algorithmique

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

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

# Méthode de vérification exhaustive ou algorithmique

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

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

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

# Current Summary

# PlusCal un langage algorithmique plongé dans TLA/TLA$^+$

- Définition d'un langage algorithmique simple.
- Commentaire spécifique dans entre (* et *)
  `--algorithm nom { definitions }`
- Génération d'une spécification TLA$^+$ 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;

{
l0: assert x = y + 3*z/\ /\ y=y0 /\ z=z0 ;
    x := y+3*z;
l1: assert x = y0+3*z0 /\ y=y0 /\ z=z0 ;
}
}
*)
```

# Exemple (II)

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

-----------------------------------------------
ISDEF(X,Y) == X # undef => X \in Y
DD(X) == X # undef => X \in min..max
-----------------------------------------------
i ==
    /\ pc \in {"l0","l1","Done"}
    /\ ISDEF(x,Int)  /\ ISDEF(x,Int) /\ ISDEF(z,Int)
    /\ pc = "l0" => x = y + 3*z
    /\ pc = "l1" => x+y+z \geq y
post ==       x = y0+3*z0 /\ y=y0 /\ z=z0

safetyrte ==DD(x) /\ DD(y) /\ DD(z)
safetypc == pc="Done" => post
=================================================================
```

# Current Subsection Summary

# 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
```

# Process in PlusCal

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

# 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
```

# Using macros for defining sending and receiving prilmitives

```
--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
```

# Defining processes S and R

```
--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
```

# Current Subsection Summary

# Macros et Procedures

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

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

# Current Summary

# Vérification du contrat avec le solveur Z3

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

begin
$intX = x0;$
$0 : x = x0$
$X = X{+}2;$
$1 : x = x0{+}2$
end

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

## 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())
```

# Current Summary

# Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

# Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

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

# Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

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

# Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

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

## Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

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

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

# Tryptique $(\mathcal{D}, \mathcal{S}, \mathcal{R})$

- $\mathcal{R}$ : 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, ...

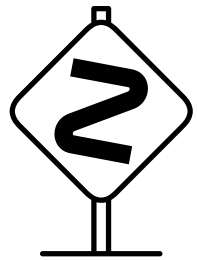
# Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \begin{cases} \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \end{cases}$$

$$\frac{\mathcal{D} \qquad \textbf{pre}(\textsc{Alg})(v) \qquad \textbf{post}(\textsc{Alg})(v_0, v)}{\textsc{Alg}}$$

# Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \begin{cases} \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \end{cases}$$

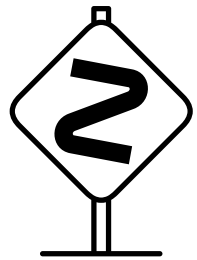


**Vérification de conditions de vérification**

$$\frac{\mathcal{D} \qquad \textbf{pre}(\textsc{Alg})(v) \qquad \textbf{post}(\textsc{Alg})(v_0, v)}{\textsc{Alg}}$$

# Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \begin{cases} \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \end{cases}$$



**Vérification de conditions de vérification**

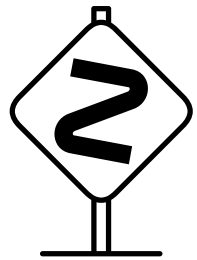
$$\frac{\mathcal{D}}{\dfrac{\textbf{pre}(\textsc{Alg})(v)}{\textbf{post}(\textsc{Alg})(v_0, v)}}{\textsc{Alg}}$$

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

# Sommaire des résultats

$$\mathcal{D}, \textsc{Alg} \quad \textsc{satisfait} \quad \begin{cases} \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \end{cases}$$

$$\frac{\mathcal{D}}{\dfrac{\textbf{pre}(\textsc{Alg})(v)}{\textbf{post}(\textsc{Alg})(v_0, v)}}{\textsc{Alg}}$$



**Vérification de conditions de vérification**

- 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.

# Observations

# Observations

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

# Observations

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

# Observations

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

# Observations

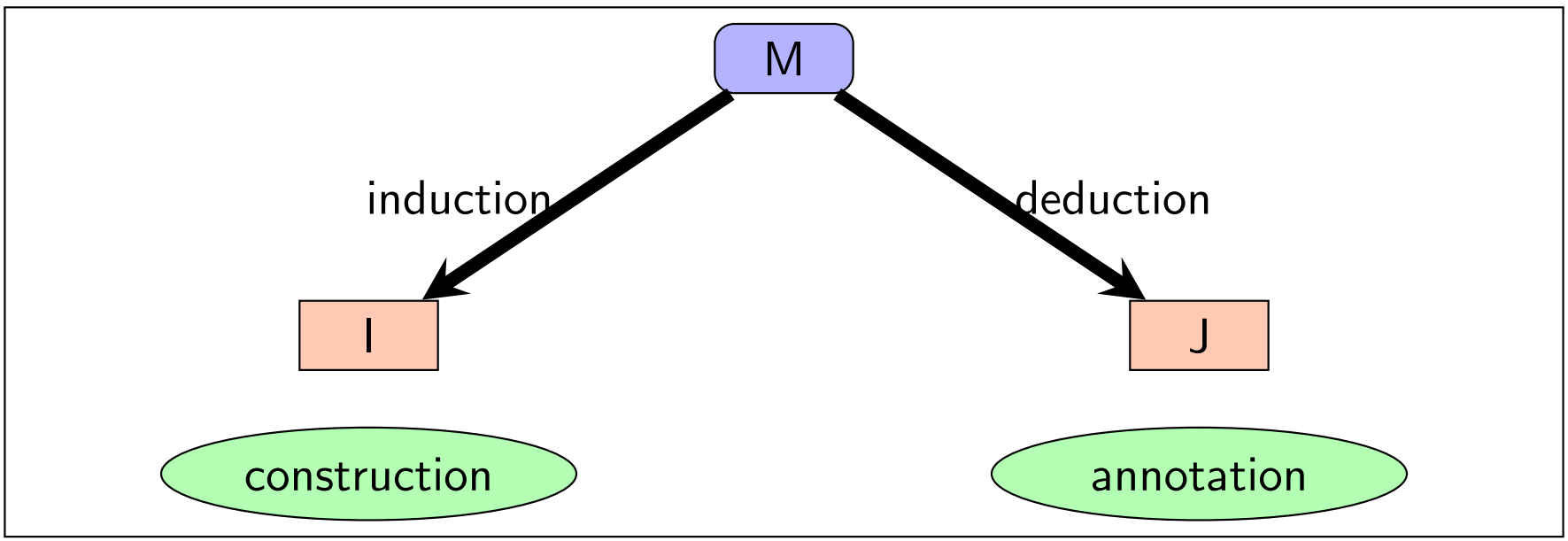
- Vérifier les énoncés de la forme $\Gamma \vdash P$ (séquents)
- Enoncer ou calculer les invariants d'un modèle : $\text{REACHABLE}(M)$.
- TLA$^+$ versus Event-B
  - Plate-formes : TLA$^+$ avec TLAPS et Toolbox, Event-B avec Rodin
  - Langage de la théorie des ensembles avec quelques différences
  -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

# Revue des outils

- TLA$^+$ 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++

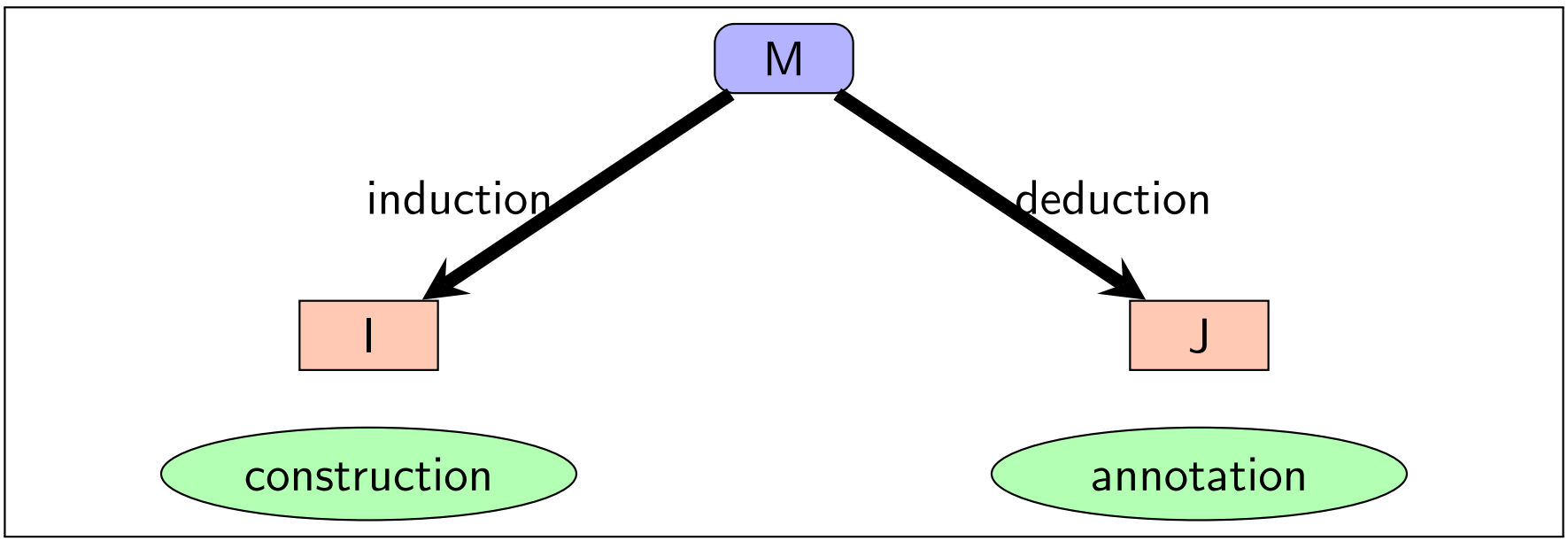
# Vérification à faire mais comment automatiquement ?

- 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.



## Point d'étape

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

  $(\mathcal{P}(\textsc{Vals}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (D, \sqsubseteq)$
  $\alpha(\textsc{reachable}(M)) \sqsubseteq A$ ssi $\textsc{reachable}(M) \sqsubseteq \gamma(A)$
  Si $\gamma(A) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$, alors
  $\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$

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

$$(\mathcal{P}(\textsc{Vals}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (D, \sqsubseteq)$$
$$\alpha(\textsc{reachable}(M)) \sqsubseteq A \text{ ssi } \textsc{reachable}(M) \sqsubseteq \gamma(A)$$

Si $A$ vérifie $\gamma(A) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$, alors
$\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$

# 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 : $pre(v_0)$ and $v_f$ satisfies post : $post(v_0, v_f)$
- $pre(v_0) \wedge v_0 \xrightarrow{P} v_f \Rightarrow post(v_0, v_f)$
- $\mathbb{D}$ est le domaine RTE de V

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

begin
$0 : P_0(v_0, v)$
$\texttt{instruction}_0$
...
$i : P_i(v_0, v)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(v_0, v)$
end

- $pre(v_0) \wedge v = v_0 \Rightarrow P_0(v_0, v)$
- $P_f(v_0, v) \Rightarrow post(v_0, v)$
- For any pair of labels$\ell, \ell'$ such that $\ell \longrightarrow \ell'$, one verifies that, pour any values $v, v' \in \textsc{Memory}$

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

- For any pair of labels $m, n$ such that $m \longrightarrow n$, one verifies that, $\forall v, v' \in \textsc{Memory} : P_m(v_0, v) \Rightarrow$ **DOM**$(m, n)(v)$

# Summry of concepts