Cours MALG & MOVEX

Modélisation, spécification et vérification (II)

Dominique Méry
Telecom Nancy, Université de Lorraine

Année universitaire 2024-2025

Plan

Sommaire

Current Summary

Annotation

Méthode de correction de propriétés de sûreté

Soit $\mathrm{A}(\ell_0, v_9, \ell, v)$ une propriété d'un programme 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 :

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

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

alors $\mathrm{A}(\ell_0, v_9, \ell, v)$ est une propriété de sûreté pour le programme 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 (ℓ, ℓ') 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 (ℓ, ℓ') 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 $\mathbf{pre}(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$ 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 (ℓ, ℓ') 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 : $\mathrm{A}_1(\ell_0, v_0, \ell, v) \stackrel{def}{=} \ell = \ell_f \Rightarrow \mathbf{post}(\mathrm{P})(v_0, v)$
- Absence d'erreurs à l'exécution : $\mathrm{A}_2(\ell_0, v_0, \ell, v) \stackrel{def}{=} \wedge_{\ell', \ell \rightarrow \ell'} \mathbf{DOM}(\ell, \ell')(v)$

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

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

- 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>>
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 ℓ 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 ℓ se trouve une assertion $P_\ell(v_0, v)$.
- Si les deux points de contrôle ℓ, ℓ' 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 ℓ se trouve une assertion $P_\ell(v_0, v)$.
- Si les deux points de contrôle ℓ, ℓ' 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 ℓ et terminant en o ℓ' ; v est mise à jour.

Méthode de vérification avec le toolset TLAPS et TLA/TLA$^+$

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

Méthode de vérification avec le toolset TLAPS et TLA/TLA$^+$

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

- $Type$ 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 ℓ, (resp. en ℓ').
- $S(c, v_0, v)$ 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;
```