UNIVERSITÉ DE LORRAINE

TELECOM nancy
Ingénieurs du numérique • Inspiring your digital future

MVSI Course
Modelling and Verification of Computer Systems

Lecture 3 – Modelling, Specification and Verification (III)

Dominique Méry
Telecom Nancy, Université de Lorraine
November 20, 2026

Plan

1. Examples of partial correctness (simple assignment)
2. Annotation and tool-supported verification with TLA/TLA$^+$
   - Verification with TLA and its tools
3. Conclusion and limits

Part 1

Examples of partial correctness (simple assignment)

A simple contract (I)

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

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

List of conditions to check for proving (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) and (II) are equivalent and (II) is the definition of the invariance of $A(x_0, x) \stackrel{def}{=} (x_0 = (0, v_0) \wedge x = (f, v) \Rightarrow post(v_0, v))$.

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

A simple contract (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

List of conditions to check for proving (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) and (II) are equivalent and (II) is the definition of the invariance of $A(x_0, x) \stackrel{def}{=} (x = (f, v) \Rightarrow post(v_0, v))$.

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

(II)

Part 2

Annotation and tool-supported verification with TLA/TLA$^+$

Annotation: the steps of the method

1. Define the precondition **pre**$(\mathrm{P})(v_0)$.
2. Annotate the program with predicates $P_\ell(v_0, v)$, where $\ell \in$ Locations.
3. Check that **pre**$(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$, where $\ell \in$ Inputs (the set of entry points).
4. Check that $P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$, where $\ell \in$ Locations.
5. For each pair of control points (ℓ, ℓ') such that $\ell \longrightarrow \ell'$ (consecutive points), check that $\big(P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')\big)$.



The method is justified by the relational induction principle (lecture 2).

Annotation: the steps of the method

1. Define the precondition **pre**$(\mathrm{P})(v_0)$.
2. Annotate the program with predicates $P_\ell(v_0, v)$, where $\ell \in$ Locations.
3. Check that **pre**$(\mathrm{P})(v_0) \wedge v = v_0 \Rightarrow P_\ell(v_0, v)$, where $\ell \in$ Inputs (the set of entry points).
4. Check that $P_\ell(v_0, v) \Rightarrow \mathrm{A}(\ell_0, v_0, \ell, v)$, where $\ell \in$ Locations.
5. For each pair of control points (ℓ, ℓ') such that $\ell \longrightarrow \ell'$ (consecutive points), check that $\big(P_\ell(v_0, v) \wedge cond_{\ell,\ell'}(v) \wedge v' = f_{\ell,\ell'}(v) \Rightarrow P_{\ell'}(v_0, v')\big)$.



The method is justified by the relational induction principle (lecture 2).

Examples of safety properties

- Partial correctness: $\mathrm{A}_1(\ell_0, v_0, \ell, v) \stackrel{def}{=} \ell = \ell_f \Rightarrow$ **post**$(\mathrm{P})(v_0, v)$
- Absence of run-time errors:
  $\mathrm{A}_2(\ell_0, v_0, \ell, v) \stackrel{def}{=} \wedge_{\ell', \ell \to \ell'}$ **DOM**$(\ell, \ell')(v)$

Mechanising and automating the verification

- The verifications are long and numerous.
- The verifications are often elementary and rather easy to prove.
- Approach by algorithmic verification, with TLA and its tools.
- Approach by mechanisation of symbolic reasoning, with Event-B and its tools.

Translating the annotations

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

Translating the annotations

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

- Annotating the code
- Translating the invariant to check
- Expressing the partial correctness property
- Checking the property

Translating the annotations

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

- Annotating the code
- Translating the invariant to check
- Expressing the partial correctness property
- Checking the property

```
---------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
==============================================================
```

Verification with TLAPS and TLA/TLA$^+$: the actions

- The program or the algorithm is annotated at control points $\ell \in \textsc{Locations}$, and at each control point ℓ there is an assertion $P_\ell(v_0, v)$.

Verification with TLAPS and TLA/TLA$^+$: the actions

- The program or the algorithm is annotated at control points $\ell \in \text{LOCATIONS}$, and at each control point ℓ there is an assertion $P_\ell(v_0, v)$.
- If the two control points ℓ, ℓ' define an elementary computation, then we define an action $\mathcal{E}(\ell, \ell')$ as follows:

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

Verification with TLAPS and TLA/TLA$^+$: the actions

- The program or the algorithm is annotated at control points $\ell \in \text{LOCATIONS}$, and at each control point ℓ there is an assertion $P_\ell(v_0, v)$.
- If the two control points ℓ, ℓ' define an elementary computation, then we define an action $\mathcal{E}(\ell, \ell')$ as follows:

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

- v is the state memory variable or the list of the state memory variables; v includes the local variables and the result variables.
- c is a new variable which models the control flow; its type is LOCATIONS.
- $\mathcal{E}(\ell, \ell')$ simulates the computation starting at ℓ and ending at ℓ'; v is updated.

Verification with TLAPS and TLA/TLA$^+$: the invariant

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

Verification with TLAPS and TLA/TLA$^+$: the invariant

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

- $Type$ is the type of the variables v; it is a set of possible values.
- The annotation gives for free the conditions satisfied by v when the control is at ℓ (resp. at ℓ').
- $S(c, v_0, v)$ is a safety property to check; in the case of *Event-B*, it is a theorem.

Exhaustive or algorithmic verification method

Exhaustive or algorithmic verification method

- The transition relation *Next* is defined by:

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

Exhaustive or algorithmic verification method

- The transition relation *Next* is defined by:

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

- The initial conditions of the variables are defined by a predicate *Init*.

Part 3

Conclusion and limits

Summary of the results

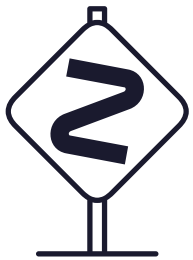
$$\mathcal{D}, \textsc{Alg} \quad \text{SATISFIES} \quad \begin{cases} \textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v) \end{cases}$$

$$\frac{\mathcal{D}}{\begin{array}{l}\textbf{pre}(\textsc{Alg})(v) \\ \textbf{post}(\textsc{Alg})(v_0, v)\end{array}}{\textsc{Alg}}$$

Summary of the results

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

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



Checking the verification conditions

Summary of the results

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

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



Checking the verification conditions

Checking the verification conditions with a model-checker, by exploring all the states.

Summary of the results

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

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



Checking the verification conditions

- Checking the verification conditions with a model-checker, by exploring all the states.
- Checking the verification conditions with a formal proof tool.

Observations

Observations

- Checking statements of the form $\Gamma \vdash P$ (sequents).

Observations

- Checking statements of the form $\Gamma \vdash P$ (sequents).
- Stating or computing the invariants of a model: $\textsc{reachable}(M)$.

Observations

- Checking statements of the form $\Gamma \vdash P$ (sequents).
- Stating or computing the invariants of a model: $\textsc{reachable}(M)$.
- TLA$^+$ versus Event-B
  - Platforms: TLA$^+$ with TLAPS and the Toolbox, Event-B with Rodin.
  - Language of set theory, with a few differences.
  - Functionalities of the tools:
    - model editors: TLA$^+$ and Event-B;
    - model checking: TLA$^+$ and Event-B;
    - proof assistant: Event-B;
    - animator and model checker ProB.

Observations

- Checking statements of the form $\Gamma \vdash P$ (sequents).
- Stating or computing the invariants of a model: $\textsc{reachable}(M)$.
- TLA$^+$ versus Event-B
  - Platforms: TLA$^+$ with TLAPS and the Toolbox, Event-B with Rodin.
  - Language of set theory, with a few differences.
  - Functionalities of the tools:
    - model editors: TLA$^+$ and Event-B;
    - model checking: TLA$^+$ and Event-B;
    - proof assistant: Event-B;
    - animator and model checker ProB.
- Development of symbolic tools, such as SMT solvers or decision procedures.

Review of the tools

- TLA$^+$ and the TLA Toolbox: temporal logic, set theory, predicate calculus, model checker.
- Event-B and Rodin: set theory, proof assistant, model checker, animator.
- B, Event-B and ProB: set theory, model checker, animator, validation.
- Promela and SPIN: temporal logic, model checking.
- C and Frama-C: semantic analysis of programs, proof assistants, SMT solvers.
- Spec# and Rise4fun: pre/post, contracts.
- PAT: generic framework for building one's own model checker (classical, real-time, probabilistic, stochastic).
- C and `cppcheck`: static analysis of C or C++ programs.

Where we stand

- $\forall x_0, x \in \textsc{Vals}. Init(x_0) \wedge \textsc{Next}^\star(x_0, x) \Rightarrow A(x)$
- $\forall x \in \textsc{Vals}. (\exists x_0. x_0 \in \textsc{Vals} \wedge Init(x_0) \wedge \textsc{Next}^\star(x_0, x)) \Rightarrow A(x)$.
- $\textsc{reachable}(M) = \{u | u \in \textsc{Vals} \wedge (\exists x_0. x_0 \in \textsc{Vals} \wedge Init(x_0) \wedge \textsc{Next}^\star(x_0, u))\}$ is the set of states reachable from the initial states.
- Model checking: we must show the inclusion $\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$.
- Proofs: define an invariant $\mathrm{I}(\ell, v) \equiv \bigvee_{\ell \in \textsc{Locations}} \left(\bigvee_{v \in \textsc{Memory}} P_\ell(v) \right)$ with the family of annotations $\{P_\ell(v) : \ell \in \textsc{Locations}\}$, and prove the verification conditions.
- Automatic analysis: see the next frame.

Automatic analysis

- Mechanising the checking of the verification conditions.
- Computing $\textsc{reachable}(M)$ as a fixed point.
- Computing an approximation of $\textsc{reachable}(M)$.

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

If A satisfies $\gamma(A) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$, then $\textsc{reachable}(M) \subseteq \{u | u \in \textsc{Vals} \wedge A(u)\}$.

Summary of concepts

