Semantics and Verifications

Based on the lectures of Colin Riba
Notes written by Hugo Salou

September 8, 2025

Contents

1 Introduction.

Let us give precisions on the terms in the name of the course, and in the broader space of semantics and verifications.

Verification. Formal techniques to ensure the correctness software or hardware of systems.

Model Checking. "Automatic" checking of the correctness by means of exhaustive exploration.

Example 1.1. Consider a program that is 10 lines long, contains 3 booleans variables and 5 integers variables in the range $\{0, \dots, 9\}$. The number of states for this program is:

$$10 \times 2^3 \times 10^5 = 8\,000\,000.$$

The real issue with the state exploration problem is the factor 10^5, coming from the use of 5 integers.

Example 1.2. Consider a server and n clients. Clients can make requests to the server and the server can answer a client. The specification of this server should include the following:

- ▷ Each client which makes a request is eventually answered.
- ▷ We *abstract* away from precise quantitative constraints.

We will sometimes reason about an infinite amount of executions. For example, if some client makes infinitely-many requests (then it'll have infinitely-many answers). Infinite sequences are represented by ω-*words*, *i.e.* infinite words indexed by $\mathbb{N}$. Thus, ω-words on some

alphabet Σ are functions $\mathbb{N} \to \Sigma$. We will denote Σ^ω the set of those infinite words on the alphabet Σ.

If $|\Sigma| \geq 2$, then the set Σ^ω is uncountable.

This course will cover the following:

- Transition systems;
- Linear-time properties;
- Topology;
- Orders and Lattices;
- Linear Temporal Logic (LTL);
- Büchi automata;
- **Stone duality** (mostly in homework);
- Bisimilarity/bisimulation;
- Modal Logic.

Ressources from this course include:

- the course notes (available online, non-exhaustive);
- Baier, C. and Katoen, J.-P., Principles of Model Checking, MIT Press, 2008.

Prerequisites for this course include:

- First-order logic (*see* my course notes for the "Logique" L3 course, in french);
- Finite automata ("FDI" L3 course).

Evaluation for this course will be in two parts: the final exam (50 %) and the homework, in two parts (25 % each).

The tutorials will be done by Lison Blondeau-Patissier.

2 Transition systems.

2.1 Transition systems.

Definition 2.1. A transition system is a tuple

$$TS = (S, \text{Act}, \rightarrow, I, \text{AP}, L)$$

where

- S is the set of *states*;
- Act is the set of *actions*;
- $\rightarrow \subseteq S \times \text{Act} \times S$ the *transition relation*;
- $I \subseteq S$ the set of *initial states*;
- AP is the set of *atomic propositions*;
- $L : S \to \wp(\text{AP}) \cong \mathbf{2}^{\text{AP}}$ is the *state labelling function*.

We will write $s \xrightarrow{\alpha} s'$ when $(s, \alpha, s') \in \rightarrow$.

Example 2.1 (Beverage Vending Machine, BVM). We can model a beverage vending machine using a diagram like in figure 2.1. Here we have that:

- $S = \{\mathsf{pay}, \mathsf{select}, \mathsf{soda}, \mathsf{beer}\}$,
- $I = \{\mathsf{pay}\}$,
- $\text{Act} = \{\mathtt{ic}, \tau, \mathtt{gb}, \mathtt{gs}\}$.[1]

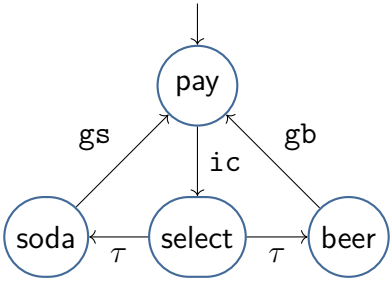

Figure 2.1 | *Transition system for the BVM*

We can define the labels:

$L(\mathsf{pay}) = \emptyset \quad L(\mathsf{soda}) = L(\mathsf{beer}) = \{\mathsf{paid}, \mathsf{drink}\} \quad L(\mathsf{select}) = \{\mathsf{paid}\},$

with $\mathrm{AP} = \{\mathsf{paid}, \mathsf{drink}\}$.

2.2 Program graphs.

The goal is to represent the evaluation of a program.

Definition 2.2 (Typed variables). ▷ A set Var of *variables*.

- ▷ For each variable $x \in \mathrm{Var}$, consider a set $\mathrm{Dom}(x)$.
- ▷ Given $TV = (\mathrm{Var}, (\mathrm{Dom}(x))_{x \in \mathrm{Var}})$, we define

$$\mathrm{Eval}(TV) = \prod_{x \in \mathrm{Var}} \mathrm{Dom}(x),$$

the set of valuations of the form $\eta : x \in \mathrm{Var} \mapsto \eta(x) \in \mathrm{Dom}(x)$.

[1]The meaning of the actions are the following: `ic` means *insert coin*, `gb` means *get beer* and `gs` for *get soda*.

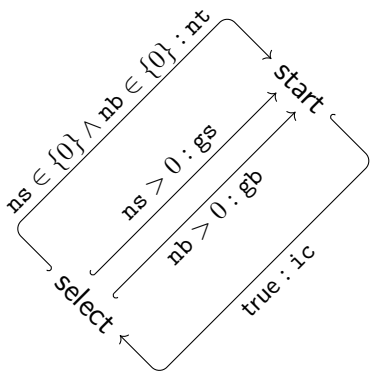

Figure 2.2 | *BVM as a program graph*

Definition 2.3 (Program graph)**.** A *program graph* is a tuple

$$PG = (\text{Loc}, \text{Act}, \text{Effect}, \hookrightarrow, \text{Loc}_0, g_0),$$

where

- Loc is the set of *locations* (lines of codes);
- *Act* is the set of *actions*;
- $\text{Effect} : \text{Act} \times \text{Eval}(TV) \to \text{Eval}(TV)$;
- $\hookrightarrow \subseteq \text{Loc} \times \text{Conditions} \times \text{Act} \times \text{Loc}$ where conditions are propositional formula built from atoms of the forms "$x \in D$" for some variable x and some set $D \subseteq \text{Dom}(x)$;
- $\text{Loc}_0 \subseteq \text{Loc}$ the set of initial locations;
- g_0 is the *initial condition*.

We will write $\ell \xhookrightarrow{g:\alpha} \ell'$ for $(\ell, g, \alpha, \ell') \in \hookrightarrow$.

Example 2.2 (BVM as a program graph)**.** In figure 2.2, we use

- $\text{Loc} = \{\mathsf{start}, \mathsf{select}\}$;

- ▷ $\text{Var} = \{\texttt{ns}, \texttt{nb}\}$;
- ▷ $\text{Act} = \{\texttt{ic}, \texttt{nt}, \texttt{gs}, \texttt{gb}, \texttt{refill}\}$;
- ▷ $\text{Loc}_0 = \{\textsf{start}\}$;
- ▷ $g_0 = \texttt{ns} \in \{\max\} \wedge \texttt{nb} \in \{\max\}$
- ▷

$$\begin{aligned} \text{Effect} : \text{Act} \times \text{Eval}(TV) &\longrightarrow \text{Eval}(TV) \\ (\texttt{refill}, \eta) &\longmapsto [\texttt{ns} \mapsto \max, \texttt{nb} \mapsto \max] \\ (\texttt{gs}, \eta) &\longmapsto \eta[\texttt{ns} \mapsto \eta(\texttt{ns}) - 1] \\ (\texttt{gb}, \eta) &\longmapsto \eta[\texttt{nb} \mapsto \eta(\texttt{nb}) - 1] \end{aligned}$$

2.3 Transition system of a program graph.

Definition 2.4. Given TV and PG a program graph, we define

$$TS(PG) := (S, \text{Act}, \to, I, \text{AP}, L)$$

where

- ▷ $S = \text{Loc} \times \text{Eval}(TV)$;
- ▷ $\text{AP} = \text{Loc} \cup \text{Conditions}$;
- ▷ $I = \{(\ell_0, \eta) \mid \ell_0 \in \text{Loc}_0, \eta \models g_0\}$;
- ▷ $\to$ is defined by:

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \qquad \eta \models g}{(\ell, \eta) \overset{\alpha}{\to} (\ell', \text{Effect}(\alpha, \eta))},$$

- ▷ and $L(\ell, \eta) = \{\ell\} \cup \{g \mid \eta \models g\}$.

Example 2.3. The BVM program graph example seen in the previous example can be transformed as a transition system thanks to the previous definition; it is shown in figure 2.3. To simplify, we assume max = 1.

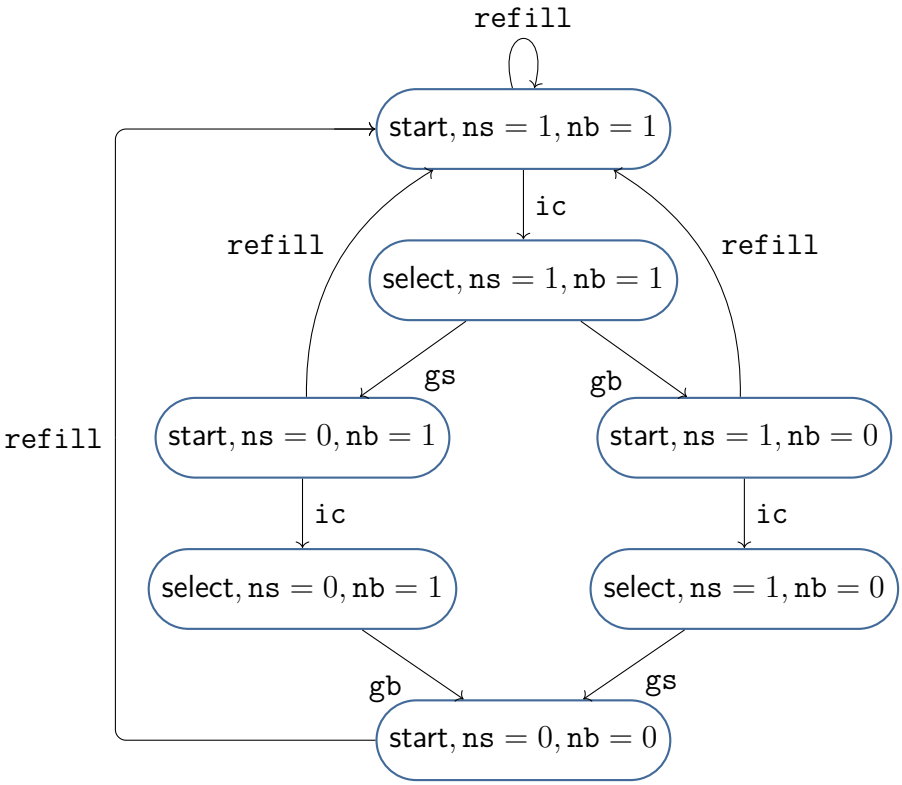

Figure 2.3 | *Transition system of the PVM program graph*