Model Checking I
alias
Reactive Systems Verification

Luca Tesei

MSc in Computer Science, University of Camerino

Topics

- Guarded Commands Language
- Syntax and Intuitive semantics of `nanoPromela`
- Examples

Material

Reading:

Chapter 2 of the book, pages 63–68.

More:

The slides in the following pages are taken from the material of the course "Introduction to Model Checking" held by Prof. Dr. Ir. Joost-Pieter Katoen at Aachen University.

Guarded Command Language (GCL)

by Dijkstra

Guarded Command Language (GCL)

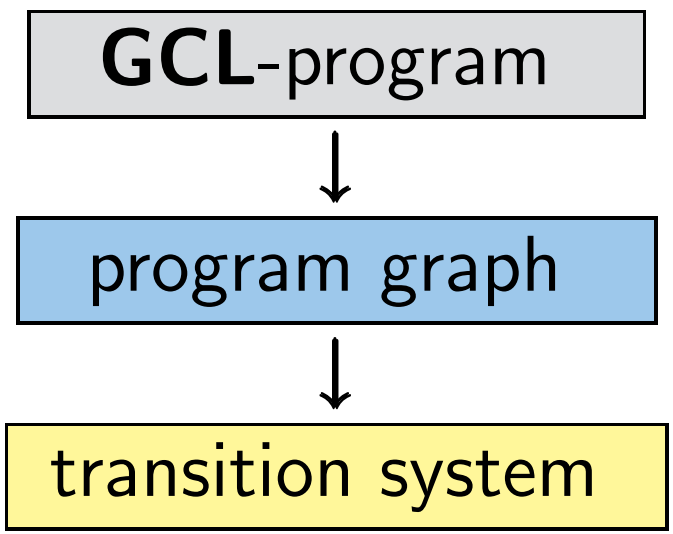
by Dijkstra

- high-level modeling language that contains features of imperative languages and nondeterministic choice

Guarded Command Language (GCL)

by Dijkstra

- high-level modeling language that contains features of imperative languages and nondeterministic choice
- semantics:



Guarded Command Language (GCL)

guarded command $g \Rightarrow \mathit{stmt}$

g	:	guard, i.e., Boolean condition on the program variables
stmt	:	statement

Guarded Command Language (GCL)

guarded command $g \Rightarrow$ ***stmt*** $\leftarrow$ enabled if g is true

g	:	guard, i.e., Boolean condition on the program variables
stmt	:	statement

Guarded Command Language (GCL)

guarded command $g \Rightarrow \mathit{stmt}$ ← enabled if g is true

g	:	guard, i.e., Boolean condition on the program variables
stmt	:	statement

repetitive command/loop:

`DO` :: $g \Rightarrow \mathit{stmt}$ `OD`

Guarded Command Language (GCL)

guarded command $g \Rightarrow$ ***stmt*** $\leftarrow$ enabled if g is true

g	:	guard, i.e., Boolean condition on the program variables
stmt	:	statement

repetitive command/loop:

`DO` :: $g \Rightarrow$ ***stmt*** `OD` $\leftarrow$ `WHILE` g `DO` ***stmt*** `OD`

Guarded Command Language (GCL)

guarded command $g \Rightarrow \mathit{stmt}$ $\leftarrow$ enabled if g is true

g	: guard, i.e., Boolean condition on the program variables
stmt	: statement

repetitive command/loop:

$\texttt{DO} :: g \Rightarrow \mathit{stmt}\ \texttt{OD}$ $\leftarrow$ $\texttt{WHILE}\ g\ \texttt{DO}\ \mathit{stmt}\ \texttt{OD}$

conditional command:

$$\begin{aligned}&\texttt{IF} :: g \Rightarrow \mathit{stmt}_1\\ &\quad :: \neg g \Rightarrow \mathit{stmt}_2\\ &\texttt{FI}\end{aligned}$$

Guarded Command Language (GCL)

guarded command $g \Rightarrow \mathit{stmt}$ $\leftarrow$ enabled if g is true

g	: guard, i.e., Boolean condition on the program variables
stmt	: statement

repetitive command/loop:

$\texttt{DO} :: g \Rightarrow \mathit{stmt}\ \texttt{OD}$ $\leftarrow$ `WHILE` g `DO` stmt `OD`

conditional command:

$$\texttt{IF} :: g \Rightarrow \mathit{stmt}_1 \;\; :: \neg g \Rightarrow \mathit{stmt}_2 \;\; \texttt{FI}$$

$\leftarrow$

```
IF g THEN stmt1
   ELSE stmt2
FI
```

Guarded Command Language (GCL)

guarded command $g \Rightarrow$ ***stmt*** $\leftarrow$ enabled if g is true

repetitive command/loop:

DO :: $g \Rightarrow$ ***stmt*** OD $\leftarrow$ WHILE g DO ***stmt*** OD

conditional command:

IF :: $g \Rightarrow$ ***stmt***$_1$
:: $\neg g \Rightarrow$ ***stmt***$_2$
FI

$\leftarrow$

IF g THEN ***stmt***$_1$
ELSE ***stmt***$_2$
FI

symbol :: stands for the nondeterministic choice between enabled guarded commands

Guarded Command Language (GCL)

modeling language with nondeterministic choice

$$
\begin{array}{lll}
\mathit{stmt} & \stackrel{\text{def}}{=} & x := \mathit{expr} \;\mid\; \mathit{stmt}_1 ; \mathit{stmt}_2 \;\mid \\
& & \texttt{DO} \;::g_1 \Rightarrow \mathit{stmt}_1 \;\ldots\; ::g_n \Rightarrow \mathit{stmt}_n \;\texttt{OD} \\
& & \texttt{IF} \;::g_1 \Rightarrow \mathit{stmt}_1 \;\ldots\; ::g_n \Rightarrow \mathit{stmt}_n \;\texttt{FI} \\
& & \vdots
\end{array}
$$

where x is a typed variable and expr an expression of the same type

Guarded Command Language (GCL)

modeling language with nondeterministic choice

$$\begin{aligned} \mathit{stmt} \stackrel{\text{def}}{=} \quad & x := \mathit{expr} \mid \mathit{stmt}_1 ; \mathit{stmt}_2 \mid \\ & \texttt{DO} :: g_1 \Rightarrow \mathit{stmt}_1 \ \ldots \ :: g_n \Rightarrow \mathit{stmt}_n \ \texttt{OD} \\ & \texttt{IF} :: g_1 \Rightarrow \mathit{stmt}_1 \ \ldots \ :: g_n \Rightarrow \mathit{stmt}_n \ \texttt{FI} \\ & \vdots \end{aligned}$$

where x is a typed variable and expr an expression of the same type

semantics of a **GCL**-program: program graph

GCL-program for beverage machine

GCL-program for beverage machine

uses two variables $\#sprite, \#coke \in \{0, 1, \ldots, max\}$
for the number of available drinks (sprite or coke)

GCL-program for beverage machine

uses two variables $\#sprite, \#coke \in \{0, 1, \ldots, max\}$
for the number of available drinks (sprite or coke)

uses the following actions:

	enabled	effect
get_coke	if $\#coke > 0$	$\#coke := \#coke - 1$
get_sprite	if $\#sprite > 0$	$\#sprite := \#sprite - 1$

GCL-program for beverage machine

uses two variables $\#sprite, \#coke \in \{0, 1, \ldots, max\}$
for the number of available drinks (sprite or coke)

uses the following actions:

	enabled	effect
get_coke	if $\#coke > 0$	$\#coke := \#coke - 1$
get_sprite	if $\#sprite > 0$	$\#sprite := \#sprite - 1$
refill	any time	$\#sprite := max$ $\#coke := max$

GCL-program for beverage machine

uses two variables $\#sprite, \#coke \in \{0, 1, \ldots, max\}$
for the number of available drinks (sprite or coke)

uses the following actions:

	enabled	effect
get_coke	if $\#coke > 0$	$\#coke := \#coke - 1$
get_sprite	if $\#sprite > 0$	$\#sprite := \#sprite - 1$
refill	any time	$\#sprite := max$ $\#coke := max$
insert_coin	any time	no effect on variables

GCL-program for beverage machine

uses two variables $\#sprite, \#coke \in \{0, 1, \ldots, max\}$
for the number of available drinks (sprite or coke)

uses the following actions:

	enabled	effect
get_coke	if $\#coke > 0$	$\#coke := \#coke - 1$
get_sprite	if $\#sprite > 0$	$\#sprite := \#sprite - 1$
refill	any time	$\#sprite := max$ $\#coke := max$
insert_coin	any time	no effect on variables
return_coin	if machine is empty and user has entered a coin (no effect on variables)	

GCL-program for beverage machine

DO :: true ⇒ insert_coin;

IF :: $\#sprite = \#coke = 0 \Rightarrow$ return_coin

:: $\#coke > 0 \Rightarrow \#coke := \#coke - 1$

:: $\#sprite > 0 \Rightarrow \#sprite := \#sprite - 1$

FI

:: true ⇒ $\#sprite := max; \#coke := max$

OD

GCL-program for beverage machine


```
DO  ::  true ⇒ insert_coin; (* user inserts a coin *)
        IF  ::  #sprite = #coke = 0 ⇒ return_coin
            ::  #coke  > 0 ⇒ #coke := #coke − 1
            ::  #sprite > 0 ⇒ #sprite := #sprite−1
        FI
    ::  true ⇒ #sprite := max; #coke := max
OD
```

GCL-program for beverage machine


```
DO  ::  true ⇒ insert_coin; (* user inserts a coin *)
        IF  ::  #sprite = #coke = 0 ⇒ return_coin
                                  (* no beverage available *)
            ::  #coke  > 0 ⇒ #coke := #coke − 1

            ::  #sprite > 0 ⇒ #sprite := #sprite−1

        FI
    ::  true ⇒ #sprite := max; #coke := max
OD
```

GCL-program for beverage machine


```
DO  ::  true ⇒ insert_coin; (* user inserts a coin *)
        IF  ::  #sprite = #coke = 0 ⇒ return_coin
                                (* no beverage available *)
            ::  #coke  > 0 ⇒ #coke := #coke − 1
                                (* user selects coke *)
            ::  #sprite > 0 ⇒ #sprite := #sprite−1
                                (* user selects sprite *)
        FI
    ::  true ⇒ #sprite := max; #coke := max
                                (* refilling of the machine *)
OD
```

GCL-program for beverage machine


```
DO  ::  true ⇒ insert_coin; (* user inserts a coin *)
        IF  ::  #sprite = #coke = 0 ⇒ return_coin
                                  (* no beverage available *)
            ::  #coke  > 0 ⇒ get_coke
                                  (* user selects coke *)
            ::  #sprite > 0 ⇒ get_sprite
                                  (* user selects sprite *)
        FI
    ::  true ⇒ refill
                                  (* refilling of the machine *)
OD
```

GCL-program for beverage machine


```
DO  ::  true ⇒ insert_coin;
        IF  ::  #sprite = #coke = 0 ⇒ return_coin
            ::  #coke  > 0 ⇒ get_coke
            ::  #sprite > 0 ⇒ get_sprite
        FI
    ::  true ⇒ refill
OD
```

GCL-program for beverage machine


```
DO  ::  true ⇒ insert_coin;
        IF  ::  #sprite = #coke = 0
                                ⇒ return_coin
            ::  #coke  > 0 ⇒ get_coke
            ::  #sprite > 0 ⇒ get_sprite
        FI
    ::  true ⇒ refill
OD
```

... yields a program graph with

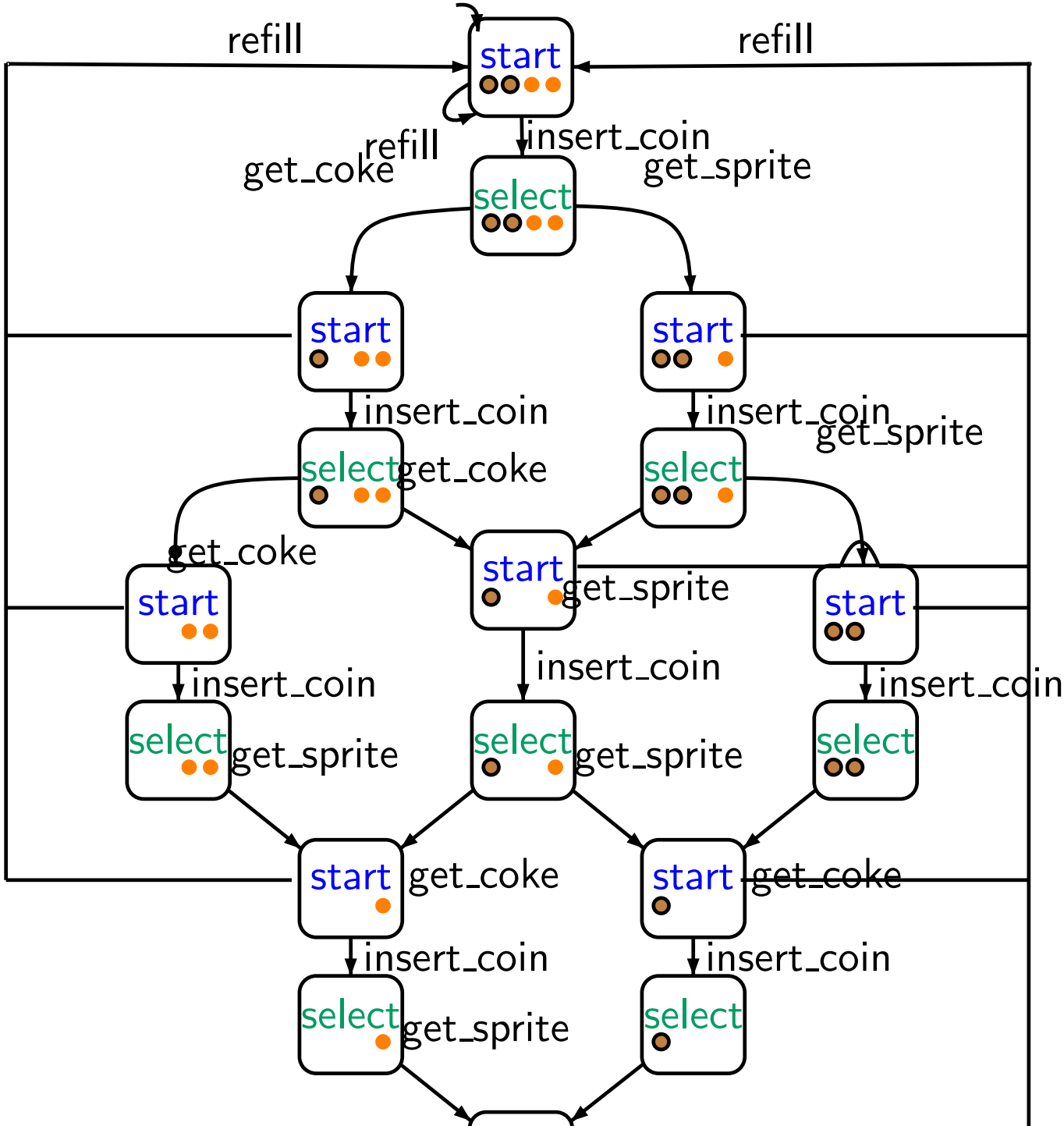
- two variables $\#sprite, \#coke \in \{0, 1, \ldots, max\}$

GCL-program for beverage machine

$$
\begin{array}{lll}
\textit{start} \longrightarrow \texttt{DO} & :: & \text{true} \Rightarrow \text{insert\_coin}; \\
\textit{select} \longrightarrow \quad\quad \texttt{IF} & :: & \#\textit{sprite} = \#\textit{coke} = 0 \\
 & & \qquad\qquad \Rightarrow \text{return\_coin} \\
 & :: & \#\textit{coke} > 0 \Rightarrow \text{get\_coke} \\
 & :: & \#\textit{sprite} > 0 \Rightarrow \text{get\_sprite} \\
\quad\quad \texttt{FI} & & \\
 :: & \text{true} \Rightarrow \text{refill} & \\
\texttt{OD} & &
\end{array}
$$

... yields a program graph with

- two variables $\#\textit{sprite}, \#\textit{coke} \in \{0, 1, \ldots, \textit{max}\}$
- two locations ***start*** and ***select***


refill
start
refill
refill
insert_coin
get_coke
get_sprite
select
start
start
insert_coin
insert_coin
get_sprite
select
get_coke
select
get_coke
start
get_sprite
start
start
insert_coin
insert_coin
insert_coin
select
get_sprite
select
get_sprite
select
start
get_coke
start
get_coke
insert_coin
insert_coin
select
get_sprite
select