CML tutorial

Incorporating the Dwarf Signal Example

Simon Foster Jim Woodcock

University of York

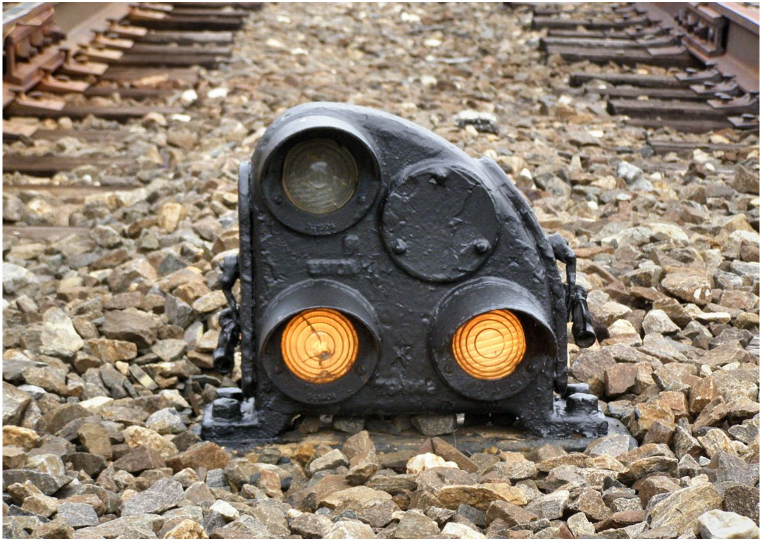
February 14, 2013

Outline

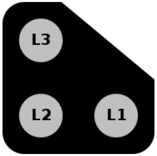
CML Introduction

- a formal language for specifying Systems of Systems
- draws input from formal languages VDM and Circus
- a CML consists of
  - types with invariants, e.g.
    - basic types: bool, int, string, real etc.
    - enumerations ("quote" type)
    - sets
    - maps
    - records
  - functions with pre and postconditions
  - operations which act on a state
  - processes from CSP
- we illustrate these by an example

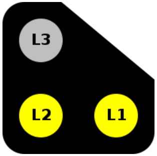
Dwarf Railway Signals



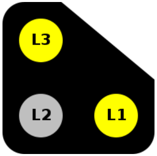
Proper States



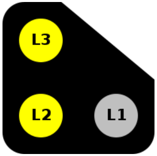
Dark
$\{\}$



Stop
$\{L1, L2\}$



Warning
$\{L1, L3\}$



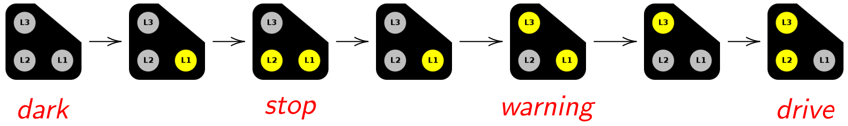
Drive
$\{L2, L3\}$

- Other (transient) states: $\{L1\}, \{L2\}, \{L3\}, \{L1, L2, L3\}$

Safety Requirements

- Only one lamp may be changed at once
- All three lamps must never be on concurrently
- The signal must never be dark except if the dark aspect has to be shown or there is lamp failure
- The change to and from dark is allowed only from stop and to stop

Typical Trace



Outline

Dwarf Signal basic types in CML

```
types
  LampId      = <L1> | <L2> | <L3>
  Signal      = set of LampId
  ProperState = Signal
    inv ps == ps in set {dark, stop, warning, drive}

values
  dark: Signal     = {}
  stop: Signal     = {<L1>, <L2>}
  warning: Signal = {<L1>, <L3>}
  drive: Signal    = {<L2>, <L3>}
```

Dwarf Signal State

```
types
  DwarfType :: lastproperstate     : ProperState
               desiredproperstate : ProperState
               turnoff            : set of LampId
               turnon             : set of LampId
               laststate          : Signal
               currentstate       : Signal
```

Dwarf Signal State

```
types
  DwarfType :: lastproperstate     : ProperState
               desiredproperstate : ProperState
               turnoff            : set of LampId
               turnon             : set of LampId
               laststate          : Signal
               currentstate       : Signal
```

- the previous/current proper state the signal was in

Dwarf Signal State

```
types
  DwarfType :: lastproperstate     : ProperState
               desiredproperstate  : ProperState
               turnoff             : set of LampId
               turnon              : set of LampId
               laststate           : Signal
               currentstate        : Signal
```

- the proper state we desire to reach

Dwarf Signal State

```
types
  DwarfType :: lastproperstate    : ProperState
               desiredproperstate : ProperState
               turnoff            : set of LampId
               turnon             : set of LampId
               laststate          : Signal
               currentstate       : Signal
```

- lamps we need to turn off to reach the desired proper state

Dwarf Signal State

```
types
  DwarfType :: lastproperstate    : ProperState
               desiredproperstate : ProperState
               turnoff            : set of LampId
               turnon             : set of LampId
               laststate          : Signal
               currentstate       : Signal
```

- lamps we need to turn on to reach the desired proper state

Dwarf Signal State

```
types
  DwarfType :: lastproperstate     : ProperState
               desiredproperstate : ProperState
               turnoff            : set of LampId
               turnon             : set of LampId
               laststate          : Signal
               currentstate       : Signal
```

- the actual last state the signal was in

Dwarf Signal State

```
types
  DwarfType :: lastproperstate     : ProperState
               desiredproperstate : ProperState
               turnoff            : set of LampId
               turnon             : set of LampId
               laststate          : Signal
               currentstate       : Signal
```

- the actual current state the signal is in

Dwarf Signal State - Invariants

```
inv d ==
  (((d.currentstate \ d.turnoff) union d.turnon)
        = d.desiredproperstate)
```

- desired state = (current state - lamps to off) + lamps to on

Dwarf Signal State - Invariants

```
inv d ==
  (((d.currentstate \ d.turnoff) union d.turnon)
        = d.desiredproperstate)
  and
  (d.turnoff inter d.turnon = {})
```

- we can't simultaneously desire to turn a light on and off

Dwarf Signal State

```
types
  DwarfType :: lastproperstate    : ProperState
               desiredproperstate : ProperState
               turnoff            : set of LampId
               turnon             : set of LampId
               laststate          : Signal
               currentstate       : Signal
  inv d ==
    (((d.currentstate \ d.turnoff) union d.turnon)
          = d.desiredproperstate)
    and
    (d.turnoff inter d.turnon = {})
```

Outline

Processes in CML

- channels to communicate on, optionally carrying data
- state variables to read and write to
- operations acting on the state, with pre/postconditions
- actions which describe reactive behaviours
- process body, the main behaviour of the process

CML process syntax

Syntax	Description
`Stop`	Deadlocked process
`Skip`	Null behaviour
`a -> P`	Communicate on `a` then behave like `P`
`a?v -> P`	Input value `v` over channel `a` then do `P`
`a!v -> P`	Output value `v` on channel `a` then do `P`
`P ; Q`	Execute process `P` followed by `Q`
`P [] Q`	Pick `P` or `Q` based on the first communication
`P [|{a,b,c}|] Q`	Execute `P` and `Q` in parallel, with synchronisation allowed on `a`, `b` and `c`
`[cond] & P`	allow execution of `P` only if `cond` holds

A basic CML process

```
channels
  a: int
  b: int

process Simple = begin

@

(a?v -> b!(v * 2) -> Skip) [|a|] (a!5 -> Skip)

end
```

Basic process behaviour

$$\begin{tikzcd}
\texttt{(a?v -> b!(v * 2) -> Skip) [|a|] (a!5 -> Skip)} \arrow[d, "\texttt{a.5}"] \\
\texttt{(b!(v * 2) -> Skip) [|a|] (Skip)} \arrow[d, "\texttt{b.10}"] \\
\texttt{(Skip) [|a|] (Skip)}
\end{tikzcd}$$

Outline

Dwarf Process

```
channels
  init
  light: LampId
  extinguish: LampId
  setPS: ProperState
  shine: Signal

process Dwarf = begin

state
  dw : DwarfType

...

end
```

Init operation

```
operations
  Init : () ==> ()
  Init() ==
    dw := mk_DwarfType(stop, {}, {}, stop, stop, stop)

    post dw.lastproperstate = stop and
         dw.turnoff = {} and
         dw.turnon = {} and
         dw.laststate = stop and
         dw.currentstate = stop and
         dw.desiredproperstate = stop
```

Set New Proper State

```
SetNewProperState: (ProperState) ==> ()
SetNewProperState(st) ==
  dw := mk_DwarfType( dw.currentstate
                    , dw.currentstate \ st
                    , st \ dw.currentstate
                    , dw.laststate
                    , dw.currentstate
                    , st)

  pre dw.currentstate = dw.desiredproperstate and
      st <> dw.currentstate
```

Turn On

```
TurnOn: (LampId) ==> ()
TurnOn(l) ==
  dw := mk_DwarfType( dw.lastproperstate
                    , dw.turnoff \ {l}
                    , dw.turnon \ {l}
                    , dw.currentstate
                    , dw.currentstate union {l}
                    , dw.desiredproperstate)

  pre l in set dw.turnon
```

Turn Off

```
TurnOff : (LampId) ==> ()
TurnOff(l) ==
  dw := mk_DwarfType( dw.lastproperstate
                    , dw.turnoff \ {l}
                    , dw.turnon \ {l}
                    , dw.currentstate
                    , dw.currentstate \ {l}
                    , dw.desiredproperstate)

  pre l in set dw.turnon
```

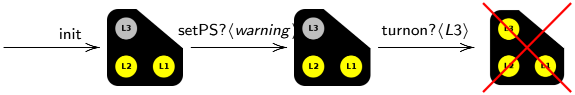
Dwarf Signal Process

```
actions
  DWARF =
    (  (light?l -> TurnOn(l); DWARF)
    [] (extinguish?l -> TurnOff(l); DWARF)
    [] (setPS?l -> SetNewProperState(l); DWARF)
    [] shine!dw.currentstate -> DWARF)

@

init -> Init() ; DWARF
```

Practical: Example Interaction

A bad trace

- not all traces have good results:



- we have violated the safety property:

```
NeverShowAll: DwarfType -> bool
NeverShowAll(d) == d.currentstate <> {<L1>,<L2>,<L3>}
```

The test in CML

```
actions
  ...

  -- Tries to turn on 3 lights simultaneously
  TEST = setPS!warning -> light!<L3> -> extinguish!<L2>
      -> setPS!drive -> extinguish!<L1> -> light!<L2>
      -> Stop

  DWARF_TEST = DWARF [|{setPS,light,extinguish}|] TEST
```

- can be thought of as a counterexample

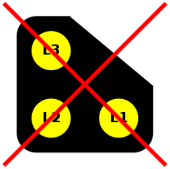
Practical: Represent this

Outline

Safety Properties (1)

- A signal must never show all the lights

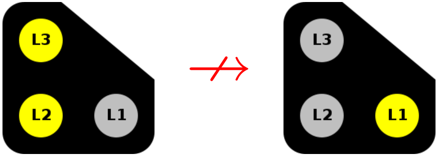
```
functions
  NeverShowAll: DwarfType -> bool
  NeverShowAll(d) == d.currentstate <> {<L1>,<L2>,<L3>}
```



Safety Properties (2)

- Only one lamp at a time may change

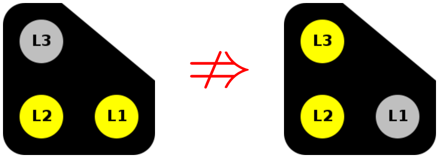
```
MaxOneLampChange: DwarfType -> bool
MaxOneLampChange(d) ==
  card ((d.currentstate \ d.laststate)
       union (d.laststate \ d.currentstate)) <= 1
```



Safety Properties (3)

- The signal may not go straight from stop to drive

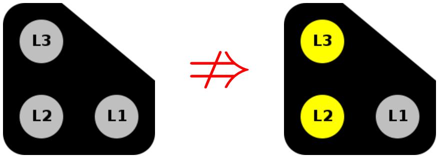
```
ForbidStopToDrive : DwarfType -> bool
ForbidStopToDrive(d) ==
  (d.lastproperstate = stop
   => d.desiredproperstate <> drive)
```



Safety Properties (4)

- the only proper aspect following dark is stop

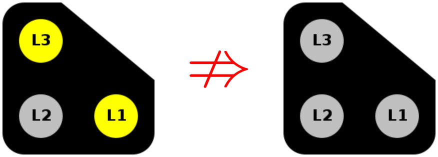
```
DarkOnlyToStop : DwarfType -> bool
DarkOnlyToStop(d) ==
  (d.lastproperstate = dark
  => d.desiredproperstate in set {dark,stop})
```



Safety Properties (5)

- the only proper aspect preceeding dark is stop

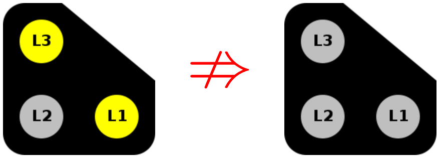
```
DarkOnlyFromStop: DwarfType -> bool
DarkOnlyFromStop(d) == ?
```



Safety Properties (5)

- the only proper aspect preceeding dark is stop

```
DarkOnlyFromStop: DwarfType -> bool
DarkOnlyFromStop(d) ==
  (d.desiredproperstate = dark
  => d.lastproperstate in set {dark,stop})
```



Correct Dwarf Signal Type

```
types
  DwarfSignal = DwarfType
  inv d == NeverShowAll(d) and
           MaxOneLampChange(d) and
           ForbidStopToDrive(d) and
           DarkOnlyToStop(d) and
           DarkOnlyFromStop(d)
```

Practical: 2 more tests