A Domain-Specific Language for Generic Interlocking Models and Their Properties

Linh, H. Vu, Technical University of Denmark
Anne E. Haxthausen, Technical University of Denmark
Jan Peleska, University of Bremen

DTU Compute
Department of Applied Mathematics and Computer Science

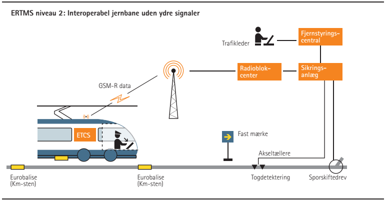
Outline

1. Introduction
   - Background on Interlocking Systems
   - Motivation

2. **IDL:** A Domain-Specific Language for Generic Interlocking Models

3. Conclusions

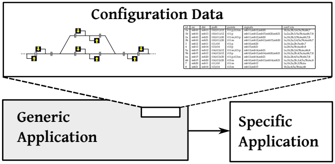
Interlocking Systems



Source: Banedanmark

- An *interlocking system* is a signalling system component responsible for *safe* routing of trains through a railway network.

State-of-the-art Architecture of Interlocking Systems

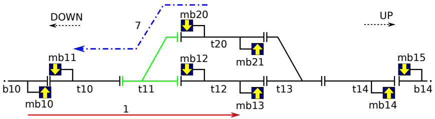


Product line paradigm:

- Interlocking systems come in *product lines*.
- Each product line has its own *generic application* which can be instantiated with *configuration data* to *specific applications* (product instances).
- *Configuration data* is specified by a *track plan* and an *interlocking table*.

Specification of Configuration Data – Example

(1) track plan:

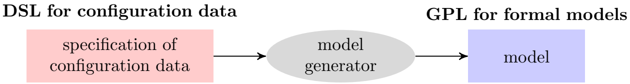


(2) interlocking table:

id	src	dst	path	points	signals	conflicts
1a	mb10	mb13	t10;t11;t12	t11:p;t13:m	mb11;mb12;mb20	1b;2a;2b;3;4;5a;5b;6b;7
1b	mb10	mb13	t10;t11;t12	t11:p	mb11;mb12;mb15;mb20;mb21	1a;2a;2b;3;5a;5b;6a;6b;7;8
2a	mb10	mb21	t10;t11;t20	t11:m;t13:p	mb11;mb12;mb20	1a;1b;2b;3;5b;6a;6b;7;8
2b	mb10	mb21	t10;t11;t20	t11:m	mb11;mb12;mb13;mb15;mb20	1a;1b;2a;3;4;5a;5b;6a;6b;7
3	mb12	mb11	t11;t10	t11:p	mb10;mb20	1a;1b;2a;2b;5a;6b;7
4	mb13	mb14	t13;t14	t13:p	mb15;mb21	1a;2b;5a;5b;6a;6b;8
5a	mb15	mb12	t14;t13;t12	t11:m;t13:p	mb13;mb14;mb21	1a;1b;2b;3;4;5b;6a;6b;8
5b	mb15	mb12	t14;t13;t12	t13:p	mb10;mb13;mb14;mb20;mb21	1a;1b;2a;2b;4;5a;6a;6b;7;8
6a	mb15	mb20	t14;t13;t20	t11:p;t13:m	mb13;mb14;mb21	1b;2a;2b;4;5a;5b;6b;7;8
6b	mb15	mb20	t14;t13;t20	t13:m	mb10;mb12;mb13;mb14;mb21	1a;1b;2a;2b;3;4;5a;5b;6a;8
7	mb20	mb11	t11;t10	t11:m	mb10;mb12	1a;1b;2a;2b;3;5b;6a
8	mb21	mb14	t13;t14	t13:m	mb13;mb15	1b;2a;4;5a;5b;6a;6b

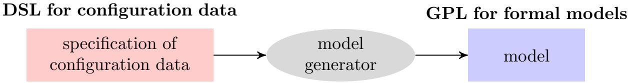
Modelling Interlocking Systems for Verification

- State-of-the-art FMs for interlocking verification provide a *model-generator*:



Modelling Interlocking Systems for Verification

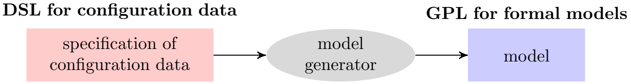
- State-of-the-art FMs for interlocking verification provide a *model-generator*:



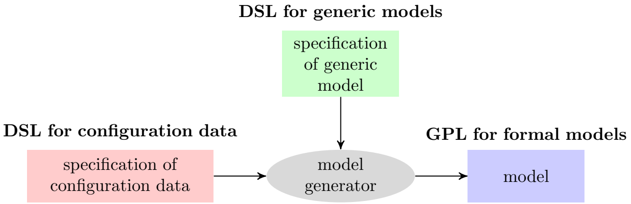
- Inconveniences:
  - A new model generator is needed: (1) for each new product line, (2) when making different model abstractions for the same product line, and (3) when fixing modelling bugs.

Modelling Interlocking Systems for Verification

- State-of-the-art FMs for interlocking verification provide a *model-generator*:



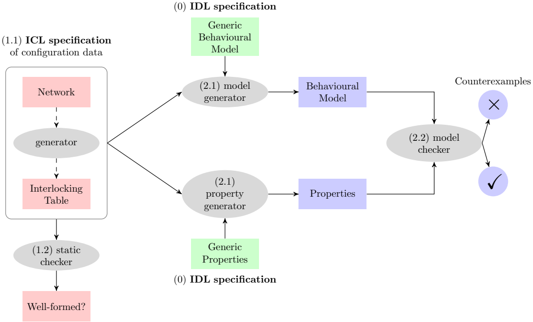
- Inconveniences:
  - A new model generator is needed: (1) for each new product line, (2) when making different model abstractions for the same product line, and (3) when fixing modelling bugs.
- We suggest to let the generator take a 2. argument in **a DSL for generic models**:



Advantages of the extra DSL: (1) Easier to read, write and change generic models. (2) Need only model generator.

RobustRailS Verification Method & Tools

- In the **RobustRailS** project (2012-17) supporting the Danish re-signalling programme, we have a model generator with inputs from *two DSLs*: **IDL** and **ICL**.



For each product line:

- Generic model and properties are defined once-and-for-all in **IDL**.
- Two-step verification for each product instance, (1) configuration data is defined in **ICL** and verified by a static analyser, and (2) models and properties generated and verified by SMT-based model checking (using induction).

For more details on the method and its applications: *Vu, Haxthausen & Peleska: Formal modelling and verification of interlocking systems featuring sequential release.* Science of Comp. Progr., *133, Part 2:91 – 115, 2017.*

Outline

IDL Specifications – Overview

- major specification elements:
  - *generic variable declarations* (encodings)
  - *generic transition relation* definition
  - *generic properties* (state invariants)
  - *macros*

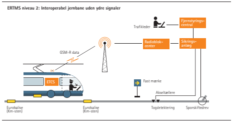
IDL Specifications – Overview

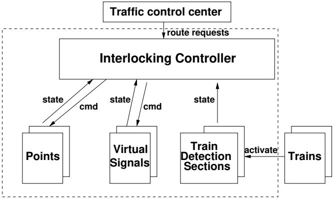
- major specification elements:
  - *generic variable declarations* (encodings)
  - *generic transition relation* definition
  - *generic properties* (state invariants)
  - *macros*
- special domain-specific features supporting genericity:
  - pre-defined *element types* (e.g. **Point**) each representing a set of elements (e.g. points) in the configuration data
  - built-in *domain-specific functions* for generic references to elements in the configuration data(e.g. **first**(r))

IDL Specifications – Overview

- major specification elements:
  - *generic variable declarations* (encodings)
  - *generic transition relation* definition
  - *generic properties* (state invariants)
  - *macros*
- special domain-specific features supporting genericity:
  - pre-defined *element types* (e.g. **Point**) each representing a set of elements (e.g. points) in the configuration data
  - built-in *domain-specific functions* for generic references to elements in the configuration data(e.g. **first**(r))
- semantics: $\mathcal{M} : Specification \rightarrow (ConfigData \rightarrow Model \times Properties)$
  $\mathcal{M}(spec)(cd) = (M, P)$, where
  - $M = (S, I, R)$ is a behavioural model with
    - state space S: a set of variable assignments $\sigma : V \rightarrow Value$ for a set V of variables.
    - initial condition I: a predicate over variables in V
    - transition relation R: a predicate over variable in V (pre states) and V' (post states).
  - P is a predicate over variables in V representing desired state invariants.

Interlocking System Case Study





Traditional control loop:

- The interlocking receives *route requests* from the traffic control center.
- It sets a requested route, if no conflicting route is already set.
- While setting a route it commands points and signals to the settings required for the route (specified in the interlocking table).
- Once a route is set, it commands the entry signal to OPEN.
- Once a train enters the route, it sets the (virtual) signal to CLOSED.
- It releases the route, when the train has finished using it.

Generic Variable Declarations – Example

```
encoding /* generic variable declarations */
  Linear ::
    CNT  → [INPUT,"unsigned int",0,0,2]    /* occupied counter */
  Point ::
    CNT  → [INPUT,"unsigned int",0,0,2]    /* occupied counter */
    POS  → [INPUT,"unsigned int",0,0,1]    /* actual position */
    CMD  → [OUTPUT,"unsigned int",0,0,1]   /* commanded position */
  Signal ::
    ACT  → [INPUT,"unsigned int",0,0,1]    /* actual aspect */
    CMD  → [OUTPUT,"unsigned int",0,0,1]   /* commanded aspect */
  Route ::
    MODE → [LOCAL,"unsigned int",0,0,4]    /* current mode */
```

Instantiation of Generic Variable Declarations

encoding /∗ *generic variable declarations* ∗/

  Linear ::

    CNT → [**INPUT**,"unsigned int",0,0,2] /∗ *occupied counter* ∗/

  Point ::

    CNT → [**INPUT**,"unsigned int",0,0,2] /∗ *occupied counter* ∗/

    POS → [**INPUT**,"unsigned int",0,0,1] /∗ *actual position* ∗/

    CMD → [**OUTPUT**,"unsigned int",0,0,1] /∗ *commanded position* ∗/

  Signal ::

    ACT → [**INPUT**,"unsigned int",0,0,1] /∗ *actual aspect* ∗/

    CMD → [**OUTPUT**,"unsigned int",0,0,1] /∗ *commanded aspect* ∗/

  Route ::

    MODE → [**LOCAL**,"unsigned int",0,0,4] /∗ *current mode* ∗/

gives

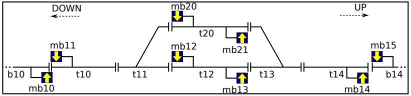
- state space $S \equiv S_{\textsf{Linear}} \times \cdots \times S_{\textsf{Route}}$
- initial condition $I \equiv I_{\textsf{Linear}} \wedge \cdots \wedge I_{\textsf{Route}}$

Instantiation of Generic Variable Declarations

Instantiation of

encoding /∗ *generic variable declarations* ∗/
  Linear ::
    CNT → [**INPUT**,"unsigned int",0,0,2] /∗ *occupied counter* ∗/

with



Linear = { t10, t12, t14, t20 }

gives rise to the following concrete variables:

t10.CNT, t12.CNT, t14.CNT, t20.CNT

all with domain $0..2$ and initial value 0.

$S_{\textsf{Linear}} \equiv \{\texttt{t10.CNT}, \texttt{t12.CNT}, \texttt{t14.CNT}, \texttt{t20.CNT}\} \to \{0..2\}$

$I_{\textsf{Linear}} \equiv \texttt{t10.CNT} = 0 \wedge \texttt{t12.CNT} = 0 \wedge \texttt{t14.CNT} = 0 \wedge \texttt{t20.CNT} = 0$

Macros - Examples

macro /* *signal aspects* */

def CLOSED = 0, **def** OPEN = 1

macro /* *route modes* */

def FREE = 0, ..., **def** LOCKED = 3, **def** OCCUPIED = 4

Generic Transition Relation Definition

General form:

transrel `te`

where `te` is a *(generic) transition relation expression* in one of the forms:

- *atomic transition rule*: $\texttt{guard-expr} \longrightarrow \texttt{update-expr}$
- *non-deterministic choice*: $\texttt{te}_1\ [=]\ \texttt{te}_2$
- *prioritised choice*: $\texttt{te}_1\ [>]\ \texttt{te}_2$
- *quantified transition rule*: $[=]\ \texttt{id : ElementType} \bullet \texttt{te}$

For the running example, the transition rule takes the form

transrel

$$\texttt{te}_{\texttt{route\_dispatcher}}\ [=]\ (\texttt{te}_{\texttt{IXL}}\ [>]\ (\texttt{te}_{\texttt{points}}\ [=]\ \texttt{te}_{\texttt{signals}})\ [>]\ \texttt{te}_{\texttt{sections}})$$

- where $te_{route\_dispatcher}$, te_{IXL}, te_{points}, $te_{signals}$, and $te_{sections}$ consist of quantified transition rules describing the behaviour of the route dispatcher, the route controller, and the track elements.

Instantiation of Transition Relation

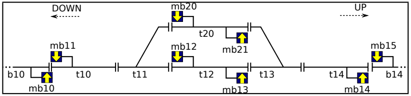
- Instantiation of **transrel** `te` with configuration data yields a transition relation R in predicate form. R is found from `te` in three steps:

  0. Macros are expanded away.
  1. Instantiation step: generic constructs are expanded away
     - Quantified transition rules $[=]$ `id : ElementType • te` are expanded to non-deterministic choices.
     - Applications of domain-specific functions are expanded.
  2. Semantic transformation step: remaining constructs ($\longrightarrow$, $[=]$, and $[>]$) are expanded away.

Example: Quantified Transition Rule for Points

$$([=]\ p : \textbf{Point} \bullet\ [\text{switch\_point}]\ \ p.\text{CMD} \neq p.\text{POS} \longrightarrow p.\text{POS}' = p.\text{CMD})$$

Instantiation with



Point = { t11, t13 }

first expands to

$$(\text{t11.CMD} \neq \text{t11.POS} \longrightarrow \text{t11.POS}' = \text{t11.CMD})\ [=]$$
$$(\text{t13.CMD} \neq \text{t13.POS} \longrightarrow \text{t13.POS}' = \text{t13.CMD})$$

and then gives

$$(\text{t11.CMD} \neq \text{t11.POS} \wedge \text{t11.POS}' = \text{t11.CMD} \wedge \phi_{t11.POS}) \vee$$
$$(\text{t13.CMD} \neq \text{t13.POS} \wedge \text{t13.POS}' = \text{t13.CMD} \wedge \phi_{t13.POS})$$

where $\phi_{id.v} \equiv \wedge_{x \in V \setminus \{id.v\}}(x' = x)$ is a formula expressing that all variable instances except $id.v$ remain unchanged by the transition.

Example: a Transition Rule for the Interlocking

$$([=]\ r : \textbf{Route} \bullet [\,\text{train\_enters\_route}\,]$$
$$r.\text{MODE} = \text{LOCKED} \wedge \textbf{first}(r).\text{CNT} > 0$$
$$\longrightarrow$$
$$r.\text{MODE}' = \text{OCCUPIED} \wedge \textbf{src}(r).\text{CMD}' = \text{CLOSED}$$
$$)$$

Instantiation with

id	src	dst	path	points	signals	conflicts
1a	mb10	mb13	t10;t11;t12	t11:p;t13:m	mb11;mb12;mb20	1b;2a;2b;3;4;5a;5b;6b;7
...	...	...	...	...	...	...
8	mb21	mb14	t13;t14	t13:m	mb13;mb15	1b;2a;4;5a;5b;6a;6b

(for which **Route** = { r1a, ..., r8 }) yields the relation:

$R_{r1a} \vee \ldots \vee R_{r8}$, where e.g.

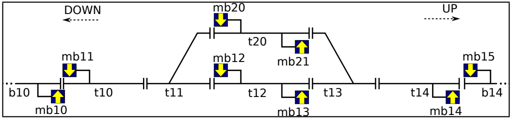
$$R_{r1a} \equiv$$
$$(\texttt{r1a.MODE} = 3 \wedge \texttt{t10.CNT} > 0) \wedge (\texttt{r1a.MODE}' = 4 \wedge \texttt{mb10.CMD}' = 0)$$
$$\wedge\ (\wedge_{\texttt{x} \in \texttt{V} \setminus \{\texttt{r1a.MODE},\texttt{mb10.CMD}\}} (\texttt{x}' = \texttt{x}))$$

Generic Properties – Example

invariant

[no_collision] $(\forall s : \textbf{Section} \bullet s.\mathrm{CNT} < 2)$

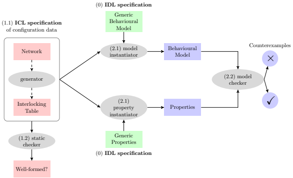
Instantiation with



yields the concrete property:

$$P \equiv \texttt{t10}.\mathrm{CNT} < 2 \land \cdots \land \texttt{t14}.\mathrm{CNT} < 2$$

Conclusions



Contributions:

- Suggestion to use a *domain-specific language* for *generic* interlocking models and their properties.
- *Advantages:* easier to read, write and change generic models and properties.
- Presented such a language.
- The language has been given a *semantics* defining the result of instantiating a generic specification with configuration data.
- The language and generator tools based on the semantics have been *implemented* as part of the **RobustRailS** tool set using Verified's **RT-Tester** tool set as backend.
- These have successfully been *applied* (1) to specify generic models of the novel Danish ERTMS 2 based interlocking systems and (2) to instantiate these for real-world lines and stations. See Vu, Haxthausen & Peleska: Formal modelling and verification of interlocking systems featuring sequential release. *Science of Computer Programming*, 133, Part 2:91 – 115, 2017.

Future work:

- Investigate to which extend the language could be applied to other classes of interlocking systems.
- Make extensions/adaptions of the language.

Questions?