Halfway Generic Verification of Railway Control Systems

Gustav Zickert ‹gustav.zickert@prover.com›

Nikitas Stathatos

RSSRail2023

PROVER

Outline

- Background & problem statement
  - Generic Applications (GA) and Specific Applications (SA)
  - Formal verification of SAs and GAs
  - Software Product Lines (SPL)
- Proposed Method to bridge the gap between SA and GA verification
  - Halfway Generic Verification (HGV)
    - Supports development/debugging of the generic design of a GA
    - Bonus benefit: also supports exploration of static properties of conf data
- Prototype implementation

PROVER

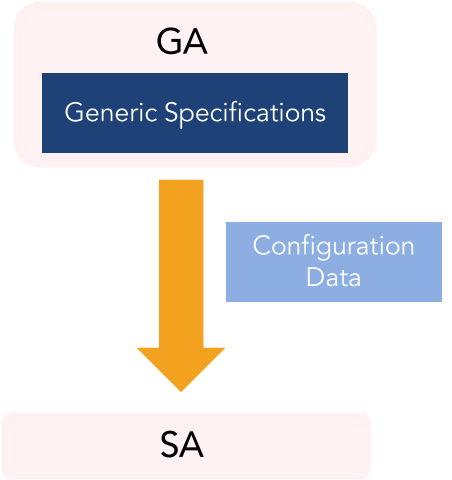
Background

Formal verification of SAs and GAs

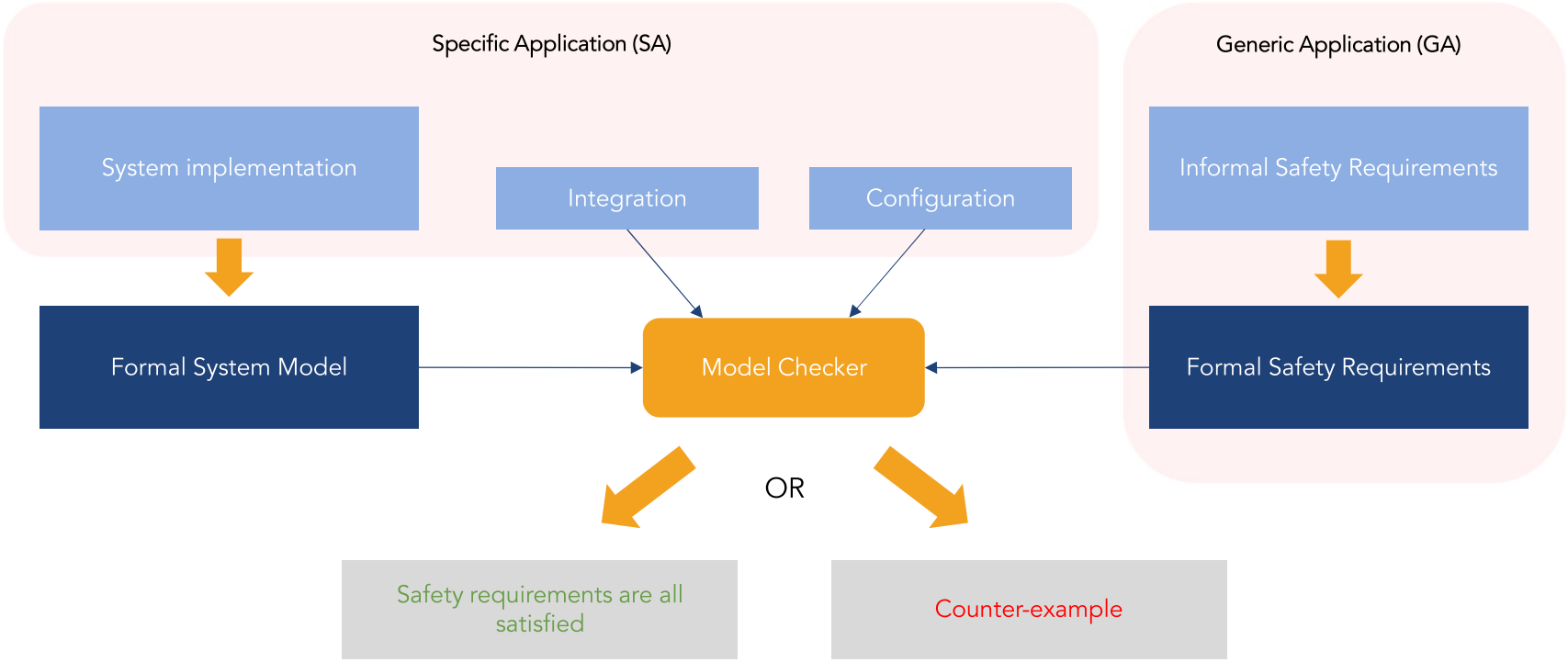
Software Product Lines

Standard development phases: From GA to SA

- GA: Generic specifications for a family of systems common signaling principles
  - Object Model
    - Generic objects (classes); inputs, outputs, internal states, and static relations.
  - Design Specification
    - Generic logic for code generation
  - Test Specification
    - Generic test cases
  - Safety Specification
    - Generic safety requirements for formal verification
- SA: Individual signaling system that is associated with a combination of a GA and some compatible configuration data
- In code generation projects, SA source code is generated from a GDS and configuration data (track layout, route tables, etc.)



Example of a FV approach at SA level, based on Model Checking



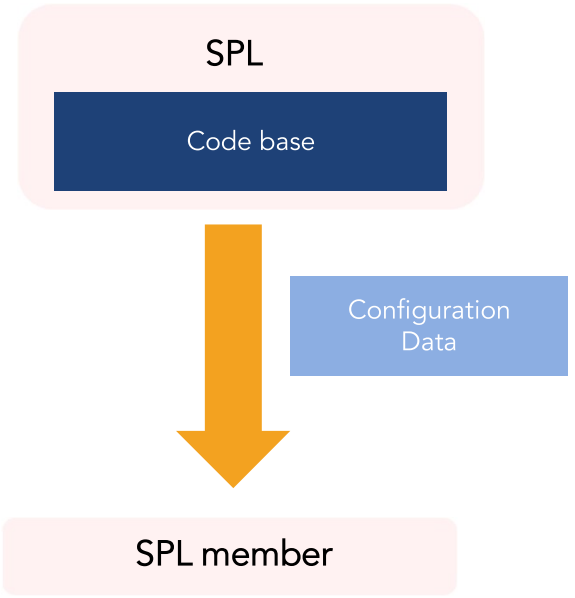
PROVER

Formal Verification of a GA

- It desirable to have an early assessment at GA level
  - Reducing the amount of effort required for the assessment of each SA
  - This requires finding issues as early as possible
    - E.g., the GDS may fail to fulfill the GSS for certain configuration data
- We think that FV is beneficial for assessment of signaling system logic
- FV work in signaling is traditionally done at SA level
- Challenges with applying FV to GAs
  - To directly apply Model Checking at GA level would require a model of all configuration
    - Such a model might not be readily available, or be very large
  - Interactive proof engines typically requires some manual work by experts

Software Product Lines

- A Software Product Line (SPL) is a family of similar software products.
- Variability described in terms of **features**.
  - Each feature is present in some subset of the SPL.
- **Note:** *A family of computerized signaling systems corresponding to a given GA may be considered as a SPL.*



PROVER

FV of Software Product Lines, cont.

- Configuration lifting, a.k.a. variability encoding:
- Encode features using Boolean feature variables
- Semantics: feature variable is True if the feature is present in some possible product.
- Leads to metaproduct/metamodel capable of simultaneously simulating all products of the SPL
- FV of metaproduct against the corresponding metaspecifications
  - In effect one verifies all products of the SPL.

Halfway Generic Verification

Halfway Generic Verification (HGV)

- Formal SPL analysis applied to a family of railway control system SAs, originating from the same GDS
  - Adding feature variables to represent presence of physical and abstract objects in the railyard
- Supports development of GDS and GSS
- Supports the exploration of properties of conf data

PROVER

Overview of the HGV framework

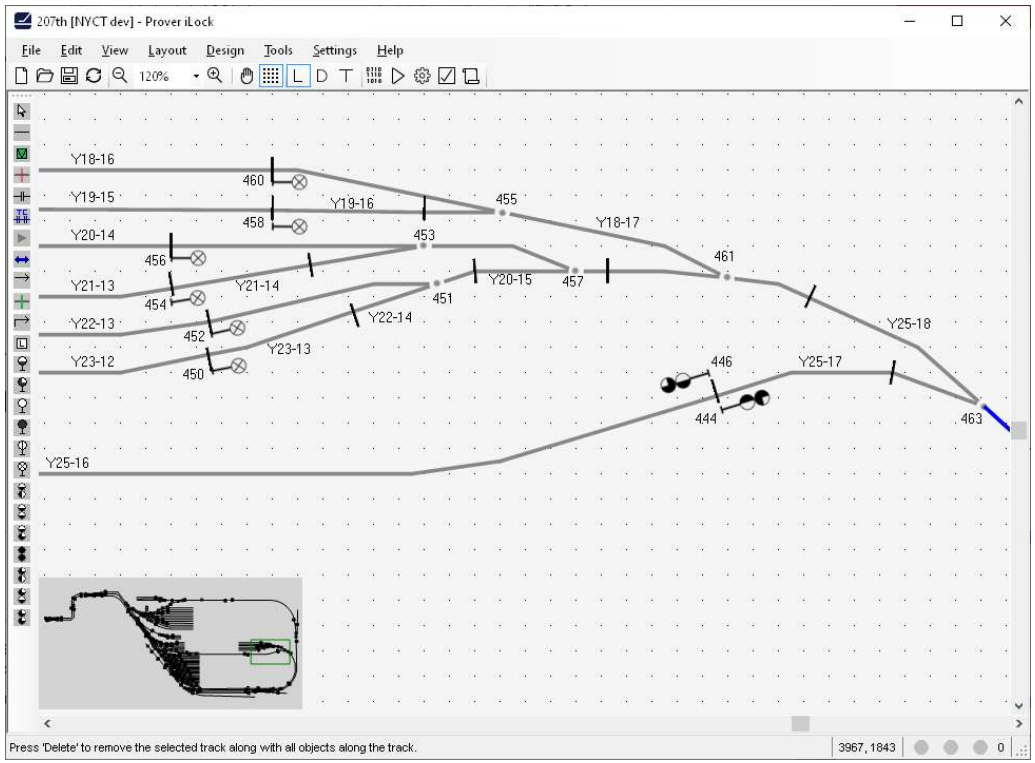
- Input: $GA_i = (OM_i, GDS_i, GSS_i)$ and configuration data C_i.
- Three-step recipe for transforming $(GA_i, C_i) \rightarrow (GA_o, C_o)$:

1. Introduce new "potential" object classes to the OM.
   - Each such potential object class inherits properties from some object class already present in OM_i, but additionally has a special Boolean feature variable Exists.
2. Align the OM, GDS and GSS with the modifications introduced in the first step.
   - Restrict the scope of any quantification that appears in any expression in the GA to existent objects.
3. Replace some (user specified) object instances in C_i with corresponding potential object instances.

Example | Prototype implementation using Prover iLock

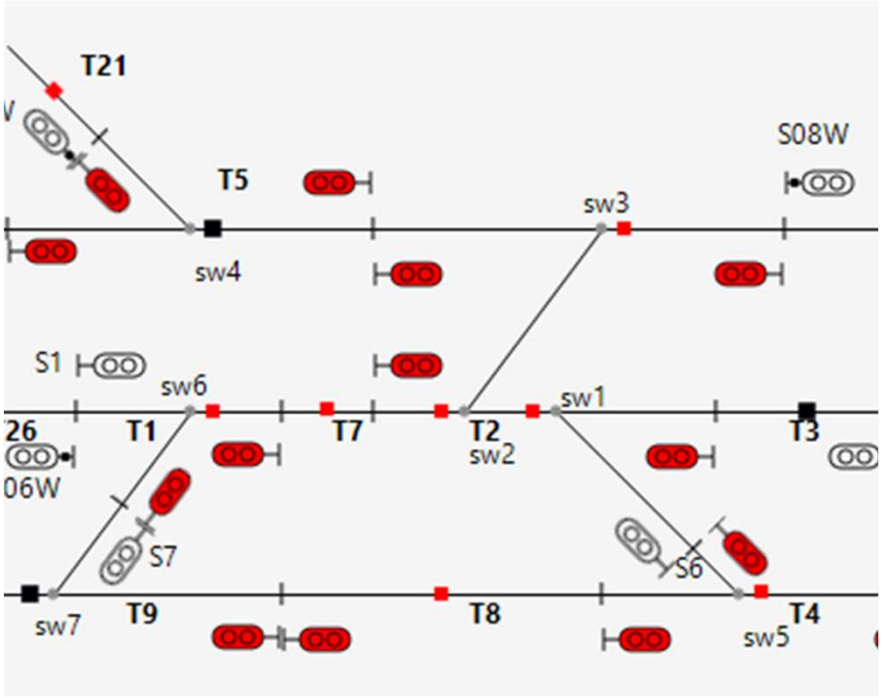
Prover iLock – “cockpit” for Signaling Design Automation

- Main functionality
  - Graphical layout editor
  - Data import/export
  - Simulator
  - Verifier
  - Coder



Toy interlocking model

- Input Object Model OM_i
  - SIGNAL,
  - ROUTE,
  - PROTECTION_AREA,
  - SWITCH,
  - BALISE,
  - ...
- Potential (or variable) object types
  - VAR_SIGNAL,
  - VAR_BALISE,
  - VAR_ROUTE,
  - VAR_ PROTECTION_AREA



Transformation of Generic Design (GDS)

- Original equation in GDS_i

```
CommandedProceed :=
  SOME rt (
    start_signal(rt, SELF) &
    ready_to_proceed(rt)
  );
```

- Transformed equation in GDS_o

```
CommandedProceed :=
  SOME rt (
    rt.exists &
    start_signal(rt, SELF) &
    ready_to_proceed(rt)
  );
```

Transformation of Generic Safety (GSS)

- Original requirement in GSS_i

```
ALL si
ALL rt
ALL rts (
     CommandedProceed(si) &
     start_signal(rt, si) &
     set(rt) &
     route_sections(rt, rts) ->
        ALL tc (
           tracks(rts, tc) ->
              Clear(tc)
        )
);
```

- Transformed requirement in GSS_o

```
ALL si (
  si.exists ->
    ALL rt (
      rt.exists ->
        ALL rts (
           CommandedProceed(si) &
           start_signal(rt, si) &
           set(rt) &
           route_sections(rt, rts) ->
             ALL tc (
                tracks(rts, tc) ->
                  Clear(tc)
             )
        )
    )
);
```

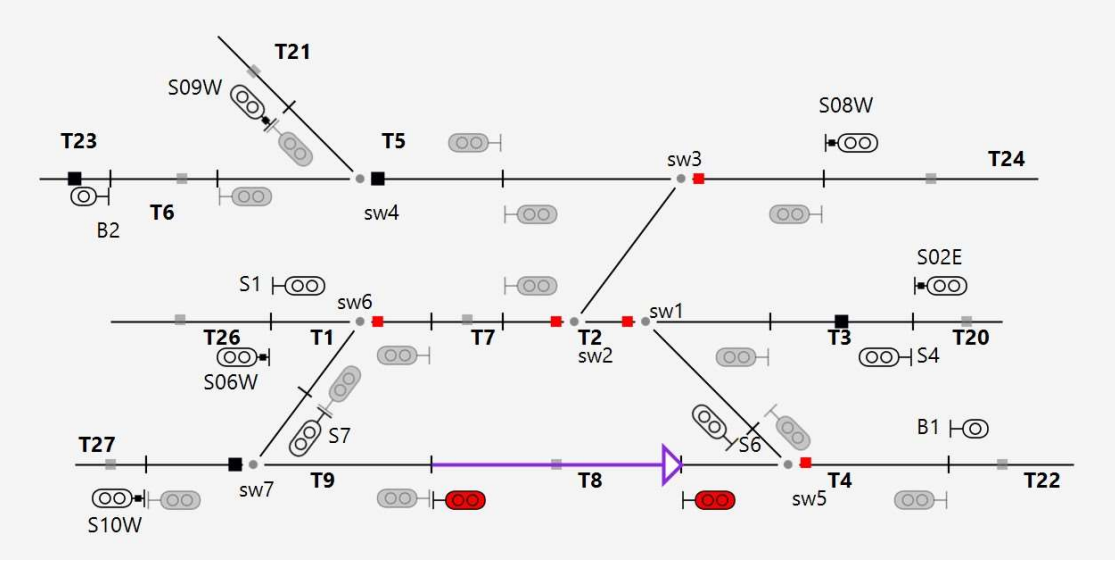
HGV for dynamic requirements

- Constraints on variable objects:
  - All possible combinations of VAR_SIGNAL and VAR_BALISE were analyzed.
  - VAR_ROUTE and VAR_PROTECTION_AREA were **defined** in terms of existence of VAR_SIGNAL instances
    - A route was defined to exist between two existing co-direct signals iff there is no other existing co-direct signal in-between them
    - A protection area was defined to exist if all protected and protective signals exist
- We then performed model checking of the metamodel against the metaspecifications
- In effect, the original generic design GDS_i was verified against the GSS_i for all SAs that may be obtained by including or not including each instance of VAR_SIGNAL and VAR_BALISE.
- The prototype layout had 29 instances of those two object classes.
- Hence, this corresponds to the simultaneous verification of $2^{29} \approx 5 \times 10^8$ systems

Bonus application | HGV for conf data checking

Bonus application: HGV for conf data checking

- Constraints (`C`) and Requirements (`Req`)
  - `C1`: All switches have a balise.
  - `C2`: All routes that do not contain any switch contain a track circuit with a balise.
  - `Req0`: All routes pass over some balise.
- Constraint consistency requirements
  - `Req1`: There is **no** config satisfying `C1` and `C2`.
    - Falsified
- Requirements of type "`C -> Req`"
  - `Req2`: `C1` **implies** `Req0`.
    - Falsified
  - `Req3`: `C1` and `C2` **together imply** `Req0`
    - Valid



Counterexample to Req2

Conclusion and Outlook

- Verification at a generic level is desirable, but could be challenging
- HGV offers a way to formally verify a finite, but large set of systems
- We plan to further develop the method and explore its usefulness in practice to get an GA level assessment