Rule-based Incremental Verification Tools Applied to Railway Designs and Regulations

Bjørnar Luteberget, Christian Johansen,
Claus Feyling, and Martin Steffen

November 10, 2016

UiO : University of Oslo

RailCOMPLETE

Talk overview

Use case

1. Objective and scope: static infrastructure verification
2. Domain background: railway layout and control system
3. Prototype tool: formalization of regulations and Datalog solver integrated with CAD tool

Incremental evaluation

4. Efficiency concerns: incremental evaluation
5. Algorithms: known approaches to incremental Datalog
6. Solvers: current state of the art in incremental solvers

Railway verification and formal methods

- Railway systems: large-scale, safety-critical infrastructure
- High safety requirements: SIL 4 for passenger transport
- Increasingly computerized components
- Typical use of formal methods in railways: model checking of control systems

Objective

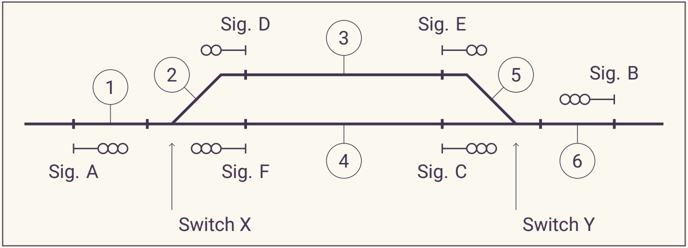
Given a railway signalling and interlocking design,
verify that it complies with regulations.

Secondary objectives:

- Integrate with engineering/design tools
  - On-the-fly verification ("lightweight")
  - Usable for engineers who are not formal methods experts
- Find suitable language for expressing regulations

> "Formal methods will never have a significant impact until they can be used by people that don't understand them."
>
> — (attributed to) Tom Melham

Railway designs for signalling and interlocking



(a) Track and signalling component layout

Route	Start	End	Sw. pos	Detection sections	Conflicts
AC	A	C	X right	1, 2, 4	AE, BF
AE	A	E	X left	1, 2, 3	AC, BD
BF	B	F	Y left	4, 5, 6	AC, BD
BD	B	D	Y right	3, 5, 6	AE, BF

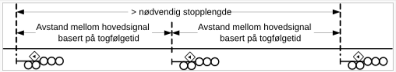
(b) Tabular interlocking specification

Technical regulations

- In our case study: Norwegian regulations from infrastructure manager Jernbaneverket
- Static kind of properties, often related to object properties, topology and geometry (examples later)

https://trv.jbv.no/wiki/Signal/Prosjektering/Lyssignal

e) Dersom nødvendig stopplengde er lengre enn avstanden mellom to etterfølgende hovedsignal, skal det benyttes gjennomsignalering ved hjelp av ATC (Signal/Prosjektering/ATC), se Figur 7.



Figur 7: Forsignalplassering og gjennomsignalering

f) Et forsignal skal plasseres på foregående hovedsignals mast dersom avstanden mellom det tilhørende hovedsignalet og det foregående hovedsignalet er ≤ 2200 meter.

g) Mellom et forsignal og det tilhørende hovedsignalet skal det ikke plasseres andre hoved- eller forsignal.

h) Et forsignal skal plasseres slik at siktavstanden oppfyller kravene til enten “brutt sikt” eller til “ubrutt sikt” i Tabell 4:

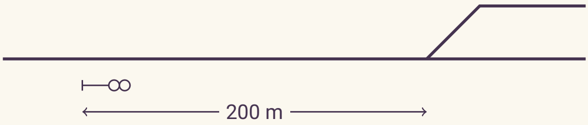
Tabell 4: Siktkrav til forsignal

Sikt	Strekningens høyeste tillatte kjørehastighet [km/h]																	
	40	45	50	55	60	65	70	75	80	85	90	95	100	105	110	115	120	125	≥130
	Siktavstand [m]																		
Ubrutt	78	88	97	107	117	126	136	146	156	165	175	185	194	204	214	224	233	243	250

Technical regulations

Example from regulations:

- A *home main signal* shall be placed at least 200 m in front of the first controlled, facing switch in the entry train path.



- Some categories of regulations useful in static infrastructure design:
  - Object properties
  - Topological layout properties
  - Geometrical layout properties
  - Interlocking properties

Formalization of regulations checking

- Formalize the following information
  - The CAD design (extensional information, or facts)
  - The regulations (intensional information, or rules)
- Use a solver which:
  - Is capable of verifying the rules
  - Runs fast enough for on-the-fly verification

Datalog

- Basic Datalog: conjunctive queries with fixed-point operators ("SQL with recursion")
  - Guaranteed termination
  - Polynomial running time (in the number of facts)
- Expressed as logic programs in a Prolog-like syntax:

$$a(X, Y) \text{ :- } b(X, Z), c(Z, Y)$$

$$\Updownarrow$$

$$\forall x, y : ((\exists z : (b(x, z) \land c(z, y))) \rightarrow a(x, y))$$

- We also use:
  - Stratified negation (negation-as-failure semantics)
  - Arithmetic (which is "unsafe")

Encoding facts and rules in Datalog

- The process of formalizing the railway data and rules to Datalog format is divided into three stages:
  1. Railway designs (station data) – facts
  2. Derived concepts (used in several rules) – rules
  3. Technical regulations to be verified – rules
- Now, more details about each stage...

Input documents representation

- Translate the railML XML format into Datalog facts using the ID attribute as key:

$$
\begin{aligned}
\textit{track}(a) &\leftarrow \text{element}_a \text{ is of type } \texttt{track}, \\
\textit{signal}(a) &\leftarrow \text{element}_a \text{ is of type } \texttt{signal}, \\
&\vdots \\
\textit{pos}(a,p) &\leftarrow (\text{element}_a.\texttt{pos} = p), \quad a \in \text{Atoms}, p \in \mathbb{R}, \\
&\vdots \\
\textit{signalType}(a,t) &\leftarrow (\text{element}_a.\texttt{type} = t), \\
& t \in \{\text{main, distant, shunting, combined}\}\,.
\end{aligned}
$$

Input documents representation

- To encode the hierarchical structure of the railML document, a separate predicate encoding the parent/child relationship is added:

> *belongsTo*$(a, b) \leftarrow b$ is the closest XML ancestor of a
> whose element type inherits from
> `tElementWithIDAndName`.

Derived concepts

- Derived concepts are defined through intermediate rules
- Railway concepts defined independently of the design
- Example:

$$\textit{directlyConnected}(a,b) \leftarrow \exists t : \textit{track}(t) \wedge \textit{belongsTo}(a,t) \wedge \textit{belongsTo}(b,t),$$

$$\textit{connected}(a,b) \leftarrow \textit{directlyConnected}(a,b) \vee (\exists c_1, c_2 : \textit{connection}(c_1,c_2) \wedge \textit{directlyConnected}(a,c_1) \wedge \textit{connected}(c_2,b)).$$

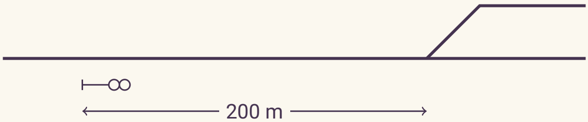
- A library of concepts allows concise expression of technical regulations

Technical regulations as Datalog rules

- Detecting errors in the design corresponds to finding objects involved in a regulation violation
- To *validate* the rules in a given design, we show that there are no satisfiable instances of the *negation* of the rule
- Some examples:
  - Example 1, home signal placement: topological and geometrical layout property for placement of a home signal
  - Example 2, train detector conditions: relates interlocking to topology
- These are Jernbaneverket regulations which are relevant for automatic verification

Rule: example 1

- A *home main signal* shall be placed at least 200 m in front of the first controlled, facing switch in the entry train path.
- Uses arithmetic and negation

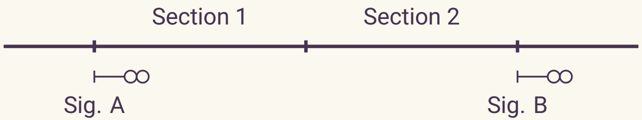


$$
\begin{aligned}
\textit{isFirstFacingSwitch}(b,s) \leftarrow\ & \textit{stationBoundary}(b) \land \textit{facingSwitch}(s) \land \\
& \neg(\exists x : \textit{facingSwitch}(x) \land \textit{between}(b,x,s)),
\end{aligned}
$$

$$
\begin{aligned}
\textit{example1Violation}(b,s) \leftarrow\ & \textit{isFirstFacingSwitch}(b,s) \land \\
& (\neg(\exists x : \textit{signalFunction}(x, \text{home}) \land \textit{between}(b,x,s)) \lor \\
& (\exists x, d, l : \textit{signalFunction}(x, \text{home}) \land \\
& \land\ \textit{distance}(x,s,d,l) \land l < 200).
\end{aligned}
$$

Rule: example 2

- Each pair of adjacent train detectors defines a track detection section. For any track detection sections overlapping the route path, there shall exist a corresponding condition on the activation of the route.



Tabular interlocking:

Route	Start	End	Sections must be clear
AB	A	B	1, 2

Rule: example 2

$$\begin{aligned}
\textit{adjacentDetectors}(a, b) \leftarrow & \textit{trainDetector}(a) \wedge \textit{trainDetector}(b) \wedge \\
& \neg \textit{existsPathWithDetector}(a, b),
\end{aligned}$$

$$\begin{aligned}
\textit{detectionSectionOverlapsRoute}(r, d_a, d_b) \leftarrow & \textit{trainRoute}(r) \wedge \\
& \textit{start}(r, s_a) \wedge \textit{end}(r, s_b) \wedge \\
& \textit{adjacentDetectors}(d_a, d_b) \wedge \textit{overlap}(s_a, s_b, d_a, d_b),
\end{aligned}$$

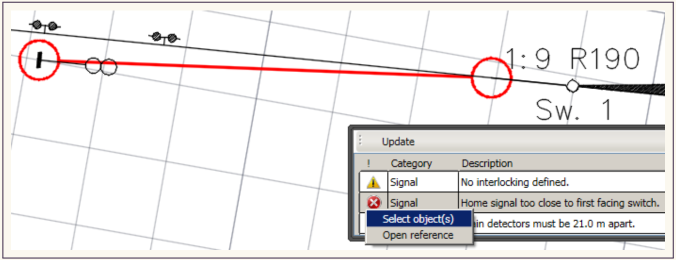
$$\begin{aligned}
\textit{detectionSectionCondition}(r, d_a, d_b) \leftarrow & \textit{detectionSectionCondition}(c) \wedge \\
& \textit{belongsTo}(c, r) \wedge \textit{belongsTo}(d_a, c) \wedge \textit{belongsTo}(d_b, c).
\end{aligned}$$

$$\begin{aligned}
\textit{ruleViolation}(r, d_a, d_b) \leftarrow & \\
& \textit{detectionSectionOverlapsRoute}(r, d_a, d_b) \wedge \\
& \neg \textit{detectionSectionCondition}(r, d_a, d_b).
\end{aligned}$$

Prototype tool implementation

- Prototype using XSB Prolog tabled predicates, front-end is the RailCOMPLETE tool based on Autodesk AutoCAD
- Rule base in Prolog syntax with structured comments giving information about rules

```
%| rule: Home signal too close to first facing switch.
%| type: technical
%| severity: error
homeSignalBeforeFacingSwitchError(S,SW) :-
    firstFacingSwitch(B,SW,DIR),
    homeSignalBetween(S,B,SW),
    distance(S,SW,DIR,L), L < 200.
```



Running time

	Testing station	Arna phase A	Arna phase B
Relevant components	15	152	231
Interlocking routes	2	23	42
Datalog facts	85	8283	9159
Running time (s)	0.1	4.4	9.4

- Running time for verification of a few properties: $\approx$1 – 10 s
  - Acceptable, for now
  - More optimization needed for truly on-the-fly verification

- Increase margins for
  - Many times larger models (stations)
  - 10x — 100x more rules

Efficiency considerations

- Incremental updates
  - Changes in the CAD design causes the whole verification to start over
  - More efficient: recompute only the parts that are affected by the changes

Approaches to incremental Datalog

- Propagate added or deleted sets of base propositions ΔP through constant set of rules (view maintenance)

Typical incremental Datalog approaches:

- Add extra "book-keeping" to the algorithm, to remember how derived facts were derived.
  - Gets complicated with recursive rules
- Without extra book-keeping:
  - Adding items (positively) is straight-forward
  - Deleting items (positively) requires search for alternative support
  - Conversely for negated terms (assuming stratified negation)

The *delete and rederive* algorithm (DRed)

- Described by Gupta et al., 1993.
- Forward-chaining approach:
  - Example:

    ```
    a(X) :- b(X).    (1)
    a(X) :- c(X).    (2)
    b(1). c(1).
    ```

  - Adding a base fact $\Delta^+ = \{b(2)\}$ makes rule (1) fire, producing `a(2)`.
  - Removing a base fact $\Delta^- = \{b(1)\}$ from $\{b(1). c(1). a(1).\}$ propagates through rule (1), producing a minimum set $\{c(1).\}$. This set is used for forward chaining through rules again, producing $\{c(1). a(1).\}$
- Expressible in Datalog itself (Staudt and Jarke, 1996)
- Negation in body flips addition/removal, OK with stratification.

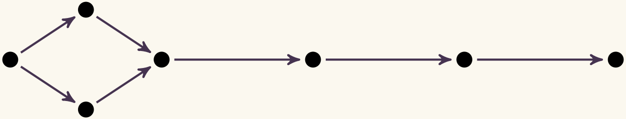
Problem: transitive rules

Highly interconnected facts and rules (few strata), such as transitive rules, can be inefficient with DRed.

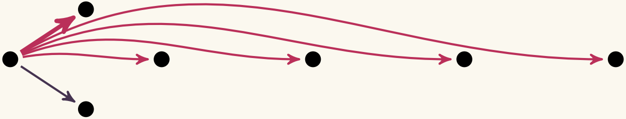
- Example: Graph reachability

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).
```

- Edge relation:

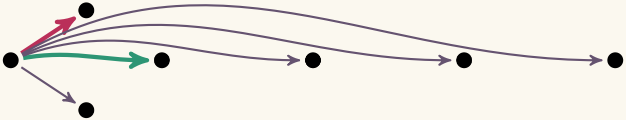


- Paths from leftmost vertex:



The *Forward/Backward/Forward* algorithm (FBF)

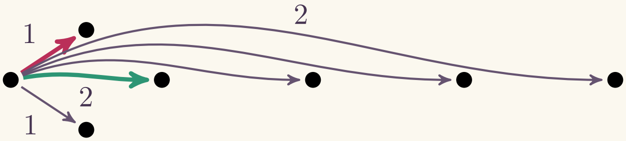
- Newer algorithm by Motik et al., 2015
- Combination of forward and backward chaining
  - When adding a potential deletion to the overapproximation, search for alternative support for the conclusion.



- More efficient than DRed on most tests, especially for highly interconnected strata

Counting and other “bookkeeping” approaches

- Add more information to the result set, for example how many derivations a fact has.



- More complicated in the presence of recursion and other features $\rightarrow$ save the support of derived facts.
- Example from Saha and Ramakrishnan, 2003.

```
edge(0,1)
edge(0,2)
edge(1,1)
edge(1,2)
```

(a)

Answer	Supports
reach(0,1)	⟨1,{edge(0,1)}⟩, ⟨2,{reach(0,1),edge(1,1)}⟩
reach(0,2)	⟨1,{edge(0,2)}⟩, ⟨2,{reach(0,1),edge(1,2)}⟩

(b)

Efficiency gains

Using XSB's incremental facilities in our prototype tool

			Testing station	Arna phase A	Arna phase B
Relevant components			15	152	231
Interlocking routes			2	23	42
Datalog input facts			85	8283	9159
XSB:					
Non-incremental verif.:	Running time:	(s)	0.015	2.31	4.59
	Memory	(MB)	20	104	190
Incremental verif. baseline:	Running time	(s)	0.016	5.87	12.25
	Memory	(MB)	21	1110	2195
Incr. single object update:	Running time	(s)	0.014	0.54	**0.61**
	Memory	(MB)	22	1165	2267

Case study size and running times on a standard laptop.

Tools for incremental Datalog

- XSB Prolog
  - It works! However, memory usage increases 11x.
- RDFox
  - FBF algorithm, lower memory usage.
  - Does not support higher-arity relations (only 1 or 2 parameters, corresponding to RDF triples).
- LogicBlox
  - Commercially supported implementation.
  - Not evaluated by us, yet.
- Dyna
  - Statistical AI research language.
  - Implementation not mature enough for our use.

Status

- Tool support for incremental evaluation comes close, but is not fully capable of supporting our use case.
- Collaboration with Boris Motik's group in the University of Oxford on further development on RDFox for supporting our use case.