Use of Domain Theories in Applied Formal Methods

Lars-Henrik Eriksson
lhe@it.uu.se

Department of Information Technology
Uppsala University
Box 337
SE-751 05 UPPSALA, Sweden

Abstract. A formal theory of an application domain can serve a key role in formal specification and verification of systems operating in that domain. This is particularly relevant when working with families of similar systems intended to control some kind of industrial process or function where the control principles are general while the specifics of the system depend on the particular installation the system is intended to control. In such situations a domain theory can facilitate writing generic specifications or implementations for the entire family of systems which can then be configured for each particular installation. Use of concepts defined in the domain theory can also facilitate communication with domain experts without knowledge in formal methods.
An example of such a family is railway signalling systems which all implement the same general signalling principles of ensuring safe train operation, while the exact function of a particular signalling system depends on the railway track layout under its control.
We will give concrete examples from industrial practise in the railway domain, showing how domain theories can help in the formal verification process. The examples include writing and validating formal specifications, formally verifying implementations and analysing and communicating the results of failed verifications.

1 Introduction

1.1 Domain theories and configurable systems

We consider the role of formal theories of the application domain in formal specification and verification of configurable systems.

By "configurable systems" we mean families of systems where each individual system has the same abstract behaviour, while the concrete behaviour differs according to details specific to the particular system. An example is provided by railway signalling systems ("interlockings"), which all have identical functions on an abstract level as they implement the same general signalling principles. On the concrete level, differences in function between different interlockings are determined by the particular physical layout as well as other properties – both

abstract and concrete – of the railway track system controlled by the interlocking. Another examples is provided by telecommunication systems, which carry out the general function of providing telephone and other communication services, while the specific behaviour depends on the particular structure and components used in the particular system or subsystem.

If there is a uniform way of configuring the individual systems of a family of systems, then the fact that the abstract behaviour is identical over the family means that there will be a regularity in how the expected behaviour of the concrete systems differ. This can be exploited to simplify the formal specifications by writing a *generic* specification for the family. The generic specification is then parameterised by applying configuration data, so that the specification of a specific system is obtained. In the same way, a generic implementation could be written and then configured to implement the function of a specific system.

By a "domain theory" we mean an axiomatic description of the structure and concepts of the application domain – in particular those concepts which are important for configuring the specification or implementation of a system as outlined above. In the case of the railway signalling systems, concepts which are axiomatised include concrete and abstract objects such as signals, points, routes etc. and their relations such as the position of signals relative to the track system. Although the examples in this paper will be exclusively from the railway signalling domain, the ideas and techniques presented are general and applicable to other domains.

The use of a domain theory can be motivated also from a non-technical perspective. If the axiomatised concepts are similar to those used by domain experts, a domain theory will facilitate the communication between the formal methods practitioner and the domain expert, both regarding the formal specification and the results of formal verification (see section 3.7). The axiomatised concepts can also be integrated with a specification language, to obtain a domain-specific language tailored to the application domain. See [7] for a discussion of an "Interlocking Specification Language" for the railway signalling domain.

The use of domain theories is in fact applicable to all aspects of systems engineering. See e.g. [2] for an discussion of how and why a domain theory for railways – such as the one we use in this paper – should be extended to cover all aspects of railway operation and management.

1.2 Contributions

The contribution of this paper is to show the practical application and significance of domain theories in formal specification and verification by giving examples from the industrial work of Industrilogik L4i AB[1].

Since the mid-90s, Industrilogik has developed and used a toolset and methodology for formal specification and verification of configurable systems in the sense given above [4] [5]. The main application area has been railway signalling systems. The examples in this paper have been adapted from several industrial

[1] In 2005 Industrilogik was aquired by Prover Technology, www.prover.com

projects with computer- and relay-based interlocking systems in Sweden and Norway. Overview of some of these projects are included in [3] [4] [6].

Sample runs using the main tool of the toolset, GTO, will be shown[2]. GTO supports validation using theorem proving and simulation as well as verification by refinement proof. The notation of the tool is a simple form of temporal predicate logic with finite domains (no function symbols and only a previous-moment temporal operator). As all domains are finite, a theorem proving problem can be reduced to a satisfiability problem in propositional logic, which is solved by an independent SAT solver which is interfaced to GTO. This approach to modelling and proving is similar to that used by the Alloy [8] modelling tool.

Peleska et.al. (e.g. [10]) have also worked on formal specification and verification of configurable systems in the railways domain. They use similar data for the configuration of the systems, but do not use a proper theory of the domain to axiomatise the configuration data.

2 The Railway Domain

We will begin by introducing our example domain theory, what the configuration data of the domain is, a sample configuration, and fragments of the axiomatisation.

2.1 Geographical data

In the case of railway signalling systems, a formal description of the configuration data is called the *geographical data* of the particular interlocking. (This sense of geographical data is similar, but not identical, to the one used in work on formal verification of geographical data of the british SSI interlockings [9] [11].)

Using geographical data, generic requirements specifications that describe general signalling principles can be specialised to give a requirements specification for a particular interlocking installation (see section 3.4). Similarly, interlockings can be implemented using generic modules (either in software or hardware) which are configured using geographical data to give a specialised implementation for a particular site. An example of interlockings working using this principle are Bombardier Transportation EBILOCK family of interlockings (see section 3.3).

Given that the precise requirements of a generic specification – as well as the precise behaviour of a generic interlocking – are critically dependent on the geographical data, the correctness of the geographical data is important. Some kinds of geographical data – let us call them "primary" geographical data – are direct descriptions of the physical track structure and its concrete properties. Clearly, this data can not be formally verified, but its internal consistency – e.g. that it describes a physically possible track system – can be checked using a domain theory for rail systems (see section3.1).

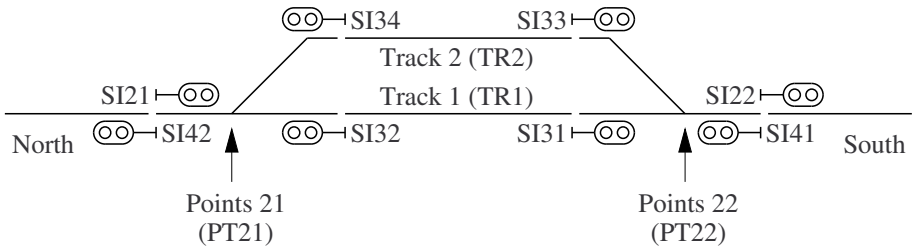
[2] The actual computer output has been edited slightly for brevity and clarity.

Other kinds of geographical data – let us call them "secondary" geographical data – are data that are wholly or in part determined by the primary geographical data. One example is the description of all possible *routes* through the track system – a route typically being defined as a path through (part of) the track system on which a train could run, beginning and ending at a signal. Other examples are the various kinds of protection areas required around a route to prevent possible collision with trains or vehicles close to the route. The construction and verification of secondary geographical data is of critical importance to the safety of the interlocking, while being one of the most time-consuming and error prone tasks in the interlocking design process.

Given a sufficiently complete domain theory and generic requirements specification, secondary geographical data can be formally verified or automatically generated given a set of primary geographical data (see section 3.2).

2.2 A simple domain theory

As the running example, we will use a simple railway system made up of signals and railway track[3].



The geographical data for this system is given by a number of sets of objects and relations on those objects. The sets are

UNITS – the track system is made up of as a set of "units", a unit being a set of points, a linear piece of track etc. In this example *NORTH*, *PT21*, *TR1*, *TR2*, *PT22* and *SOUTH* represent units.

POINTS – the points, a subset of *UNITS*. In this example *PT21* and *PT22* represent points.

BORDERS – the track units bordering parts of the railway outside the part under consideration, a subset of *UNITS*. In this example *NORTH* and *SOUTH* represent borders.

[3] The numbering scheme of this example is the conventional one used in Swedish railway signalling.

SIGNALS – the signals. In this example *SI21*, *SI22*, *SI31*, *SI32*, *SI33*, *SI34*, *SI41*, *SI42* represent signals.

ROUTES – the train routes. We will regard the routes as abstract objects having certain properties, such as the set of the track units making up each route. The concrete wiew of the routes actually being such sets (together with other relevant data) would also be possible. In this example *TR2131*, *TR2133*, *TR3141*, *TR3341*, *TR2232*, *TR2234*, *TR3242*, *TR3442* represent routes. E.g. *TR2131* represents the route from signal 21 to signal 31.

The relations are given by the predicates:

connectsTo – relates two adjacent pieces of track.

leftBranch – relates facing points with the track unit reached by going to the left through the points.

rightBranch – relates facing points with the track unit reached by going to the right through the points.

ahead – relates a signal to the track unit ahead of the signal.

inRear – relates a signal to the track unit in rear of the signal.

partOf – relates a route to the track units making up the route.

before – relates a route to the track unit from which a train enters the route.

entry – relates a signal to a route starting at that signal.

conflict – relates routes which are "in conflict", i.e. they may not be used simultaneously by two different trains.

connectsTo, *leftBranch* and *rightBranch* determine the topology of the track system. *ahead* and *inRear* together determine the location and direction of the signals. *partOf* and *before* together determine the extent and direction of each route.

Some examples of predicate instances which hold are:
connectsTo(PT21,NORTH), *connectsTo(PT21,TR1)*, *connectsTo(PT21,TR2)*, *connectsTo(NORTH,PT21)*, *connectsTo(TR1,PT21)*, *connectsTo(TR2,PT21)*, *leftBranch(PT21,TR2)*, *rightBranch(PT21,TR1)*, *ahead(SI21,NORTH)*, *inRear(SI21,PT21)*, *entry(SI21,RT2131)*, *entry(SI21,RT2133)*, *conflict(RT2131,RT2232)*, *conflict(RT2133,RT2234)*.

The domain theory express the constraints on the geographical data which are necessary for the data to represent a possible real system[4]. Fragments of a domain theory for the track system is given by the following predicate logic formulae:

1. $\forall u1, u2 \in \mathit{UNITS}\ (\mathit{connectsTo}(u1, u2) \rightarrow \mathit{connectsTo}(u2, u1)$
2. $\forall p \in \mathit{POINTS}\ \forall u \in \mathit{UNITS}\ (\mathit{rightBranch}(u, p) \rightarrow \mathit{connectsTo}(u, p))$
3. $\forall u1 \in \mathit{UNITS}\ \exists u2 \in \mathit{UNITS}\ \mathit{connectsTo}(u1, u2)$
4. $\forall p \in \mathit{POINTS}\ \exists u1, u2, u3 \in \mathit{UNITS}\ (\mathit{connectsTo}(p, u1) \wedge \mathit{connectsTo}(p, u2) \wedge \mathit{connectsTo}(p, u3) \wedge u1 \neq u2 \wedge u1 \neq u3 \wedge u2 \neq u3 \wedge \forall u4 \in \mathit{UNITS}\ (\mathit{connectsTo}(p, u4) \rightarrow u1 = u4 \vee u2 = u4 \vee u3 = u4))$

[4] This domain theory excludes some real systems that are possible but unrealistic.

These axioms express the internal coherence of the data, such that the *connectsTo* relation is symmetric (1), that going right through facing points, you must reach a unit connected to the points (2), that every unit must be connected to some other unit (3), and that a set of points must be connected to exactly three different units (4).

5. $\forall r \in ROUTES\ \forall u1, u2 \in UNITS$
   $((partOf(u1, r) \vee before(u1, r)) \wedge (partOf(u2, r) \vee before(u2, r)) \rightarrow$
   $\neg\exists p \in POINTS\ (partOf(p, r) \wedge leftBranch(p, u1) \wedge rightBranch(p, u2)))$

Axiom (5) expresses a physical constraint. As trains cannot go "sideways" through points, no route can do this either.

6. $\forall r \in ROUTES\ \exists s \in SIGNALS\ entry(s, r)$
7. $\forall r1, r2 \in ROUTES\ (r1 \neq r2 \wedge \exists u \in UNITS\ (partOf(u, r1) \wedge partOf(u, r2) \rightarrow$
   $conflict(r1, r2))$

Axioms (6) and (7) are examples of axioms which express constraints on the design of the system. In the case of (6), the requirement that there must be an "entry" signal at the beginning of every train route. In the case of (7), the requirement that if two routes overlap, then they must be in conflict.

In this case the track units, signals and their relations are primary geographical data, while the routes and their properties are secondary geographical data. In principle the possible routes are determined by the configuration of track units and signals.

2.3 Interaction with the environment

In any formal specification of a control system, it is important to describe how the system interacts with its environment. However, for the subject of this paper it is not an important point. We will simply assume that there are predicates which represent (parts of) the currect state of the environment without going into any details of how to express changes to that state.

3 Applications of the domain theory

Having introduced the domain theory for railway signalling, we will now discuss how it can be put to use in a number of situations related to specification, configuration and verification of railway signalling systems.

3.1 Verification of the system configuration

The geographical data for a specific system (to be used either with a generic specification or implementation) can be verified against the domain theory by showing that every axiom of the domain theory is true given the sets and predicates representing the data. This can be shown by theorem proving, but it is also

possible to do a straight calculation of the truth values of the axioms assuming that all quantified variables in the domain axioms range over done over the sets representing particular (finite) sets of objects of the geographical data.

Suppose that $connectsTo(PT21, TR1)$ is included in the geographical data, but $connectsTo(TR1, PT21)$ is not. In this sample run of GTO, a file with the domain theory and data is loaded. The tool detects the inconsistency by computing the truth value of axiom (1) by iteration over all units and checking the implication for each one. (`domain_1` is the identifier of axiom (1) above.) The user asks for a explanation for the falsity of the axiom using the GTO `why` command which attempts to give a motivation for the truth value of a formula. The tool notes that the data violates the axiom because the predicate instance $connectsTo(TR1, PT21)$ is missing.

```
> load sample1
Violated invariants: domain_1
> why domain_1
Formula is FALSE because ~connectsTo(TR1,PT21)
```

3.2 Generation of secondary geographical data

For a large system with complicated structure, the generation of secondary geographical data can be a complicated task, even if the correctness of that data can be verified automatically. However, the domain theory can be used to automatically generate secondary data.

In some cases, this can be done simply by turning axioms into definitions. If the truth values of the predicates occurring in the definiens is known, then the truth values of the defined predicates can be generated by a straightforward calculation. E.g. axiom (7) can be turned into the definition

8. $conflict(r1, r2) \equiv r1 \neq r2 \wedge \exists u \in UNITS\ (partOf(u, r1) \wedge partOf(u, r2))$

which will define *conflict* to be the smallest relation consistent with the axiom.

GTO automatically computes the truth values of defined predicates when the truth values of the definiens is completely known. The following sample interaction shows the definition and lists the result.

```
> listdef conflict
conflict(r1,r2) == r1<>r2&SOME u:UNITS (partOf(u,r1)&partOf(u,r2))
> list conflict
conflict(RT2131,RT2133)
conflict(RT2131,RT2232)
conflict(RT2131,RT3242)
conflict(RT2131,RT3442)
conflict(RT3141,RT3341)
...
```

In more complicated situations, the secondary data can be generated by finding a satisfying truth assignment to the corresponding predicates, given a known truth assignment to the primary data predicates. The primary data gives a "partial interpretation" of the domain axioms where secondary data predicates are undetermined. Here "partial interpretation" can be understood both in the logical sense of an interpretation of predicates or in the sense of simplifying the axioms using known values of the primary geographical data. The problem of finding values for the secondary predicates is a satisfiability problem. Since the sets are finite, the problem can be solved automatically using a propositional satisfiability (SAT) solver. The SAT solver would generate truth assignments to the secondary data predicates, effectively creating correct secondary geographical data.

A problem with using a SAT solver is that in some cases the numbers of elements of sets belonging to the secondary data (e.g. the set *ROUTES*) are not known in advance, as they must be known in order to create a SAT problem. One solution is making a conservative estimate of the maximum number of elements (the Alloy [8] approach). Another one which is possible in some cases (particularly regarding the set *ROUTES*) is to include only one element, but generate the complete set by finding successive solutions to the SAT problem. The latter approach is implemented in the SST/SVT formal methods toolset used by Bombardier Transportation for interlocking software development.

In this sample run of GTO, the relations defining routes have been left undetermined. The *ROUTES* set includes a single identifier *route*. By using a SAT solver, an assignment is found that defines a route. The (true instances of) relations *before*, *partOf* and *entry* are listed.

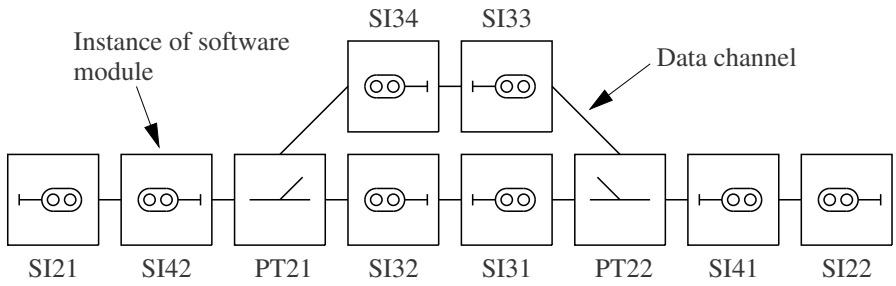
```
> load sample2
> satisfy
> list before partOf entry
before(route,NORTH)
partOf(route,PT21)
partOf(route,TR1)
entry(SI21,route)
```

3.3 Configuring an implementation

One use of configuration data is to instantiate general software modules to make up a complete program to work with the system described by the data. An example is the computer-based EBILOCK 950 interlocking (signalling) system manufactured by Bombardier Transportation.

The EBILOCK software is made up of modules designed to handle all functions related to specific kinds of physical objects in a railway track system such as signals and points. To configure the software for a specific installation, one instances is made of each module for every physical object of the corresponding type. The modules have communication ports which are connected according to the geographical layout of the track system. For the sample rail system above,

one signal module and one points module are instantiated 2 and 8 times, respectively, and connected to give a complete software system with the following structure:



Clearly, the correctness of the instantiated software depends on the correctness of the geographical data. Bombardier Transportation uses a custom formal specification tool (SST) to check geographical data according to a domain theory. The tool is an offspring of the GTO tool and uses a similar predicate logic-based notation. It can directly read EBILOCK configuration data files and use the contents to define the truth values of specific predicates, such as predicates which relate an object instance to its neighbours. In fact, the SST tool implements a domain-specific specification language tailored to the domain of EBILOCK implementations.

3.4 Concept definitions for requirements

To facilitate writing and validating the requirements specification for a system, the specification should preferably be written on a level of abstraction where the concepts used are taken from the application domain, rather than from the implementation domain. In the railway example, one typical requirement on signals is that two different signals leading in to the same track section must not display a proceed aspect[5] at the same time, as that could cause a train collision. Taken as requirement only on one of the signals, *SI21*, this could be formalised as

9. $proceed(SI21) \rightarrow \neg proceed(SI32) \wedge \neg(proceed(SI34) \wedge \neg(proceed(SI22) \wedge (left(PT21) \wedge left(PT22) \vee right(PT21) \wedge right(PT22)))$

Here *proceed* is a predicate that represents an abstract state of a signal controlled by the signalling system. If *proceed* is true for the signal, it displays a proceed aspect, otherwise it does not.

[5] ...i.e. a “green light”.

Even in this case of a single signal in a simple example track system, such a formal requirement is slightly tricky to formulate. In more complicated layouts it becomes difficult and error-prone. In a generic specification, it would not even be possible to write such a "low-level" requirement as it is not known until the system is configured what the actual signals or their relationships are.

Signalling engineers solve this problem using the abstract concept of a "route" explained above. Each route can be reserved ("locked") for the movement of a train. A requirement on the signal will be that it can display proceed only if it is the entry signal of a locked route. An additional requirement is that the signal can display proceed only if that route is not in conflict with another route which also is locked.

In this case, a domain theory of the application domain provides a natural means of expressing the requirements using formalised concepts such as "route" and "conflict". This simplifies both writing and validation of the specification.

We require that if a signal displays a proceed aspect, then a route beginning at the signal must be "ready" (10). We define "ready" to mean – among other things – that the route is locked and conflict-free (11). The predicate *routeLocked* is used to represent the locking status of the routes. Finally we define that a route is conflict-free when it is not in conflict with any other locked route (12).

10. $\forall s \in SIGNALS\ (proceed(s) \rightarrow \exists r \in ROUTES\ (entry(s, r) \wedge ready(r))$
11. $ready(r) \equiv routeLocked(r) \wedge conflictFree(r) \wedge \ldots)$
12. $conflictFree(r) \equiv \neg\exists r1 \in ROUTES\ (r \neq r1 \wedge routeLocked(r1) \wedge conflict(r, r1)$

The requirement (10) with associated definitions is both more abstract and not dependent on any particular route. Of course, using specific geographical data to provide concrete values for the sets and *conflict* relation, we will obtain the requirement (9) as a special case.

3.5 Validation of requirements specifications

By using concepts from a domain theory, validation of the requirements specification is facilitated as the specification can be expressed in general and abstract terms. In many cases, directly "translating" the formal requirements into natural language will result in a text which can be given to domain experts for approval. However, the domain theory also makes it possible to validate the specification by stating and proving correctness properties.

In using the abstract route concept, we improved the structure of the specification, but lost the intuitiveness of the property that "two different signals leading in to the same track section must not display a proceed aspect at the same time". It is not immediately obvious from the formulation of the formal requirement (10) and associated concept definitions that this intuitive safety condition is ensured.

Again using the domain theory, this safety condition can be formalised and proved to follow from the formal specification – in particular formulae (10)-(12) above. The safety condition can be formalised as

13. $\forall s1, s2 \in SIGNALS\ (proceed(s1) \wedge proceed(s2) \rightarrow \neg\exists u \in UNITS\ \exists r1, r2 \in ROUTES\ (entry(s1, r1) \wedge entry(s2, r2) \wedge partOf(u, r1) \wedge partOf(u, r2)))$

This formula still uses the axiomatisation of routes, as it provides a convenient way of describing the parts of the track system in rear of a signal, but the concepts of locking and conflict are not used.

This proof can not be carried out using a SAT solver as the domains of quantified variables is unknown. Either the proof can be done using particular sized domains (again the Alloy [8] approach) or it can be done using an interactive theorem prover such as Isabelle or PVS[6].

3.6 Relating the implementation to the specification

The formal specifications describe requirements in abstract terms, while the implementations by necessity deal with concrete events. To carry out a formal verification, the abstract terms of the specification must be related to the implementation by a set of correspondence axioms.

We will consider an implementation with colour light signals, each with a green and a red light. Let the variables *GREEN21* and *RED21* represent the state of the lights of signal 21 (true: lit, false: extinguished) – similarly for other signals.

Suppose that the example signalling system is implemented using a computer program[7] which assigns values to the state variables of the signal lights depending on other variables representing the state of the signalling system. A formal model of the program (obtained using a formal semantics of the implementation language) include the following postcondition relations:

14. $GREEN21 \leftrightarrow LOCK21 \wedge \neg LOCK32 \wedge \neg LOCK34 \wedge TC21 \wedge (LEFT21 \wedge TC2 \wedge (\neg LOCK22 \vee LEFT22) \vee RIGHT21 \wedge TC1 \wedge (\neg LOCK22 \vee RIGHT22))$
15. $RED21 \leftrightarrow \neg GREEN21$

Here the logical variables represent parts of the concrete state of the computer program and of the devices interfaced to it. *TC21*, *TC1* and *TC2* represent the state of the train detection devices ("track circuits") of track units *PT21*, *TRACK1* and *TRACK2*, respectively. The variables are true if the units are not already occupied by trains, false otherwise. *LEFT21* and *RIGHT21* represent the state of the proving devices of points 21 — proven to be in the left or right position, respectively (or uncontrolled if both *LEFT21* and *RIGHT21* are false). Similarly for points 22.

To make the correspondence axioms below more interesting, we make the (actually quite realistic) assumption that the concrete state variables used in the implementation to represent route locking are associated with signals rather than individual routes. I.e. the variable *LOCK21* is true if some route is locked from signal 21 – similarly for the other signals.

[6] Industrilogik used PVS to prove this and similar safety conditions.

[7] ...or relay system, which are still common in railway signalling today!

In an actual implementation, equipment faults and their detection (e.g. a burnt lamp) would be an important consideration which we disregard here for simplicity.

Some of the correspondence axioms for this implementation would be

16. $proceed(SI21) \leftrightarrow GREEN21 \wedge \neg RED21$
17. $routeLocked(RT2133) \leftrightarrow LOCK21 \wedge LEFT21$

These correspondence axioms are quite simple. The first states that signal 21 displays a proceed aspect if and only if its green light is lit while its red light is not. The second states that route *RT2133* is locked if a route is locked from signal 21 while points 21 are in the left position. Frequently the abstract representations of the system state have a more complex relationship with concrete state variables.

The requirements of the formal specification, such as formula (10) can now be shown to be (part of) a refinement of the specification by demonstrating that it is a logical consequence of the implementation model, the domain and correspondence axioms and the geographical data relations.

3.7 Analysis of counterexamples

In practise, formal verifications seldom succeed completely because of mistakes in the implementation (or specification!). In the case of verification by propositional proof (SAT solving), a failure to prove the requirements will result in a counterexample being generated. Using the domain theory and correspondence axioms, such counterexamples can be used to provide information about the failure in abstract terms, rather than only in concrete terms.

Suppose the implementation incorrectly reversed the sense of the use of *LOCK22* corresponding to its first occurrence in formula (14). We assume that the rest of the implementation of the signalling system is correct. The part of the implementation controlling signal 21 would then instead be modelled by the formula

18. $GREEN21 \leftrightarrow LOCK21 \wedge \neg LOCK32 \wedge \neg LOCK34 \wedge TC21 \wedge (LEFT21 \wedge TC2 \wedge (LOCK22 \vee LEFT22) \vee RIGHT21 \wedge TC1 \wedge (\neg LOCK22 \vee RIGHT22))$

An attempt to prove requirement (10) would now fail, but it would not be obvious why. The counterexample would make formula (10) false, and include assignments to predicates such that *SI21* would be a witness to the falsity of the requirement – i.e. the universally quantified subformula $proceed(s) \rightarrow \exists r \in ROUTES\ (entry(s, r) \wedge ready(r))$ would be false when $s = SI21$. In other words $proceed(SI21)$ (and thus by (16) $GREEN21$) is true when it should not be.

The reason $proceed(SI21)$ should not be true is that $\exists r \in ROUTES\ (entry(s, r) \wedge ready(r))$ is false. Using information about the geographical data which makes the relation *entry* relate *SI21* only to the routes *RT2131* and *RT2133*, we see that this is because $ready(RT2131)$ and $ready(RT2133)$ are

both false. Looking at the assignments made by the counterexample in this case to the predicates used in the definition (11) of *ready* we will find that *routeLocked*(*RT2131*) is false (explaining why *ready*(*RT2131*) is false), while *conflictFree*(*RT2133*) is false (explaining why *ready*(*RT2133*) is false). Furthermore *left*($PT21$) is true, so the points are set for route *RT2133* suggesting that the problem is with that route. According to (11), as *conflictFree*(*RT2133*) is false, $\exists r1 \in ROUTES\ (r \neq r1 \wedge routeLocked(r1) \wedge conflict(r, r1)$ must be true. A witness for $r1$ is *RT2234* because *routeLocked*(*RT2234*) will be true in the assignment. Thus a possible explanation for the failure of the proof is that the implementation does not correctly take into account the possibility of a conflict with route *RT2234* when route *RT2133* is locked. Investigation of formula (18) confirms this hypothesis.

This shows how a domain theory can help in both pinpointing and explaining errors in the implementation. This is particularly important when communicating the findings to the domain experts.

The verification tool used by Industrilogik can assist in giving such "high-level" explanations. (`spec_10` is the identifier of requirement formula 10. `#` is the symbol used by GTO for logical disjunction.)

```
> load sample3
> prove spec_10
The formula is falsifiable.
> why spec_10
 Formula is FALSE because s=SI21, proceed(SI21)&
 ~SOME r (entry(SI21,r)&ready(r))
> why SOME r (entry(SI21,r) & ready(r))
 Formula is FALSE because ~(ready(RT2131)#ready(RT2133))
> why ready(RT2131)
 Formula is FALSE because ~routeLocked(RT2131)
> why ready(RT2133)
 Formula is FALSE because ~conflictFree(RT2133)
> why conflictFree(RT2133)
 Formula is FALSE because r1=RT2234, RT2133<>RT2234&
 routeLocked(RT2234)&conflict(RT2133,RT2234)
```

4 Conclusions and future work

We have given examples of how practical formal specification and verification work can be facilitated by using a domain theory which is directly available to the specification and verification tools. The examples have been taken from all phases of actual formal methods projects – from developing formal specifications to analysis of verification results. The discussed techniques presuppose the existence of a formal theory of the application domain, which shows that such a theory has a concrete practical use.

Of the tasks discussed, some could benefit from a greater degree of automation.

In practise the system to be verified is never completely correct, so the analysis of counterexamples is important [4]. It is also a task which has very little automation support and which has not attracted much research. In our experience, this takes an unproportional amount of time compared with other tasks. Finding techniques to automate the analysis will have clear practical advantage.

Techniques based on propositional satisfiability solving require that the sizes of sets which are quantified over are known in advance. When they are not (e.g. when attempting to prove general correctness properties of a specification) general theorem provers such as Isabelle or PVS must be used. This is a serious drawback, as automated tools are of major importance for the successful deployment of formal methods in industry.

Work on fully automatic techniques for handling parameterised systems of undetermined size (e.g. regular model checking [1]) opens the possibility of automatically carrying out proofs involving generic specifications for configurable systems. Furthermore such techniques will provide countermodels for failed proofs. We are currently investigating the possibility of using such techniques.

References

1. Abdulla, P. A., Jonsson, B., Nilsson, M., d'Orso, J. and Saksena, M., Regular Model Checking for MSO+LTL, In Alur, R. and Peled, D. (eds.): Computer Aided Verification, Proceedings of the 16th International Conference (CAV'04), Springer Lecture Notes in Computer Science 3114, Springer-Verlag (2004).
2. Bjørner, D., Formal Software Techniques for Railway Systems. In: Schnieder, E: (ed.), 9th IFAC Symposium on Control in Transportation Systems, pp 1–12, Technical University, Braunschweig, Germany (2000).
3. Eriksson, L-H., Specifying Railway Interlocking Requirements for Practical Use, In Schoitsch, E. (ed.): Proceedings of the 15th International Conference on Computer Safety, Reliability and Security (SAFECOMP'96), Springer-Verlag (1996).
4. Eriksson, L-H., Using Formal Methods in a Retrospective Safety Case, In Heisel, M., Liggesmeyer, P., Wittmann, S. (eds.): Computer Safety, Reliability, and Security – 23rd International Conference, SAFECOMP 2004, Springer Lecture Notes in Computer Science 3219, Springer-Verlag (2004).
5. Eriksson, L-H., The GTO toolset and method, Technical Report 2006-030, Uppsala University, Dept. of Information Technology (2006).
6. Eriksson, L-H. and Johansson, K., Using formal methods for quality assurance of interlocking systems, In Mellit, B. et.al. (eds.): Computers in Railways IV, Computational Mechanics publications (1998).
7. Eriksson, L-H. and Fahlén, M., An Interlocking Specification Language, In ASPECT IRSE 99, Papers of the International Conference, Institute of Railway Signalling Engineers, London (1999).
8. Jackson, D., Alloy: a lightweight object modelling notation, ACM Transactions on Software Engineering and Methodology, Vol. 11 No. 2, pp. 256–290 (2002).
9. Morley, M. J., Safety Assurance in Interlocking Design, Ph.D. thesis, University of Edinburgh (1996).
10. Peleska, J., Große, D., Haxthausen, A. E. and Drechsler, R., Automated Verification for Train Control Systems, In Schnieder, E. and Tarnai, G. (eds):

FORMS/FORMAT 2004 - Formal Methods for Automation and Safety in Railway and Automotive Systems, Braunschweig, Germany (2004).
11. Simpson, A. C., Woodcock, J. C. P., and Davies J. W., The mechanical verification of Solid State Interlocking geographic data. In Groves, L. and Reeves, S. (eds.), Proceedings of Formal Methods Pacific, Wellington, New Zealand, 9–11 July, pp. 223–242. Springer-Verlag (1997).