Adtranz Signal's

Formal Verification Process (2)

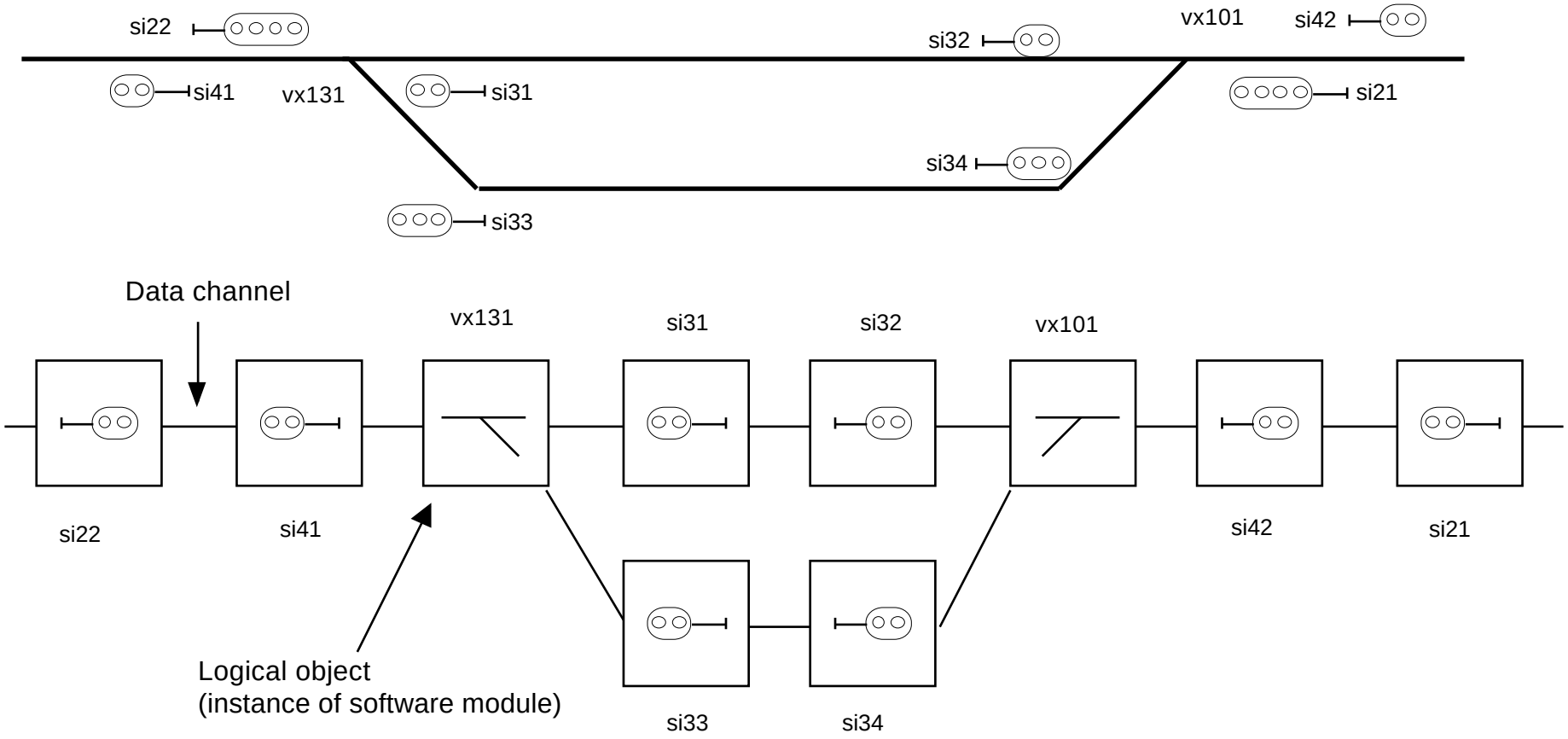
The STERNOL Specification Tool (SST)

Lars-Henrik Eriksson
Industrilogik L4i AB

Industrilogik L4i AB

- Swedish consultancy company in advanced logic, mathematics and computer science.
- Main business area: Formal methods
- Staff of 11. 7 Ph.D.
- Founded in 1997, but most of the staff worked industrially with formal methods before that.

Structure of Ebilock interlocking software



Detail of a logical object

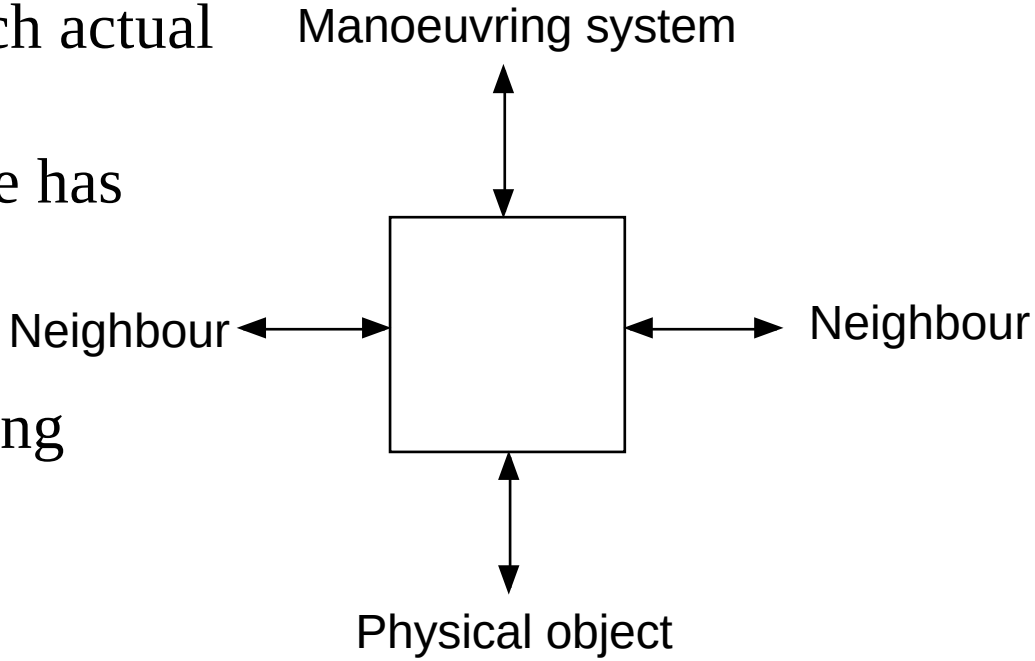
The program code of a logical object is written in STERNOL, Adtranz Signal's proprietary language for interlocking software.

Each *type* of logical object has its own program module.

Modules are *instantiated* with local configuration data for each actual logical object.

Each logical object instance has a local data area.

Communication is done using shared variables.



Formal verification using SVT

STERNOL code can be formally verified using Prover Technology's SVT tool.

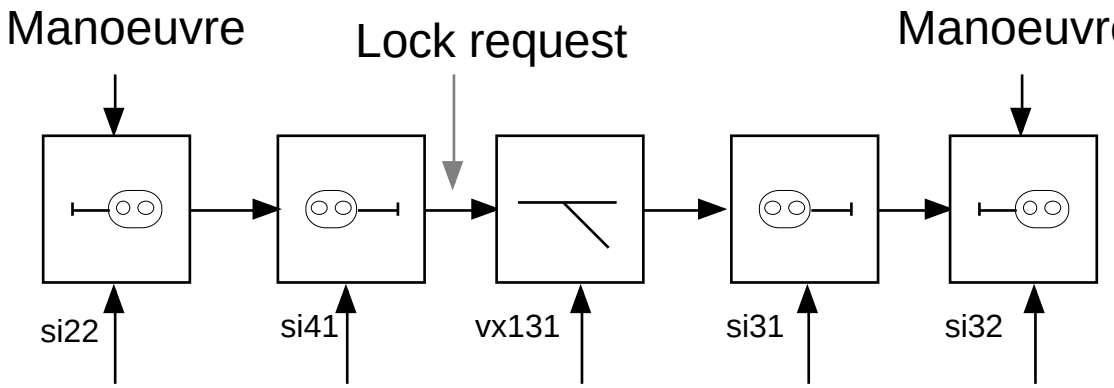
Verification can be done on

- the uninstantiated code for a logical object type
  + can be done once, valid for *every* interlocking installation
  – difficult to verify high-level requirements
- the instantiated code for an entire interlocking system
  – must be done separately for *each* interlocking installation
  + easy to verify high-level requirements

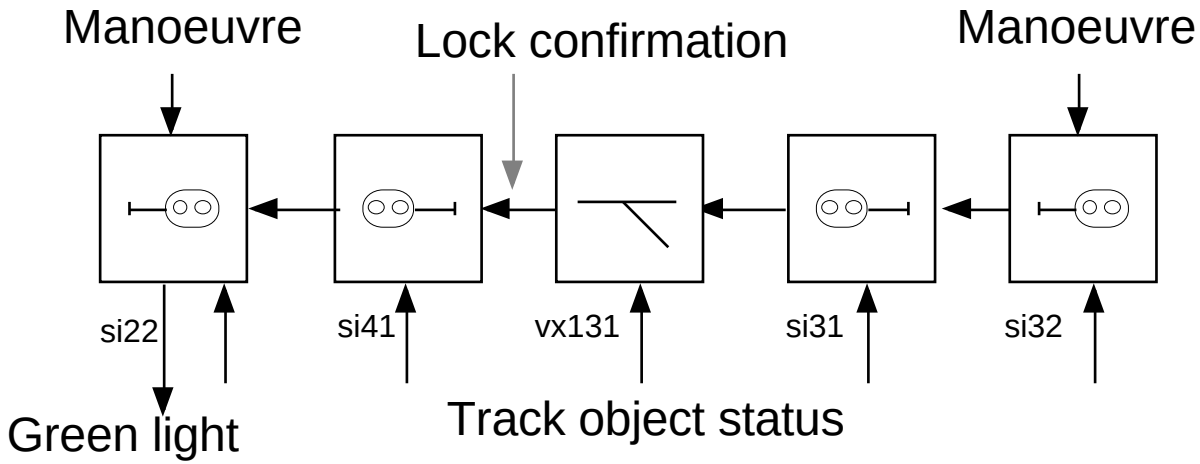
The specification language of SVT is propositional logic extended with arithmetic. (However, arithmetic theorem proving capability is limited.)

Train route locking process

(1)



(2)



Logical object-level requirements

Sample requirement for a clear signal aspect:

- All track circuits in the route must be unoccupied for the signal to clear.

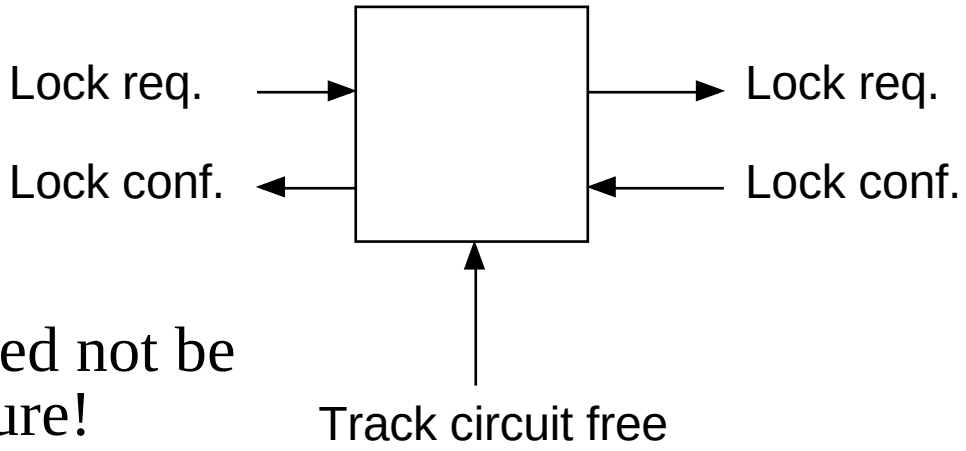
How is this expressed as a requirement on the logical objects?

”If the object sends a lock confirmation towards the beginning of the route, then it must receive a lock request from the beginning of the route, receive a lock confirmation from the end of the route and the track circuit must be free”

Wrong level of abstraction!

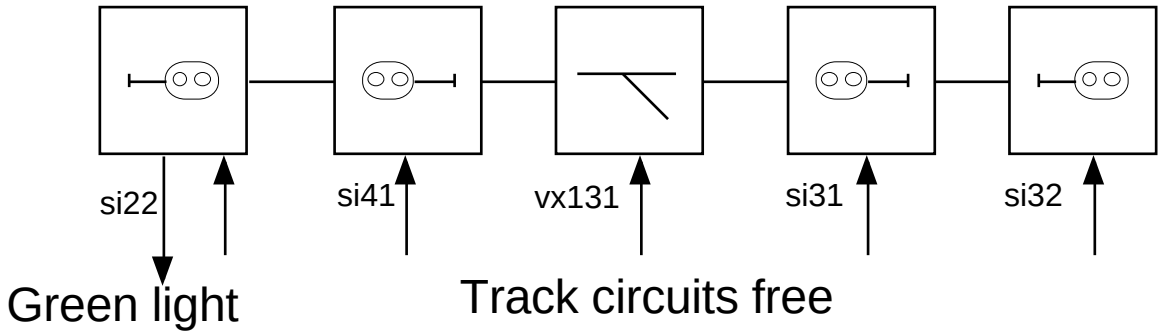
Difficult to prove formally since much irrelevant detail must be considered.

A good design architecture need not be a good specification architecture!



Rail yard level requirements

Alternatively, express requirement for a complete *actual* route!



If signal SI22 is cleared, then the track circuits of objects SI22, SI41, VX131, SI31 and SI32 must all be unoccupied.

Better level of abstraction. Easy to prove formally.

Drawbacks: A *separate* requirement must be stated for *each* route. The requirements become specific to each particular installation.

Generalised requirements

Expressing requirements of an entire route means that the requirements must be tailored to *each route* in *each installation*.

A *single general* requirement would be much better!

The SVT requirements language (essentially propositional logic) does not support generalised requirements.

Solution: Introduce more powerful specification language (IRSL – Interlocking Requirements Specification Language) based on predicate logic to allow general requirements.

For all train routes: if the route entry signal is clear, every track circuit belonging to the route must be unoccupied.

Finally the right level of abstraction!

Translation of specifications

Since SVT can not handle the specifications directly, they must be translated into propositional logic.

The actual configuration data for an interlocking installation is used for the translation.

The specification ”For all train routes: if the route entry signal is clear, every track circuit belonging to the route must be unoccupied”

is translated into the specification

”If signal SI22 is cleared, then the track circuits of objects SI22, SI41, VX131, SI31 and SI32 must all be unoccupied **AND**
If signal SI31 is cleared, then the track circuits of objects SI31, VX131 and SI41 must all be unoccupied **AND**
If signal SI32 is cleared, then the track circuits of objects SI32, VX101 and SI42 must all be unoccupied **AND** ……”

This translation is done by the SST.

The Interlocking requirements specification language

IRSL is basically predicate logic.

IRSL includes bounded quantifiers,

e.g. $x{:}S\ P(x)$ = For all x in the set S, $P(x)$

IRSL includes constructs (functions and predicates) that refer to interlocking configuration data,

e.g. *leg(x,n)* = Neighbour number *n* of logical object *x*.

e.g. *POINT* = The set of all point logical objects

IRSL presently does not include temporal operators.

Translation of IRSL formulae

SST translates IRSL formulae into SVT specification language formulae according to some interlocking configuration data.

- Quantification over finite sets are expanded:

  x:*POINT P(x)* -----> *P(VX101)* *P(VX131)*

- Constructs referring to interlocking configuration data are *replaced* with the data in question:

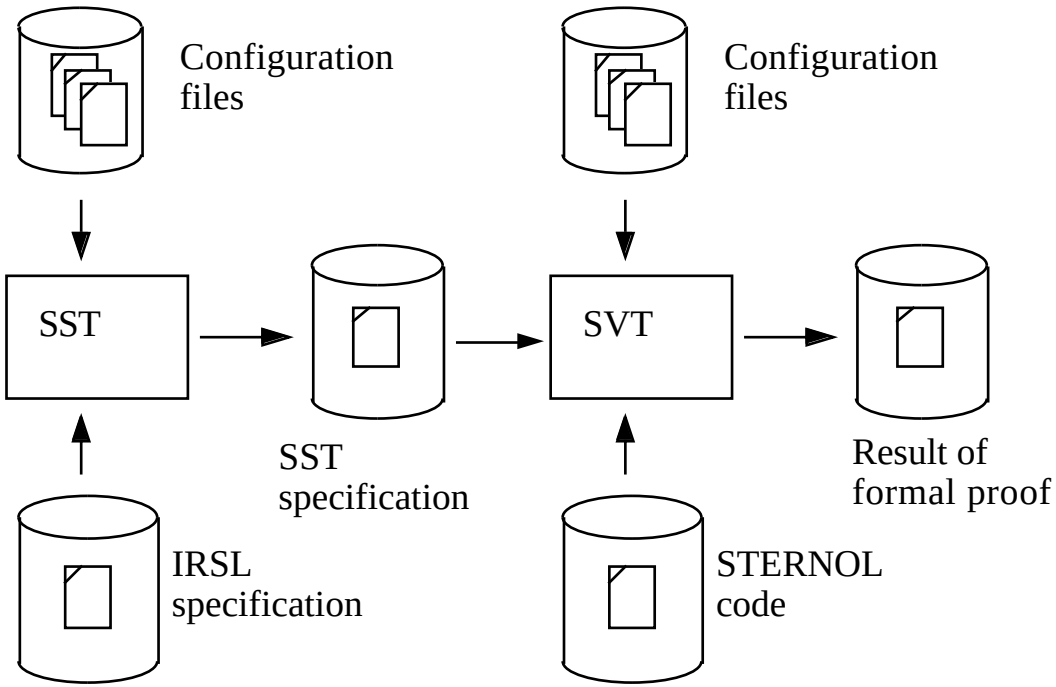
  leg(SI22,1) = SI21 -----> *SI42 = SI21*

- The formulae are *simplified* using known information

  (*SI42 = SI21* → *P)* *Q* -----> *Q*

This process is often called *partial evaluation*.

Using SST



(Simplified verification process)

Additional uses of SST

- To formally check interlocking configuration files.

  Specifications can be written to express requirements on the configuration data itself.

- To aid in the development of an interlocking site by *finding* all train routes.
  - A formal specification of train routes is written.
  - SST translates the specification into a propositional formula characterising all valid train routes.
  - SVT is used to find all *models* of the formula.
    Each model represents a train route.

Formal verification of SST itself

The partial evaluator kernel of SST has been formally verified.

- Written in Prolog
- Formally verified using the LPTP tool (Logic Program Theorem prover)
- One error found!

Verification condition: