Report T97:04

ISRN : SICS-T–97/04-SE
ISSN : 1100-3154

Using Formal Methods
A practical comparison between Z/EVES and PVS

by

Daniel Fredholm

FDT; 971204

Swedish Institute of Computer Science
Box 1263, S-164 28 KISTA, SWEDEN

Abstract

This paper consists of a review and comparison between Z/EVES and PVS– two tools designed for analyzing formal specifications. Z/EVES is a tool for analyzing specifications written in Z. PVS is a general theorem prover for a language that consists of higher order logic together with set theory.

The review has its focus on the possibility to use these tools in an industrial context. The plan for the review was to get acquainted with the tools on a general level and then to use them to partially validate a formal specification of requirements for the safety function of railway signaling systems.

The conclusion is that PVS is clearly superior to Z/EVES. PVS has such a good performance that it can be recommended for industrial use in the area of formal methods. Concerning Z/EVES, its applicability seems more restricted.

Keywords: formal methods, Z/EVES, Z, PVS, railway signaling

1 Introduction

In software development, there is a growing awareness of the importance of writing useful specifications of the programs that are to be developed. For a specification to be useful, it should at least satisfy the following:

- It should be written with such precision that some of the questions concerning the resulting program can be transferred to questions concerning the specification; these can then be dealt with before the actual implementation, thus saving time and money and increasing the safety and quality of the program.
- It should be written with such a precision that it allows the implementation to be verified with respect to it.

This need of precision leads to the solution of letting the specification be a piece of mathematics: The specification states the existence of a mathematical object (or objects) which is to satisfy a collection of requirements or axioms.

Questions concerning the resulting program can then be stated as possible theorems in the theory described by the specification and, since one has entered the realm of mathematics, one can attempt to prove or refute them.

This is the general idea of the concept of formal methods in software development.

In practice, specifications usually consist of a fairly large number of axioms of quite a simple character. Also, most of the questions asked are, for the educated human, fairly trivial. This has led to the development of computer–based tools for analyzing specifications.

In general, the core of such a tool consists of two things:

- A formal language in which the specification and the possible theorems are to be written. Usually, this language is based on concepts from mathematical logic, where they originally where developed for investigating the foundations of mathematics.
- A theorem prover, where one can attempt to prove the stated theorems. Such a prover is based on the concept of a formal proof, which also originated from mathematical logic. Usually, the process of proving a theorem with the prover is not fully mechanized. The user is expected to interactively tell the prover which rules, strategies and/or lemmas to use. The prover handles bookkeeping such as naming variables, checking whether a particular rule is applicable, etc. and, in simple cases, automatically proves the theorem or some part of it. In the ideal situation, it handles the trivial parts of the proof and lets the user concentrate on the parts of the proof that need some creative effort.

There are also devices for syntax– and type-checking, which can be very helpful in detecting gross errors in the specification.

This paper is a review and comparison of two such tools: Z/EVES and PVS. In the world of formal methods, they are both well–known and considered to be among the best of their kind.

The plan for this review was first to get acquainted with the tools on a general level and then to use them in a concrete application. This application was a formal specification of requirements for the safety function of railway signaling systems [**?**]. The specification is the major result of an ongoing study, led by the Swedish National Rail Administration (Banverket), of the possibility of using formal methods in the area of railway safety.

2 Z/EVES

Z/EVES is developed, supported and distributed by ORA Canada. It is the amalgamation of the Z–specification language and the EVES prover.

Z is one of the most commonly used languages for writing specifications [**?**, **?**]. It is based on Zermelo–Fraenkel set theory. Since virtually all known mathematics can be expressed in this theory, the expressive power is huge: almost all reasonable specifications can be written in Z. This does not mean that it can be done in the most natural and/or readable way, but since what is to be considered as natural and readable is, to a large extent, governed by personal taste and habit, this is not a serious problem. My point is merely that it can take some time to get accustomed to Z, since most people are not used to work directly in set theory.

Let us turn to the prover EVES. To my view, this is the weak link of the system. In order to explain my critique, let me make some general remarks about theorem–proving in general.

There are two major reasons for proving theorems:

First, we like to establish the truth–value of the statement in question; in particular; we want to know if the statement is true.

Second, we like to know *why* it is true. In other words: we want an argumentation which convinces us of the truth of the statement. This argumentation is the proof itself.

These aims are obviously not separated. On the contrary, one could say that establishing the truth–value is the goal, whereas constructing the argumentation is the means by which this goal is to be achieved.

There is another important point here: not only do we want to convince ourselves of the truth of a statement; we also want to convince others. Hence, it is important to have access to the proof itself and not only to the result of the proof–process.

These remarks have implications when theorems are to be proved with the use of a computer. Feeding the statement to the prover, which computes on its own for some time and then terminates with the answer "true"or "false", is generally not an agreeable situation, since it does not give you any clue to how it arrived at this conclusion. In particular, if the answer was "false", there is an indication that something is wrong, either with the statement or the theory,

and it is of vital importance to have more information in order to remove the problem.

Of course, there are situations when this is acceptable, mostly depending on the complexity of the statement to be proved. If, for instance, one is proving a propositional statement, it is usually not necessary to see the actual proof in order to feel convinced of its validity. But this is more the exception than the rule, since usually the statement is of such an involved nature that one actually wants to examine the proof in order to convince oneself and others of its correctness (this is in particular true for a language such as Z).

This brings us back to the EVES–prover. It falls in the trap described above in the sense that the commands provided to build proofs are too powerful, to mechanized and thus non-informative. I am not saying that such commands are to be banned, but simply that they should be complemented with other commands that are not–so–powerful. Not only are such commands more informative, they also give the user more control over the proof–process. This is particularly useful when things go astray (which they often do).

The proof–commands in EVES relies heavily on a database of internal axioms (called the Mathematical Toolkit). In its current state, this database seems to be in need of additions, judging from the poor performance of the system. One obvious way to increase the performance would be to increase the database. This, however, has to be done by the user.

Also, in EVES, the actual proofs are not accessible by the user, again failing with respect to the above remarks. This has other drawbacks. For instance, the only way to repeat a proof of a statement, or use it to prove some other statement which one suspect has an analogous proof, is to type it by hand.

In general, the interface is primitive.

Another major drawback with Z/EVES is that it is poorly documented [**?**].

3 PVS

PVS is developed, supported and distributed by SRI International. The PVS specification language is based on higher–order logic together with set theory; i.e. it is possible to quantify not only over individuals such as integers, but also over functions and predicates. Since higher–order logic and set theory are the basis of general mathematics, the expressive power of the language is, in most applications, more than enough. It should be noted though that this analogy with mathematics in general has the effect that the language is very flexible and that it is comparatively easy to express things in a natural way.

Two useful features of the language are the predicate subtype construction and the abstract data-type construction.

The predicate subtype construction allows the definition of a type that consists of those elements of a given type that satisfy a given predicate. A typical example would be the type of non–zero real numbers.

The abstract data-type construction allows the definition of inductively defined types in terms of constructors and accessors. The typical example here is

lists of elements of some given type. This construction allows properties of such types to be proved by induction and functions with such a type as domain to be defined by recursion. Hence, the induction and/or recursion can be performed over the data-type directly and not over the natural numbers via some coding. Again taking lists as an example, this means that the inherent structure of a list in terms of `nil` and `cons` can be used directly in an induction and not via some natural number associated to the list such as its length.

The prover has a lot to recommend it. It has a well–chosen kernel of primitive commands for building proofs. These commands are easy to understand and use, since their effects are quite limited and thus predictable. By restricting the usage to these commands alone, the prover behaves more like an editor for proofs. However, these commands can be combined to form more powerful commands, called strategies, thus raising the level of mechanization. The system provides a number of such commands: there are commands for carrying out propositional, equality and linear arithmetic reasoning with the use of definitions and lemmas and also commands for induction. There is also the possibility for the user to define his own strategies.

The proof–building commands provided by the system represents a good mixture of high and low levels of mechanization, thus yielding a system which is flexible, informative and easy to use.

The interface is quite elaborate. For instance, proofs can be viewed, saved and edited. The presentation of the proofs is a bit crude, but it is still useful.

Another feature is the possibility to display proof dependencies; given a proof, the system can display all the lemmas, definitions and axioms that are used in the proof. In principle, this is not hard to do by hand, provided one has access to the proof. But it can be quite tedious, in particular if the proof and/or the specification is large.

The interface also provides an extensive help-function.

The documentation is also quite elaborate [**?**, **?**, **?**, **?**, **?**]. It is well–written and covers the important parts of the system.

To conclude, PVS is a well–designed and mature system in its genre. There are of course things that could be improved, but the present state is more than acceptable.

4 Examples

To give the reader an illustration of the tools, let us consider an elementary theorem from arithmetic and see how it can be expressed and proved in them. The theorem is the following:

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

The usual way to prove this is to argue by induction over n. In formal languages like Z/EVES and PVS, one normally defines the summation as a recursive func-

tion; i.e. what is proved is this statement:

$$\text{sum}(n) = \frac{n(n+1)}{2}, \quad \text{where} \quad \text{sum}(n) = \text{if } n = 0 \text{ then } 0 \text{ else } n + \text{sum}(n-1)$$

Let us begin with Z/EVES. Z/EVES does not support recursion and induction directly, so we have to express things a little different. Observe that Z/EVES uses a LaTeX-based syntax.

```
\begin{axdef}
sum : \nat \fun \nat
\where
\Label{rule defsum0}
sum 0 = 0\\
\Label{rule defsum1}
\forall n : \nat @ sum(n+1) =sum(n)+n+1
\end{axdef}
```

This declaration states that sum is a function with the right domain and range and that it satisfies the obvious computation rules. As stated earlier, Z/EVES does not support induction directly, but it is incorporated as an axiom. The way to prove the statement is the following: we define the set `CFvalues` as the set of natural numbers for which the formula holds, prove that this set contains 0 and that it is closed under the successor operation. Finally, we show that the set of natural numbers is included in this set, by the use of the induction axiom. Hence, we get the following declarations:

```
\begin{zed}
CFvalues == \{ n : \nat |  2*sum(n)=(n*(n+1)) \}
\end{zed}

\begin{theorem}{closedform0}
0 \in CFvalues \land (\forall x: CFvalues @ x+1 \in CFvalues)
\end{theorem}

\begin{theorem}{closedform}
\nat \subseteq CFvalues
\end{theorem}
```

Note that in the definition of `CFvalues`, we have multiplied the original formula with 2. This is due to the fact that Z/EVES only support natural numbers and integers.

These declarations are then input to Z/EVES. We then begin the proof by proving `closedform0`:

```
=>try lemma closedform0;
Beginning proof of closedform0 ...
```

```
      0 \in CFvalues \\
\land (\forall x: CFvalues @ x + 1 \in CFvalues)
=>reduce;
Which simplifies
with invocation of CFvalues
when rewriting with [internal items]
forward chaining using knownMember, [internal items]
with the assumptions natType, [internal items] to ...

      0 \in \nat \\
\land (\forall x: \{ n: \nat | 2 * sum n = n * (n + 1) \}
         @ (      1 + x \in \nat \\
            \land 2 * sum (1 + x) = (1 + x) * (2 + x)))
=>prenex;
Prenexing produces ...

      0 \in \nat \\
\land ( x \in \{ n: \nat | 2 * sum n = n * (n + 1) \} \\
       \implies    1 + x \in \nat \\
                \land 2 * sum (1 + x) = (1 + x) * (2 + x))
=>reduce;
Which simplifies
when rewriting with [internal items], inNat
forward chaining using knownMember, [internal items]
with the assumptions natType, [internal items] to ...

               x \in \nat \\
         \land 2 * sum x = x * (1 + x) \\
\implies       1 + x \geq 0 \\
         \land 2 * sum (1 + x) = (1 + x) * (2 + x)
=>use defsum1;
Assuming [internal items] generates ...

    (\forall  | n \in \nat @ sum (n + 1) = sum n + (n + 1)) \\
         \land x \in \nat \\
         \land 2 * sum x = x * (1 + x) \\
\implies       1 + x \geq 0 \\
         \land 2 * sum (1 + x) = (1 + x) * (2 + x)
=>instantiate n == x;
Instantiating n = x gives ...

               (         x \in \nat \\
                \implies sum (x + 1) = sum x + (x + 1)) \\
\land(\forall | n \in \nat @ sum (n + 1) = sum n + (n + 1)) \\
        \land x \in \nat \\
        \land 2 * sum x = x * (1 + x) \\
```

```
\implies       1 + x \geq 0 \\
         \land 2 * sum (1 + x) = (1 + x) * (2 + x)
=>reduce;
Which simplifies
when rewriting with inNat
forward chaining using knownMember, [internal items]
with the assumptions natType, [internal items] to ...
true
```

We then continue with `closedform`

```
=>try lemma closedform;
Beginning proof of closedform ...
\nat \subseteq CFvalues
=>use closedform0;
Assuming closedform0 generates ...

               0 \in CFvalues \\
         \land (\forall x: CFvalues @ x + 1 \in CFvalues) \\
\implies \nat \subseteq CFvalues
=>use natInduction;
Assuming natInduction generates ...

               (\forall

                    |       S \in \power \num \\
                      \land 0 \in S \\
     \land (\forall x: S @ x + 1 \in S) @ \nat \subseteq S) \\
         \land 0 \in CFvalues \\
 \land (\forall x\_\_0: CFvalues @ x\_\_0 + 1 \in CFvalues) \\
\implies \nat \subseteq CFvalues
=>instantiate S == CFvalues;
Instantiating S = CFvalues gives ...

               (               CFvalues \in \power \num \\
                         \land 0 \in CFvalues \\
          \land (\forall x: CFvalues @ x + 1 \in CFvalues) \\
                \implies \nat \subseteq CFvalues) \\
         \land (\forall

                    |       S \in \power \num \\
                      \land 0 \in S \\
                \land (\forall x\_\_0: S @ x\_\_0 + 1 \in S)
                  @ \nat \subseteq S) \\
         \land 0 \in CFvalues \\
 \land (\forall x\_\_1: CFvalues @ x\_\_1 + 1 \in CFvalues) \\
```

```
\implies \nat \subseteq CFvalues
=>with enabled (inPower, subDef) rewrite;
Which simplifies
when rewriting with subsetDef, inPower
forward chaining using knownMember, [internal items]
with the assumptions select\_2\_1, select\_2\_2, natType, [internal items]
with the instantiation x = x\_\_1 to ...

              \lnot (\forall e: CFvalues @ e \in \num) \\
         \land (\forall

             |       (\forall e\_\_0: S @ e\_\_0 \in \num) \\
                      \land 0 \in S \\
                      \land (\forall x: S @ 1 + x \in S)
                  @ (\forall e\_\_1: \nat @ e\_\_1 \in S)) \\
         \land 0 \in CFvalues \\
\implies \lnot
   (\forall x\_\_0: CFvalues @ 1 + x\_\_0 \in CFvalues)
=>prenex;
Prenexing produces ...

              \lnot (         e \in CFvalues \\
                     \implies e \in \num) \\
         \land (\forall

              |       (\forall e\_\_0: S @ e\_\_0 \in \num) \\
                      \land 0 \in S \\
                      \land (\forall x: S @ 1 + x \in S)
                  @ (\forall e\_\_1: \nat @ e\_\_1 \in S)) \\
         \land 0 \in CFvalues \\
\implies \lnot
    (\forall x\_\_0: CFvalues @ 1 + x\_\_0 \in CFvalues)
=>reduce;
Which simplifies
with invocation of CFvalues
when rewriting with weakening, [internal items]
forward chaining using knownMember, [internal items]
with the assumptions natType, [internal items] to ...
true
```

Let us turn to PVS. There, the initial declarations has the following form:

```
sum : THEORY

  BEGIN
```

```
  n: VAR nat
  sum(n): RECURSIVE nat =
   (IF n=0 THEN 0 ELSE n + sum(n-1) ENDIF)
   MEASURE (LAMBDA n:n)
  closed_form: THEOREM sum(n)=(n*(n+1))/2

  END sum
```

One proof of the theorem then looks as follows:

```
<rcl>
closed_form :

  |-------
{1}    (FORALL (n: nat): sum(n) = (n * (n + 1)) / 2)

Rule?: (INDUCT "n")
Inducting on n,
this yields  2 subgoals:
closed_form.1 :

  |-------
{1}    sum(0) = (0 * (0 + 1)) / 2

Rule?: (EXPAND "sum")
Expanding the definition of sum,
this simplifies to:
closed_form.1 :

  |-------
{1}    (0 = 0 / 2)

Rule?: (ASSERT)
Simplifying, rewriting, and recording with decision procedures,

This completes the proof of closed_form.1.

closed_form.2 :

  |-------
{1}    (FORALL (j: nat):
         sum(j) = (j * (j + 1)) / 2
             IMPLIES sum(j + 1) = ((j + 1) * (j + 1 + 1)) / 2)

Rule?: (SKOSIMP)
Skolemizing and flattening,
```

```
this simplifies to:
closed_form.2 :

{-1}    sum(j!1) = (j!1 * (j!1 + 1)) / 2
  |-------
{1}    sum(j!1 + 1) = ((j!1 + 1) * (j!1 + 1 + 1)) / 2

Rule?: (EXPAND "sum" 1)
Expanding the definition of sum,
this simplifies to:
closed_form.2 :

[-1]  sum(j!1) = (j!1 * (j!1 + 1)) / 2
  |-------
{1}   (1 + sum(j!1) + j!1 = (2 + j!1 + (j!1 * j!1 + 2 * j!1)) / 2)

Rule?: (REPLACE -1)
Replacing using formula -1,
this simplifies to:
closed_form.2 :

[-1]    sum(j!1) = (j!1 * (j!1 + 1)) / 2
  |-------
{1}    (1 + (j!1 * (j!1 + 1)) / 2 + j!1 =
                (2 + j!1 + (j!1 * j!1 + 2 * j!1)) / 2)

Rule?: (ASSERT)
Simplifying, rewriting, and recording with decision procedures,

This completes the proof of closed_form.2.

Q.E.D.
```

This is a nice illustration of an informative, non-mechanized proof. We also show a more mechanized and thus non–informative proof, using the provided strategy `INDUCT-AND-SIMPLIFY`. It looks as follows:

```
<rcl>
closed_form :

  |-------
{1}    (FORALL (n: nat): sum(n) = (n * (n + 1)) / 2)

Rule? (INDUCT-AND-SIMPLIFY "n")
sum rewrites sum(0)
  to  0
```

```
sum rewrites sum(1 + j!1)
  to  1 + sum(j!1) + j!1
By induction on n, and by repeatedly rewriting and simplifying,
Q.E.D.
```

5 Conclusions

The general conclusion is that when comparing Z/EVES and PVS, PVS is the clear winner. PVS has attained a high level of maturity, whereas Z/EVES seems more like a somewhat primitive prototype. This should be clear from the above descriptions, but let me add some statistics to strengthen this conclusion: The PVS prover provides 95 proof–building commands and 17 commands for defining strategies. The corresponding figures for Z/EVES are 27 and 0, respectively. Also, a comparison between the amount of documentation provided comes out in favor of PVS by a factor 2.

Such a crude statistical comparison is of course not decisive in itself; it does not exclude the possibility that PVS contains a lot of unnecessary features which Z/EVES avoids. However, from the descriptions above, it should be clear that this is not the case.

Both Z/EVES and PVS are available for free. The relevant addresses are [?] for PVS and [?] for Z/EVES.

A The case–study

As stated in the introduction, the plan for the review was to begin by testing the tools on a general level and then use them to partially validate a concrete specification.

The specification in question formally specifies some requirements for the safety function of railway signaling systems. It was originally written for the specification tool Delphi. The task was to translate this specification to the languages of PVS and Z/EVES and then to prove 3 theorems about it. These theorems all state that some safety–critical criteria hold.

A version of this specification written for PVS is included below. The theorems are at the end of the specification. They were all successfully proved in PVS. The level of mechanization was comparable to the first PVS-proof of the example `closed_form`. The execution-time of singular PVS-commands was short enough in order for the system to used interactively. The total time for finishing the proofs was approximately four working-hours.

I like to point out, as the reader soon will find out for himself, that the specification is not presented in a particularly readable form; such a presentation can be found in [?]. It is included here for the sake of completeness and to give the reader some flavor of how an application of formal methods in general, and

PVS in particular, can look. I also like to point out that the specification was not translated to Z, since already on a general level, Z/EVES was judged too awkward to use.

```
banverket: THEORY
BEGIN
bangardsobjekt : NONEMPTY_TYPE
signal, sparsegment : TYPE FROM bangardsobjekt
ax_dis_1 : AXIOM (FORALL (x : signal, y : sparsegment) :
NOT sparsegment_pred(x) AND NOT signal_pred(y))

forsignal, hsi_lykta  : TYPE FROM signal
ax_dis_2 : AXIOM (FORALL (x : forsignal, y : hsi_lykta) :
NOT hsi_lykta_pred(x) AND NOT forsignal_pred(y))

huvudsignal, stopplykta : TYPE FROM hsi_lykta
ax_dis_3 : AXIOM (FORALL (x : huvudsignal, y : stopplykta) :
NOT stopplykta_pred(x) AND NOT huvudsignal_pred(y))

vaxel : TYPE FROM sparsegment
omrade : TYPE
vag : TYPE FROM omrade
tagvag : TYPE FROM vag
andsegment : TYPE FROM sparsegment
stoppbock : TYPE FROM andsegment
linje : TYPE FROM andsegment
bo : VAR bangardsobjekt
si : VAR signal
si1 : VAR signal
si2 : VAR signal
fsi : VAR forsignal
fsi1 : VAR forsignal
fsi2 : VAR forsignal
hsisl : VAR hsi_lykta
hsi : VAR huvudsignal
hsi1 : VAR huvudsignal
hsi2 : VAR huvudsignal
sl : VAR stopplykta
om : VAR omrade
om1 : VAR omrade
om2 : VAR omrade
ss : VAR sparsegment
ss1 : VAR sparsegment
ss2 : VAR sparsegment
ss3 : VAR sparsegment
ss4 : VAR sparsegment
```

```
vx : VAR vaxel
tv : VAR tagvag
tv1 : VAR tagvag
tv2 : VAR tagvag
vg : VAR vag
vg1 : VAR vag
vg2 : VAR vag
v : VAR nat
v1 : VAR nat
v2 : VAR nat
li : VAR linje
as : VAR andsegment
sb : VAR stoppbock
vansterlage(x: vaxel):bool
hogerlage(x: vaxel):bool
korbesked(x: tagvag,y: huvudsignal):bool
tagvagsutlosning(x:sparsegment,y:tagvag):bool
slutpunktsutlosning(x:sparsegment,y:tagvag):bool
slutdel(x:sparsegment,y:tagvag):bool
sida_till(x:sparsegment,y:tagvag):bool
ansluter_till(x:sparsegment,y:sparsegment):bool
fore(x:sparsegment,y:vag):bool
medriktad(x:bangardsobjekt,y:vag):bool
motriktad(x:bangardsobjekt,y:vag):bool
skyddsobjekt_till(x:bangardsobjekt,y:tagvag):bool
skyddsstracka_till(x:vag,y:tagvag):bool

del_av(x:sparsegment,y:omrade):bool
segments(z:vag):TYPE = { x:sparsegment | del_av(x, z)}
fin_seg:AXIOM (FORALL (z:vag)  :
           (EXISTS (m : nat, g : [below(m) -> segments(z)])
                     bijective?(g)))
% Detta ser till att en vaeg bestaar av ett aendligt antal
%  spaarsegment.

i_konflikt(x:omrade,y:omrade):bool
vanster_gren(x:sparsegment,y:vaxel):bool
hoger_gren(x:sparsegment,y:vaxel):bool
framfor(x:sparsegment,y:signal):bool
bakom(x:sparsegment,y:signal):bool
atc_forbesked(x:forsignal,y:nat):bool
atc_huvudbesked(x:huvudsignal,y:nat):bool
sth(x:tagvag,y:nat):bool
ur_kontroll(bo) : bool
kor1(hsi):bool
kor2(hsi):bool
```

```
kor3(hsi):bool
belagd(ss):bool
vaxellast(vx):bool
lokalfrigiven(vx):bool
frigivningsbar(vx):bool
lagd(vg):bool
hinderfri(vg):bool
skyddsstracka_lagd(vg):bool
skyddad_bakat(vg):bool
ej_mojlig_skyddsstracka(vg):bool
klar(tv):bool
konfliktfri(tv):bool
sidoskyddad(tv):bool
slutpunktsskyddad(tv):bool
aldrig_skyddsstracka(tv):bool
last(tv):bool
slutpunktlast(tv):bool
kort(tv):bool
varsamhet(tv):bool
tidsutlosning(tv):bool
huvudtagvag(tv):bool
vanta_stopp(fsi):bool
vanta_kor1(fsi):bool
vanta_kor2(fsi):bool
stopp(x:hsi_lykta):bool
nodstopp:bool
obevakad:bool
kan_bli_bevakad:bool
kan_bli_obevakad:bool
slutskyddande_vaxel(x:vag,y:vaxel):bool
skyddsstracka_konflikt(x:vag,y:tagvag):bool
bakom2(ss,si,vg):bool

linje(ss):bool = EXISTS li: (li=ss)
vaxel(ss):bool = EXISTS vx: (vx=ss)
andsegment(ss):bool = EXISTS as: (as=ss)
forsignal(si):bool = EXISTS fsi: (fsi=si)
stoppbock(ss):bool = EXISTS sb: (sb=ss)
sparsegment(bo):bool = EXISTS ss: (ss=bo)
stopplykta(si):bool = EXISTS sl: (sl=si)

ejlagre(x:nat,y:nat):bool = NOT (x < y)


sist(ss,vg):bool = del_av(ss,vg) AND
NOT (EXISTS ss1: EXISTS ss2: (ss1 /= ss2 AND
```

```
ansluter_till(ss,ss1) AND (del_av(ss1,vg) OR fore(ss1,vg))
AND ansluter_till(ss2,ss) AND del_av(ss2,vg)))

belopp(n:int) : nat = IF n >= 0 THEN n ELSE -n ENDIF

norm(ss, vg) : int

norm_axiom : AXIOM (FORALL (ss1, ss2, vg) :
      (EXISTS (m: nat): EXISTS (g: [below(m)->segments(vg)]):
                bijective?(g) => norm(ss1, vg)<= m)
                     AND
  NOT(fore(ss1, vg) OR del_av(ss1, vg)) => norm(ss1, vg) = -1)
                     AND
                    ((fore(ss1, vg) OR del_av(ss1, vg)) =>
                     (norm(ss1, vg) >= 0)
                      AND
                     (sist(ss1, vg) <=> norm(ss1,  vg) = 0)
                      AND
                     (fore(ss2, vg) OR del_av(ss2, vg) =>
               ((norm(ss1, vg) = norm(ss2, vg) <=> ss1 = ss2)
                      AND
        (ansluter_till(ss1, ss2) <=>
          belopp(norm(ss1, vg) - norm(ss2, vg)) = 1)))))


slutpunkt_till(hsisl,vg):bool = (EXISTS ss:
(framfor(ss,hsisl) AND sist(ss,vg))) AND (FORALL ss:
(bakom(ss,hsisl) => NOT del_av(ss,vg)))

foljande_vag(vg1,vg2):bool = (EXISTS ss: (fore(ss,vg1) AND
sist(ss,vg2))) AND NOT i_konflikt(vg1,vg2) AND NOT
(EXISTS vx: ((EXISTS ss: (hoger_gren(ss,vx) AND
del_av(ss,vg1))) AND (EXISTS ss: (vanster_gren(ss,vx) AND
del_av(ss,vg2))))) AND NOT (EXISTS vx: ((EXISTS ss:
(vanster_gren(ss,vx) AND del_av(ss,vg2))) AND
(EXISTS ss: (hoger_gren(ss,vx) AND del_av(ss,vg1)))))

mojlig_skyddsstracka_till(vg,tv):bool =
NOT ej_mojlig_skyddsstracka(vg) AND foljande_vag(vg,tv)

nasta_hsi(fsi,hsi):bool = (EXISTS vg: (lagd(vg) AND
medriktad(fsi,vg) AND slutpunkt_till(hsi,vg))) AND
(FORALL vg: (lagd(vg) AND medriktad(fsi,vg) AND
(EXISTS hsi1: (hsi /= hsi1 AND slutpunkt_till(hsi1,vg))) =>
medriktad(hsi,vg))) OR (EXISTS ss: (bakom(ss,fsi) AND
linje(ss) AND framfor(ss,hsi))) AND NOT (EXISTS ss:
```

```
framfor(ss,fsi))

samplacerade(si1,si2):bool = (EXISTS ss: (framfor(ss,si1)
AND framfor(ss,si2))) AND EXISTS ss: (bakom(ss,si1) AND
bakom(ss,si2)))

delvag(vg1,vg2):bool = (EXISTS ss: (fore(ss,vg1) AND
fore(ss,vg2))) AND (FORALL ss: (del_av(ss,vg1) =>
del_av(ss,vg2)))

ax001 : AXIOM ((FORALL (hsi, tv1, tv2): (korbesked(tv1,hsi)
AND korbesked(tv2,hsi) => tv1=tv2)))

ax002 : AXIOM ((FORALL (fsi, hsi1, hsi2): (nasta_hsi(fsi,hsi1)
AND nasta_hsi(fsi,hsi2) => hsi1=hsi2)))

ax003 : AXIOM ((FORALL (si, ss1, ss2): (framfor(ss1,si) AND
framfor(ss2,si) => ss1=ss2)))

ax004 : AXIOM ((FORALL (si, ss1, ss2): (bakom(ss1,si) AND
 bakom(ss2,si) => ss1=ss2)))

ax0041 : AXIOM (FORALL (ss, si) : NOT(bakom(ss, si) AND
framfor(ss, si)))

ax005 : AXIOM ((FORALL ss: EXISTS ss1: (ansluter_till(ss,ss1)
AND (FORALL (ss1, ss2, ss3, ss4): (ansluter_till(ss,ss1) AND
ansluter_till(ss,ss2) AND ansluter_till(ss,ss3) AND
ansluter_till(ss,ss4) => ss1=ss2 OR ss1=ss3 OR ss1=ss4 OR
ss2=ss3 OR ss2=ss4 OR ss3=ss4)))))

ax006 : AXIOM ((FORALL vg: ((EXISTS ss: fore(ss,vg)) AND
(FORALL (ss1, ss2): (fore(ss1,vg) AND fore(ss2,vg) => ss1=ss2)
))))

ax007 : AXIOM ((FORALL vg: ((EXISTS ss: sist(ss,vg)) AND
(FORALL (ss1, ss2): (sist(ss1,vg) AND sist(ss2,vg) => ss1=ss2)
))))

ax008 : AXIOM ((FORALL vx: ((EXISTS ss: vanster_gren(ss,vx))
AND (FORALL (ss1, ss2): (vanster_gren(ss1,vx) AND
vanster_gren(ss2,vx) => ss1=ss2)))))

ax009 : AXIOM ((FORALL vx: ((EXISTS ss: hoger_gren(ss,vx))
AND (FORALL (ss1, ss2): (hoger_gren(ss1,vx) AND
hoger_gren(ss2,vx) => ss1=ss2)))))
```

```
ax010 : AXIOM ((FORALL om: EXISTS ss: del_av(ss,om)))

ax011 : AXIOM ((FORALL fsi: ((EXISTS v: atc_forbesked(fsi,v))
AND (FORALL (v1, v2): (atc_forbesked(fsi,v1) AND
atc_forbesked(fsi,v2) => v1=v2)))))

ax012 : AXIOM ((FORALL hsi: ((EXISTS v: atc_huvudbesked(hsi,v)
) AND (FORALL (v1, v2): (atc_huvudbesked(hsi,v1) AND
atc_huvudbesked(hsi,v2) => v1=v2)))))

ax013 : AXIOM ((FORALL tv: ((EXISTS v: sth(tv,v)) AND
(FORALL (v1, v2): (sth(tv,v1) AND sth(tv,v2) => v1=v2)))))

ax014 : AXIOM (FORALL vx: NOT (hogerlage(vx) <=>
vansterlage(vx)))

ax101 : AXIOM ((FORALL (ss1, ss2): (ansluter_till(ss1,ss2)
 => ansluter_till(ss2,ss1))))

ax1011 : AXIOM (del_av(ss, tv ) => NOT linje(ss))

ax102 : AXIOM ((FORALL ss: NOT ansluter_till(ss,ss)))

ax103 : AXIOM ((FORALL (vx, ss): (hoger_gren(ss,vx) =>
ansluter_till(vx,ss))))

ax104 : AXIOM ((FORALL (vx, ss): (vanster_gren(ss,vx) =>
ansluter_till(vx,ss))))

ax105 : AXIOM ((FORALL (vx, ss): NOT (vanster_gren(ss,vx)
AND hoger_gren(ss,vx))))

ax106 : AXIOM ((FORALL ss: (NOT vaxel(ss) =>
NOT (EXISTS (ss1, ss2, ss3): (ansluter_till(ss,ss1) AND
ansluter_till(ss,ss2) AND ansluter_till(ss,ss3) AND ss1 /= ss2
AND ss2 /= ss3 AND ss1 /= ss3)))))

ax107 : AXIOM ((FORALL ss: (andsegment(ss) <=>
NOT (EXISTS (ss1, ss2): (ansluter_till(ss,ss1) AND
ansluter_till(ss,ss2) AND ss1 /= ss2)))))

ax201 : AXIOM ((FORALL (om1, om2): (NOT om1 = om2 AND
(EXISTS ss: (del_av(ss,om1) AND del_av(ss,om2))) =>
i_konflikt(om1,om2))))
```

```
ax202 : AXIOM ((FORALL (bo, vg): NOT (medriktad(bo,vg) AND
motriktad(bo,vg))))

ax203 : AXIOM ((FORALL (vx, vg): (medriktad(vx,vg) =>
del_av(vx,vg))))

ax204 : AXIOM ((FORALL (vx, vg): (motriktad(vx,vg) =>
del_av(vx,vg))))

ax205 : AXIOM ((FORALL (vg, ss1, ss2): ((del_av(ss1,vg) OR
fore(ss1, vg)) AND (del_av(ss2,vg) OR fore(ss2, vg)) =>
NOT (EXISTS vx: (hoger_gren(ss1,vx) AND vanster_gren(ss2,vx)))
)))

ax206 : AXIOM ((FORALL (ss, vg): (fore(ss,vg) =>
NOT del_av(ss,vg) AND
(EXISTS ss1: (del_av(ss1,vg) AND ansluter_till(ss,ss1))))))

ax207 : AXIOM ((FORALL vx: FORALL vg: (sist(vx,vg) AND
sist(vx,vg) => (EXISTS ss: ((vanster_gren(ss,vx) OR
hoger_gren(ss,vx)) AND (del_av(ss,vg) OR fore(ss,vg)))))))

ax209 : AXIOM (lagd(vg) => (FORALL vx: (del_av(vx,vg) =>
NOT ur_kontroll(vx) AND ((EXISTS ss: (vanster_gren(ss,vx) AND
(del_av(ss,vg) OR fore(ss,vg)))) => vansterlage(vx)) AND
((EXISTS ss: (hoger_gren(ss,vx) AND (del_av(ss,vg)
OR fore(ss,vg)))) => hogerlage(vx)))))

ax210 : AXIOM (hinderfri(vg) => (FORALL ss: (del_av(ss,vg) =>
NOT belagd(ss))))

ax301 : AXIOM ((EXISTS tv: (last(tv) AND del_av(vx,tv))) =>
vaxellast(vx))

ax302 : AXIOM (konfliktfri(tv) => NOT (EXISTS tv1: (last(tv1)
AND i_konflikt(tv,tv1))))

ax303 : AXIOM (klar(tv) => hinderfri(tv) AND lagd(tv) AND NOT
tidsutlosning(tv))

ax305 : AXIOM (tagvagsutlosning(ss,tv) => last(tv) AND
del_av(ss,tv) AND NOT slutdel(ss,tv))

ax306 : AXIOM (tagvagsutlosning(ss,tv) => (FORALL ss1:
(fore(ss1,tv) => NOT belagd(ss1))) AND (EXISTS ss1:
(slutdel(ss1,tv) => belagd(ss1))) AND (FORALL ss1:
```

```
(del_av(ss1,tv) => NOT belagd(ss1) OR ss = ss1 OR
slutdel(ss1,tv))))

ax401 : AXIOM ((FORALL (si, vg): (medriktad(si,vg) =>
(EXISTS (ss1, ss2): bakom(ss1,si) AND del_av(ss1,vg) AND
framfor(ss2,si) AND (del_av(ss2,vg) OR fore(ss2, vg)) AND
norm(ss2, vg) = norm(ss1, vg) + 1))))


ax402 : AXIOM ((FORALL (si, vg): (motriktad(si,vg) =>
(EXISTS (ss1, ss2): bakom(ss1,si) AND del_av(ss1,vg) AND
framfor(ss2,si) AND NOT fore(ss2,vg) AND
norm(ss1, vg) = norm(ss2, vg) + 1))))

ax403 : AXIOM ((FORALL si: ((EXISTS ss1: (framfor(ss1,si) AND
((EXISTS ss2: (bakom(ss2,si) AND ansluter_till(ss1,ss2))) OR
(stopplykta(si) AND andsegment(ss1))))) OR forsignal(si) AND
(EXISTS ss: (bakom(ss,si) AND linje(ss))) AND NOT
(EXISTS ss: framfor(ss,si)))))

ax405 : AXIOM (NOT (EXISTS (hsi1, hsi2): (NOT hsi1 = hsi2 AND
samplacerade(hsi1,hsi2))))

ax406 : AXIOM (NOT (EXISTS (fsi1, fsi2): (NOT fsi1 = fsi2 AND
samplacerade(fsi1,fsi2))))

ax407 : AXIOM ((FORALL tv: EXISTS hsi: ((EXISTS ss:
(fore(ss,tv) AND framfor(ss,hsi))) AND (EXISTS ss:
(bakom(ss,hsi) AND del_av(ss,tv))))))

ax408 : AXIOM ((FORALL (tv, hsi): (medriktad(hsi,tv) =>
(EXISTS ss: (framfor(ss,hsi) AND fore(ss,tv))))))

ax410 : AXIOM ((FORALL tv: ((EXISTS hsisl:
slutpunkt_till(hsisl,tv)) OR (EXISTS ss1: (sist(ss1,tv) AND
(EXISTS ss2: (ansluter_till(ss1,ss2) AND linje(ss2))))))))

ax411 : AXIOM (korbesked(tv,hsi) => last(tv) AND klar(tv) AND
medriktad(hsi,tv))

ax412 : AXIOM (kor1(hsi) => (EXISTS tv: (korbesked(tv,hsi) AND
NOT varsamhet(tv) AND NOT kort(tv))) OR (EXISTS ss:
(bakom(ss,hsi) AND linje(ss))))

ax413 : AXIOM (kor2(hsi) => (EXISTS tv: (korbesked(tv,hsi) AND
(NOT kort(tv) OR (EXISTS hsi1: ((EXISTS ss: (sist(ss,tv) AND
```

```
framfor(ss,hsi1))) AND NOT stopp(hsi1)))))))

ax414 : AXIOM (kor3(hsi) => (EXISTS tv: korbesked(tv,hsi)))

ax415 : AXIOM (kor2(hsi) => NOT kor1(hsi))

ax416 : AXIOM (kor3(hsi) => NOT kor2(hsi) AND NOT kor1(hsi))

ax417 : AXIOM (stopp(hsi) <=> NOT (kor1(hsi) OR kor2(hsi) OR
kor3(hsi)))

ax418 : AXIOM (nodstopp => stopp(hsi))

ax419 : AXIOM (ur_kontroll(hsi) => stopp(hsi))

ax421 : AXIOM (vanta_stopp(fsi) => NOT (vanta_kor1(fsi) OR
vanta_kor2(fsi)))

ax422 : AXIOM (NOT (vanta_kor1(fsi) OR vanta_kor2(fsi)) AND
NOT (EXISTS hsi: (samplacerade(hsi,fsi) AND NOT kor1(hsi))) =>
vanta_stopp(fsi))

ax423 : AXIOM (vanta_kor1(fsi) =>
(EXISTS hsi: (nasta_hsi(fsi,hsi) AND kor1(hsi))))

ax424 : AXIOM (vanta_kor2(fsi) =>
(EXISTS hsi: (nasta_hsi(fsi,hsi) AND NOT stopp(hsi))) AND
NOT vanta_kor1(fsi))

ax425 : AXIOM (ur_kontroll(fsi) => NOT (vanta_kor1(fsi) OR
vanta_kor2(fsi)))

ax426 : AXIOM ((EXISTS tv: (last(tv) AND (EXISTS ss:
(bakom(ss,sl) AND del_av(ss,tv))) AND (FORALL ss:
(framfor(ss,sl) => NOT del_av(ss,tv) AND NOT
fore(ss,tv))))) => stopp(sl))

ax501 : AXIOM (atc_huvudbesked(hsi,v) =>
(EXISTS tv: (korbesked(tv,hsi) AND (EXISTS v1: (sth(tv,v1) AND
ejlagre(v1,v))))) OR (EXISTS ss: (bakom(ss,hsi) AND
linje(ss))) OR v = 0)

ax502 : AXIOM (atc_huvudbesked(hsi,0) => stopp(hsi))

ax503 : AXIOM (kor1(hsi) =>
(EXISTS v1: (atc_huvudbesked(hsi,v1) AND ejlagre(v1,80))))
```

```
ax504 : AXIOM (atc_forbesked(fsi,v) => (EXISTS hsi:
(nasta_hsi(fsi,hsi) AND (EXISTS v1: (atc_huvudbesked(hsi,v1)
AND ejlagre(v1,v))))) OR v = 0)

ax601 : AXIOM (i_konflikt(vg,tv) AND NOT (EXISTS ss:
(fore(ss,vg) AND fore(ss,tv)))
=> skyddsstracka_konflikt(vg,tv))

ax602 : AXIOM (konfliktfri(tv) => NOT (EXISTS tv1:
(slutpunktlast(tv1) AND (EXISTS vg:
(skyddsstracka_till(vg,tv1) AND skyddsstracka_konflikt(vg,tv))
))))

ax603 : AXIOM (konfliktfri(tv) => (FORALL vg:
(skyddsstracka_till(vg,tv) => (FORALL tv1: (last(tv1) => NOT
skyddsstracka_konflikt(vg,tv1))))))

ax604 : AXIOM (klar(tv) => slutpunktsskyddad(tv))

ax606 : AXIOM (skyddsstracka_lagd(vg) => (FORALL vx:
(motriktad(vx,vg) => ((EXISTS ss: (vanster_gren(ss,vx) AND
(del_av(ss,vg) OR fore(ss,vg)))) => vansterlage(vx)) AND
((EXISTS ss: (hoger_gren(ss,vx) AND (del_av(ss,vg) OR
fore(ss,vg)))) => hogerlage(vx)))))

ax607 : AXIOM (skyddad_bakat(vg) => (EXISTS ss: (sist(ss,vg)
AND (stoppbock(ss) OR (EXISTS si: (bakom(ss,si) AND
motriktad(si,vg)))))) OR
(EXISTS vx: slutskyddande_vaxel(vg,vx)))

ax608 : AXIOM (slutskyddande_vaxel(vg,vx) => (EXISTS ss:
(sist(ss,vg) AND ansluter_till(vx,ss) AND NOT del_av(vx,vg)
AND (vaxel(ss) AND medriktad(ss,vg) =>
NOT (vanster_gren(vx,ss) OR hoger_gren(vx,ss))) AND
NOT ur_kontroll(vx) AND (vanster_gren(ss,vx) AND hogerlage(vx)
OR hoger_gren(ss,vx) AND vansterlage(vx)))))

ax609 : AXIOM (skyddsstracka_till(vg,tv) =>
mojlig_skyddsstracka_till(vg,tv) AND skyddsstracka_lagd(vg)
AND hinderfri(vg) AND skyddad_bakat(vg) AND NOT
(EXISTS vg1: (vg /= vg1 AND mojlig_skyddsstracka_till(vg1,tv)
AND delvag(vg1,vg) AND skyddad_bakat(vg1))))

ax610 : AXIOM (slutpunktsskyddad(tv) =>
aldrig_skyddsstracka(tv) OR (EXISTS vg:
```

```
skyddsstracka_till(vg,tv)) OR (EXISTS tv1:
(foljande_vag(tv1,tv) AND last(tv1) AND hinderfri(tv1))))

ax611 : AXIOM (aldrig_skyddsstracka(tv) => (EXISTS ss:
(sist(ss,tv) AND (stoppbock(ss) OR (EXISTS ss1:
(ansluter_till(ss,ss1) AND linje(ss1)))))))

ax612 : AXIOM (slutpunktsutlosning(ss,tv) => (NOT last(tv) OR
tagvagsutlosning(ss,tv)) AND sist(ss,tv) AND belagd(ss))

ax613 : AXIOM ((EXISTS tv: (slutpunktlast(tv) AND (EXISTS ss:
(framfor(ss,sl) AND del_av(ss,tv))) AND (FORALL ss:
(bakom(ss,sl) => NOT del_av(ss,tv) AND NOT fore(ss,tv))))) =>
stopp(sl))

ax614 : AXIOM ( (EXISTS tv: (slutpunktlast(tv) AND (EXISTS vg:
(skyddsstracka_till(vg,tv) AND slutskyddande_vaxel(vg,vx)))))
=> vaxellast(vx))

ax615 : AXIOM ( (EXISTS tv: (slutpunktlast(tv) AND (EXISTS vg:
(motriktad(vx,vg))))) => vaxellast(vx))

ax701 : AXIOM ((FORALL (ss, tv): (skyddsobjekt_till(ss,tv) AND
sparsegment(ss) => vaxel(ss))))

ax702 : AXIOM ( (EXISTS tv: (last(tv) AND
skyddsobjekt_till(vx,tv))) => vaxellast(vx))

ax703 : AXIOM (sidoskyddad(tv) => (FORALL ss:
(sida_till(ss,tv) => NOT belagd(ss) AND NOT (EXISTS tv1:
(last(tv1) AND del_av(ss,tv1))))))

ax704 : AXIOM ((EXISTS tv: (skyddsobjekt_till(sl,tv) AND
last(tv))) => stopp(sl))

ax705 : AXIOM (sidoskyddad(tv) => (FORALL sl:
(skyddsobjekt_till(sl,tv) => NOT ur_kontroll(sl))) AND
(FORALL vx: (skyddsobjekt_till(vx,tv) => NOT ur_kontroll(vx)))
)

ax706 : AXIOM (sidoskyddad(tv) => (FORALL vx:
(skyddsobjekt_till(vx,tv) => ((EXISTS ss: (vanster_gren(ss,vx)
AND (del_av(ss,tv) OR sida_till(ss,tv)))) =>
hogerlage(vx)) AND ((EXISTS ss: (hoger_gren(ss,vx) AND
(del_av(ss,tv) OR sida_till(ss,tv)))) => vansterlage(vx)))))
```

```
ax707 : AXIOM (klar(tv) => sidoskyddad(tv))

ax801 : AXIOM (lokalfrigiven(vx) => vaxellast(vx))

ax802 : AXIOM (frigivningsbar(vx) => NOT vaxellast(vx) OR
lokalfrigiven(vx))

ax803 : AXIOM (klar(tv) => (FORALL vx: (del_av(vx,tv) => NOT
lokalfrigiven(vx))))

ax804 : AXIOM (skyddsstracka_till(vg,tv) => (FORALL vx:
(motriktad(vx,vg) => NOT lokalfrigiven(vx))))

ax805 : AXIOM (slutskyddande_vaxel(vg,vx) => (NOT
lokalfrigiven(vx)))

ax806 : AXIOM (sidoskyddad(tv) => (FORALL vx:
(skyddsobjekt_till(vx,tv) => NOT lokalfrigiven(vx))))

ax901 : AXIOM ((FORALL (tv1, tv2): (huvudtagvag(tv1) AND
huvudtagvag(tv2) AND tv1 /= tv2 => (FORALL hsi: NOT
(medriktad(hsi,tv1) AND medriktad(hsi,tv2))))))

ax902 : AXIOM (kan_bli_obevakad => NOT obevakad AND
(FORALL tv: (last(tv) OR slutpunktlast(tv) =>
huvudtagvag(tv))))

ax903 : AXIOM (kan_bli_bevakad => obevakad AND (FORALL tv:
(last(tv) OR slutpunktlast(tv) => konfliktfri(tv))))

ax904 : AXIOM (tagvagsutlosning(ss,tv) => NOT obevakad)

ax905 : AXIOM (slutpunktsutlosning(ss,tv) =>  NOT obevakad)

ax906 : AXIOM (klar(tv) => (konfliktfri(tv) OR huvudtagvag(tv)
AND obevakad))



thm1 : THEOREM (FORALL hsi: (NOT stopp(hsi) AND NOT
(EXISTS ss: (linje(ss) AND bakom(ss,hsi))) => EXISTS tv:
(last(tv) AND medriktad(hsi,tv))))

lemma1 : LEMMA (medriktad(hsi1, tv) AND medriktad(hsi2, tv) =>
hsi1 = hsi2)
```

```
lemma2 : LEMMA (medriktad(si, vg1) AND medriktad(si, vg2) AND
NOT vg1 = vg2  => i_konflikt(vg1, vg2))

lemma3 : LEMMA (medriktad(hsi, tv1) AND motriktad(hsi, tv2) =>
i_konflikt(tv2, tv1))

lemma4 : LEMMA (NOT (medriktad(hsi, tv) AND EXISTS ss:
(bakom(ss, hsi) AND linje(ss))))

lemma5 : LEMMA (NOT (motriktad(hsi, tv) AND EXISTS ss:
(bakom(ss, hsi) AND linje(ss))))


thm2 : THEOREM  (NOT obevakad => FORALL tv: (klar(tv) =>
NOT EXISTS (hsi1, hsi2): (medriktad(hsi1,tv) AND
motriktad(hsi2,tv) AND NOT stopp(hsi1) AND NOT stopp(hsi2))))

thm3 : THEOREM (FORALL (fsi, hsi): (nasta_hsi(fsi,hsi) =>
FORALL (v1, v2): (atc_forbesked(fsi,v1) AND
atc_huvudbesked(hsi,v2) => ejlagre(v2,v1))))

END banverket
```