Electronic Notes in Theoretical Computer Science 174 (2007) 39–54 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Applications of Hierarchical Reasoning in the Verification of Complex Systems[3](#_bookmark1)

# Swen Jacobs[1](#_bookmark1) Viorica Sofronie-Stokkermans[2](#_bookmark0)

*Max-Planck-Institut fu¨r Informatik, Stuhlsatzenhausweg 85, Saarbru¨cken, Germany*

Abstract

In this paper we show how hierarchical reasoning can be used to verify properties of complex systems. Chains of local theory extensions are used to model a case study taken from the European Train Control System (ETCS) standard, but considerably simplified. We show how testing invariants and bounded model checking (for safety properties expressed by universally quantified formulae, depending on certain parameters of the systems) can automatically be reduced to checking satisfiability of ground formulae over a base theory.

*Keywords:* Combinations of decision procedures, Hierarchical reasoning, Verification

# Introduction

Many problems in computer science can be reduced to proving satisfiability of con- junctions of (ground) literals modulo a background theory. This theory can be a standard theory, the extension of a base theory with additional functions (free or subject to additional conditions), or a combination of theories. In [[8](#_bookmark22)] we showed that for special types of theory extensions, which we called *local*, hierarchic reasoning in which a theorem prover for the base theory is used as a “black box” is possible. Many theories important for computer science are local extensions of a base the- ory. Several examples (including theories of data structures, e.g. theories of lists (or arrays cf. [[4](#_bookmark17)]); but also theories of monotone functions or of functions satisfying semi-Galois conditions) are given in [[8](#_bookmark22),[9](#_bookmark23)]. Here we present additional examples of local theory extensions occurring in the verification of complex systems.

1 Email: [sjacobs@mpi-sb.mpg.de](mailto:sjacobs@mpi-sb.mpg.de)

2 Email: [sofronie@mpi-sb.mpg.de](mailto:sofronie@mpi-sb.mpg.de)

3 This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS). See [www.avacs.org](http://www.avacs.org/) for more information.

1571-0661 © 2007 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2006.11.038

In this paper we address a case study taken from the specification of the European Train Control System (ETCS) standard (cf. [[3](#_bookmark18)]) but considerably simpli- fied, namely an example of a communication device responsible for a given segment of the rail track, where trains may enter and leave. We suppose that, at fixed mo- ments in time, all knowledge about the current positions of the trains is available to a controller which accordingly imposes constraints on the speed of some trains, or allows them to move freely within the allowed speed range on the track. Related problems were tackled before with methods from verification [[3](#_bookmark18)].

The approach we use in this paper is different from previously used methods. We use sorted arrays (or monotonely decreasing functions) for storing the train positions. The use of abstract data structures allows us to pass in an elegant way from verification of several finite instances of problems (modeled by finite- state systems) to general verification results, in which sets of states are represented using formulae in first-order logic, by keeping the number of trains as a parameter. We show that for invariant or bounded model checking the specific properties of “position updates” can be expressed in a natural way by using chains of local theory extensions. Therefore we can use results in hierarchic theorem proving both for invariant and for bounded model checking [4](#_bookmark3) . By using locality of theory extensions we also obtained formal arguments on possibilities of systematic slicing (for bounded model checking): we show that for proving (disproving) the violation of the safety condition we only need to consider those trains which are in a neighborhood of the trains which violate the safety condition [5](#_bookmark4) .

*Structure of the paper.* Section [2](#_bookmark2) contains the main theoretical results needed in the paper. In Section [3](#_bookmark6) we describe the case study we consider. In Section [4](#_bookmark8) we present a method for invariant and bounded model checking based on hierarchical reasoning. Section [5](#_bookmark14) contains conclusions and perspectives.

# Preliminaries

Theories and models. Theories can be regarded as sets of formulae or as sets of models. Let T be a theory in a (many-sorted) signature Π = (*S,* Σ*,* Pred), where *S* is a set of sorts, Σ is a set of function symbols and Pred a set of predicate symbols (with given arities). A Π-structure is a tuple

M = ({*Ms*}*s*∈*S,* {*f*M}*f*∈Σ*,* {*P*M}*P* ∈Pred)*,*

where for every *s* ∈ *S*, *Ms* is a non-empty set, for all *f* ∈ Σ with arity

*a*(*f* )=*s*1×*.. .*×*sn*→*s*, *f*M : *n Ms* →*Ms* and for all *P* ∈ Pred with arity *a*(*P* ) =

*i*=1

i

*s*1×*.. .*×*sn, P*M ⊆ *Ms*1 × *...* ×*Ms*n . We consider formulae over variables in a (many-

sorted) family *X* = {*Xs* | *s* ∈ *S*}, where for every *s* ∈ *S*, *Xs* is a set of variables of sort *s*. A model of T is a Π-structure satisfying all formulae of T . In this paper,

4 Here we only focus on one example. However, we also used this technique for other case studies (among which one is mentioned – in a slightly different context – in [[9](#_bookmark23)]).

5 In fact, it turns out that slicing (locality) results with a similar flavor presented by Necula and McPeak in [[6](#_bookmark20)] have a similar theoretical justification.

whenever we speak about a theory T we implicitly refer to the set Mod(T ) of all models of T , if not otherwise specified.

Partial structures. Let T0 be a theory with signature Π0 = (*S*0*,* Σ0*,* Pred). We consider extensions T1 of T0 with signature Π = (*S,* Σ*,* Pred), where *S* = *S*0 ∪*S*1*,* Σ= Σ0 ∪ Σ1 (i.e. the signature is extended by new sorts and function symbols) and T1 is obtained from T0 by adding a set K of (universally quantified) clauses. Thus, Mod(T1) consists of all Π-structures which are models of K and whose reduct to Π0 is a model of T0.

A *partial* Π*-structure* is a structure M = ({*Ms*}*s*∈*S,* {*f*M}*f*∈Σ*,* {*P*M}*P* ∈Pred), where for every *s* ∈ *S*, *Ms* is a non-empty set and for every *f* ∈ Σ with arity *s*1×*.. .*×*sn*→*s*,

*f*M is a partial function from *n Ms* to *Ms*. The notion of evaluating a term *t*

*i*=1

i

with variables *X* = {*Xs* | *s* ∈ *S*} w.r.t. an assignment {*βs*:*Xs* → *Ms* | *s* ∈ *S*} for

its variables in a partial structure M is the same as for total many-sorted algebras, except that the evaluation is undefined if *t* = *f* (*t*1*,... , tn*) with *a*(*f* )=*s*1×*.. .*×*sn*→*s*, and at least one of *βs*i (*ti*) is undefined, or else (*βs*1 (*t*1)*,... , βs*n (*tn*)) is not in the domain of *f*M. In what follows we will denote a many-sorted variable assignment

{*βs*:*Xs* → *Ms* | *s* ∈ *S*} as *β* : *X* → M. Let M be a partial Π-structure, *C* a clause

and *β* : *X* → M. We say that (M*, β*) |=*w C* iff either (i) for some term *t* in *C*, *β*(*t*) is undefined, or else (ii) *β*(*t*) is defined for all terms *t* of *C*, and there exists a literal *L* in *C* s.t. *β*(*L*) is true in M. M *weakly satisﬁes C* (notation: M |=*w C*) if (M*, β*) |=*w C* for all *β* : *X* → M. M *is a weak partial model of a set of clauses* K (notation: M |=*w* K, M is a w.p.model of K) if M |=*w C* for all *C* ∈ K.

Local theory extensions. Let K be a set of (universally quantified) clauses in the signature Π = (*S,* Σ*,* Pred), where *S* = *S*0 ∪ *S*1 and Σ = Σ0 ∪ Σ1. In what follows, when referring to sets *G* of ground clauses we assume they are in the signature Π*c* = (*S,* Σ ∪ Σ*c,* Pred) where Σ*c* is a set of new constants. An extension T0 ⊆ T0 ∪K is *local* if satisfiability of a set *G* of clauses with respect to T0 ∪K only depends on T0 and those instances K[*G*] of K in which the terms starting with extension functions are in the set st(K*, G*) of ground terms which already occur in *G* or K. Formally,

K[*G*]= {*Cσ* |*C* ∈ K*,* for each subterm *f* (*t*) of *C,* with *f* ∈ Σ1*,*

*f* (*t*)*σ* ∈ st(K*, G*)*,* and for each variable *x* which does not occur below a function symbol in Σ1*, σ*(*x*)= *x*}*,*

and T0 ⊆ T1=T0 ∪K is a local extension if it satisfies condition (Loc):

(Loc) For every set *G* of ground clauses *G* |=T1 ⊥ iff there is no partial Π*c*-structure *P* such that *P*|Π0 is a total model of T0, all terms in st(K*, G*) are defined in *P* , and *P* weakly satisfies K[*G*] ∧ *G*.

In [[8](#_bookmark22),[9](#_bookmark23)] we gave several examples of local theory extensions: e.g. any extension of a theory with free function symbols; extensions with selector functions for a con- structor which is injective in the base theory; extensions of several partially ordered

theories with monotone functions. In Section [4.2](#_bookmark9) we give additional examples which have particular relevance in verification.

Hierarchic reasoning in local theory extensions. Let T0 ⊆ T1=T0 ∪K be a local theory extension. To check the satisfiability of a set *G* of ground clauses w.r.t. T1 we can proceed as follows (for details cf. [[8](#_bookmark22)]):

*Step 1: Use locality.* By the locality condition, *G* is unsatisfiable w.r.t. T1 iff K[*G*]∧*G* has no weak partial model in which all the subterms of K[*G*] ∧ *G* are defined, and whose restriction to Π0 is a total model of T0.

*Step 2: Flattening and puriﬁcation.* We purify and flatten K[*G*] ∧ *G* by introdu- cing new constants for the arguments of the extension functions as well as for the (sub)terms *t* = *f* (*g*1*,... , gn*) starting with extension functions *f* ∈ Σ1, together with new corresponding definitions *ct* ≈ *t*. The set of clauses thus obtained has the form K0 ∧ *G*0 ∧ *D*, where *D* is a set of ground unit clauses of the form *f* (*c*1*,... , cn*) ≈ *c*, where *f* ∈ Σ1 and *c*1*,... , cn,c* are constants, and K0*, G*0 are clause sets without function symbols in Σ1.

*Step 3: Reduction to testing satisﬁability in* T0*.* We reduce the problem to testing satisfiability in T0 by replacing *D* with the following set of clauses:

*n*

*N*0 = { *ci* = *di* → *c* = *d* | *f* (*c*1*,... , cn*)= *c, f* (*d*1*,... , dn*)= *d* ∈ *D*}*.*

*i*=1

Theorem 2.1 ([[8](#_bookmark22)]) *With the notations above, the following are equivalent:*

1. T0 ∧K ∧ *G has a model.*
2. T0 ∧ K[*G*] ∧ *G has a w.p.model (where all terms in* st(K*, G*) *are deﬁned).*
3. T0 ∧ K0 ∧ *G*0 ∧ *D has a w.p.model (with all terms in* st(K*, G*) *deﬁned).*
4. T0 ∧ K0 ∧ *G*0 ∧ *N*0 *has a (total)* Σ0*-model.*

# The RBC Case Study

The case study we discuss here is taken from the specification of the European Train Control System (ETCS) standard [[3](#_bookmark18)]: we consider a radio block center (RBC), which communicates with all trains on a given track segment. Trains may enter and leave the area, given that a certain maximum number of trains on the track is not exceeded. Every train reports its position to the RBC in given time intervals and the RBC communicates to every train how far it can safely move, based on the position of the preceding train. It is then the responsibility of the trains to adjust their speed between given minimum and maximum speeds.

For a first try at verifying properties of this system, we have considerably sim- plified it: we abstract from the communication issues in that we always evaluate the system after a certain time Δt, and at these evaluation points the positions

of all trains are known. Depending on these positions, the possible speed of every train until the next evaluation is decided: if the distance to the preceding train is less than a certain limit lalarm, the train may only move with minimum speed min (otherwise with any speed between min and the maximum speed max).

* 1. *Formal Description of the System Model*

We present two formal system models. In the first one we have a fixed number of trains; in the second we allow for entering and leaving trains.

Model 1: Fixed Number of Trains. In this simpler model, any state of the system is characterized by the real-valued constants Δt *>*R 0 (the time between evaluations of the system) [6](#_bookmark7) , min and max (the minimum and maximum speed of trains), lalarm (the distance between trains which is deemed secure), the integer constant n (the number of trains) and the function pos (mapping integers between 0 and *n* − 1 to real values representing the position of the corresponding train).

We use an additional function pos' to model the evolution of the system: pos'(*i*) denotes the position of *i* at the next evaluation point (after Δt time units). The way positions change (i.e. the relationship between pos and pos') is defined by the following set K*f* = {F1*,* F2*,* F3*,* F4} of axioms:

(F1) ∀*i* (*i* =0 → pos(*i*)+ Δt∗min ≤R pos'(*i*) ≤R pos(*i*)+ Δt∗max)

(F2) ∀*i* (0 *< i <* n ∧ pos(*i* − 1) *>*R 0 ∧ pos(*p*(*i*)) − pos(*i*) ≥R lalarm

→ pos(*i*)+ Δt ∗ min ≤R pos'(*i*) ≤R pos(*i*)+ Δt∗max)

(F3) ∀*i* (0 *< i <* n ∧ pos(*i* − 1) *>*R 0 ∧ pos(*p*(*i*)) − pos(*i*) *<*R lalarm

→ pos'(*i*)= pos(*i*)+ Δt∗min)

(F4) ∀*i* (0 *< i <* n ∧ pos(*i* − 1) ≤R 0 → pos'(*i*)= pos(*i*))

Note that the train with number 0 is the train with the greatest position, i.e. we count trains from highest to lowest position.

Axiom F1 states that the first train may always move at any speed between min and max. F2 states that the other trains can do so if their predecessor has already started and the distance to it is larger than lalarm. If the predecessor of a train has started, but is less than lalarm away, then the train may only move at speed min (axiom F3). F4 requires that a train may not move at all if its predecessor has not started.

Model 2: Incoming and leaving trains. If we allow incoming and leaving trains, we additionally need a measure for the number of trains on the track. This is given by additional constants first and last, which at any time give the number of the first and last train on the track (again, the first train is supposed to be the train with the highest position). Furthermore, the maximum number of trains that is allowed

6 Inequality over integers is displayed without subscript, inequality over reals is marked with an R

to be on the track simultaneously is given by a constant maxTrains. These three values replace the number of trains n in the simpler model, the rest of it remains the same except that the function pos is now defined for values between first and last, where before it was defined between 0 and n− 1. The behavior of this extended system is described by the following set K*v* consisting of axioms (V1) − (V9):

(V1) ∀*i* (*i* = first → pos(*i*)+ Δt ∗ min ≤R pos'(*i*) ≤R pos(*i*)+ Δt ∗ max)

(V2) ∀*i* (first *< i* ≤ last ∧ pos(*i* − 1) *>*R 0 ∧ pos(*i* − 1) − pos(*i*) ≥R lalarm

→ pos(*i*)+ Δt ∗ min ≤R pos'(*i*) ≤R pos(*i*)+ Δt ∗ max)

(V3) ∀*i* (first *< i* ≤ last ∧ pos(*i* − 1) *>*R 0 ∧ pos(*i* − 1) − pos(*i*) *<*R lalarm

→ pos'(*i*)= pos(*i*)+ Δt ∗ min)

(V4) ∀*i* (first *< i* ≤ last ∧ pos(*i* − 1) ≤R 0 → pos'(*i*)= pos(*i*))

(V5) last − first +1 *<* maxTrains → last' = last ∨ last' = last +1

(V6) last − first +1 = maxTrains → last' = last

(V7) last − first +1 *>* 0 → first' = first ∨ first' = first +1

(V8) last − first +1 = 0 → first' = first

(V9) last' = last +1 → pos'(last') *<*R pos'(last)

where primed symbols denote the state of the system at the next evaluation.

Here, axioms V1 − V4 are similar to F1 − F4, except that the fixed bounds are replaced by the constants first and last. V5 states that if the number of trains is less than maxTrains, then a new train may enter or not. V6 says that no train may enter if maxTrains is already reached. V7 and V8 are similar conditions for leaving trains. Finally, V9 states that if a train enters, its position must be behind the train that was last before.

# Hierarchical reasoning in verification

The safety condition which is important for this type of systems is collision freeness. Intuitively (but in a very simplified model of the system of trains) collision freeness is similar to a bounded strict monotonicity property for the function pos which stores the positions of the trains:

Mon(pos) ∀*i, j* (0 ≤ *i < j <* n → pos(*i*) *>*R pos(*j*))

Mon(pos) expresses the condition that for all trains *i, j* on the track, if *i* precedes *j*

then *i* should be positioned strictly ahead of *j*.

We will also consider a more realistic extension, which allows to express collision- freeness when the maximum length of the trains is known. In both cases, we focus on invariant checking and on bounded model checking.

* 1. *Problems: Invariant checking, bounded model checking*

In what follows we illustrate the ideas for the simple approach, in which collision- freeness is identified with strict monotonicity of the function which stores the po- sitions of the trains. To check that strict monotonicity of train positions is an invariant, we need to check that:

1. In the initial state the train positions (expressed by a function pos0) satisfy the strict monotonicity condition Mon(pos0).
2. Assuming that at a given state, the function pos (indicating the positions) sat- isfies the strict monotonicity condition Mon(pos), and the next state positions, stored in pos', satisfy the axioms K, where K ∈ {K*f ,* K*v*}, then pos' satisfies the strict monotonicity condition Mon(pos').

Checking (a) is not a problem. For (b) we need to show that in the extension T of a combination T0 of real arithmetic (sort num) with an index theory describing precedence of trains (sort i), with the two functions pos and pos' (with arity i → num) the following holds:

T |= K∧ Mon(pos) → Mon(pos')*,* i.e. T ∧ K ∧ Mon(pos) ∧ ¬Mon(pos') |=⊥ *.*

The set of formulae to be proved unsatisfiable w.r.t. T involves the axioms K and Mon(pos), containing universally quantified variables of sort i. Only ¬Mon(pos') corresponds to a ground set of clauses *G*. However, positive results for reasoning in combinations of theories were only obtained for testing satisfiability for ground formulae [[7](#_bookmark21),[5](#_bookmark19)], so are not directly applicable.

In bounded model checking the same problem occurs. For a fixed *k*, one has to show that there are no paths of length at most *k* from the initial state to an unsafe state. We therefore need to store all intermediate positions in ar- rays pos0*,* pos1*,... ,* pos*k* , and – provided that K(pos*i*−1*,* pos*i*) is defined such that

K = K(pos*,* pos') – to show:

*j*

T ∧ K(pos*i*−1*,* pos*i*) ∧ Mon(pos0) ∧ ¬Mon(pos*j* ) |=⊥ for all 0 ≤ *j* ≤ *k.*

*i*=1

* 1. *Our solution: locality, hierarchical reasoning*

Our idea. In order to overcome the problem mentioned above we proceed as follows. We consider two successive extensions of the base theory T0 (a many-sorted combination of real or rational arithmetic – for reasoning about positions, sort num – with an index theory – for describing precedence between trains, sort i):

* the extension T1 of T0 with a monotone function pos, of arity i→num,
* the extension T2 of T1 with a function pos' (arity i→num) satisfying K∈ {K*f ,* K*v*}. We show that both extensions T0 ⊆ T1 = T0 ∪ Mon(pos) and T1 ⊆ T2 = T1 ∪K

are local, where K ∈ {K*f ,* K*v*}. This allows us to reduce problem (b) to testing

satisfiability of ground clauses in T0, for which standard methods for reasoning in

combinations of theories can be applied. A similar method can be used for bounded model checking.

The base theory. As mentioned before, we assume that T0 is the many-sorted combination of a theory T i (sort i) for reasoning about precedence between trains

num 0

and a theory T0 (sort num) for reasoning about distances between trains. We

have several possibilities of choosing T i: we can model the trains on a track by using an (acyclic) list structure, where any train is linked to its predecessor, or using the theory of integers with predecessor. T num can be the theory of real or rational numbers, or linear real or rational arithmetic.

0

0

*Notation.* As a convention, everywhere in what follows *i, j, k* denote variables of sort i and *c, d* denote variables of sort num.

Collision freeness as monotonicity. In what follows let T i

be the theory of

num 0

(linear) integer arithmetic and T0 be the theory of real or rational numbers.

In both these theories satisfiability of ground clauses is decidable. Let T0 be the

(disjoint, many-sorted) combination of T i and T num. Then classical methods on

0

0

combinations of decision procedures for (disjoint, many-sorted) theories can be used

to give a decision procedure for satisfiability of ground clauses w.r.t. T0. Let T1 be obtained by extending T0 with a function pos of arity i → num mapping train indices to the real numbers, which satisfies condition Mon(pos):

Mon(pos) ∀*i, j* (first ≤ *i < j* ≤ last → pos(*i*) *>*R pos(*j*))*,*

where *i* and *j* are indices, *<* is the ordering on indices and *>*R is the usual ordering on the real numbers. (For the case of a fixed number n of trains, we can assume that first =0 and last = n − 1.)

A more precise axiomatization of collision-freeness. The monotonicity axiom above is, in fact, an oversimplification. A more precise model, in which the length of trains is considered can be obtained by replacing the monotonicity axiom for pos with the following axiom:

∀*i, j, k* (first ≤ *j* ≤ *i* ≤ last ∧ *i* − *j* = *k* → pos(*j*) − pos(*i*) ≥R *k* ∗ LengthTrain)*,*

where LengthTrain is the standard (resp. maximal) length of a train.

As base theory we consider the combination T ' of the theory of integers and reals with a multiplication operation ∗ of arity i × num → num (multiplication of *k* with the constant LengthTrain in the formula above) [7](#_bookmark11) . Let T ' be the theory obtained by extending T ' with a function pos satisfying the axiom above.

0

1

0

Theorem 4.1 *The following extensions are local theory extensions:*

7 In the light of locality properties of such extensions (cf. Theorem [4.1](#_bookmark10)), *k* will always be instantiated by values in a finite set of *concrete* integers, all within a given, *concrete* range; thus the introduction of this many-sorted multiplication does not affect decidability.

1. *The theory extension* T0 ⊆ T1*.*
2. *The theory extension* T ' ⊆T '*.*

0 1

*Proof* : (1) The clause which states that pos is strictly decreasing

Mon(pos) ∀*i, j* (first ≤ *i < j* ≤ last → pos(*i*) *>*R pos(*j*))

is flat and linear w.r.t. pos, so we can prove the claim by showing that every weak partial model *M* of T1 in which everything except pos is totally defined can be extended to a total model of T1. Locality then follows by results in [[8](#_bookmark22)]. To define pos at positions where it is undefined we use the density of real numbers and the fact that between two integers there are only finitely many integers:

Let *M* be a weak partial model of T1. We denote by *M*i the universe of *M* of sort i (i.e. the set of integers) and by *M*num the support of *M* of sort num (i.e. the set of real numbers). Then for all *i, j* ∈ *M*i, if *i < j* and both pos(*i*) and pos(*j*) are defined then pos(*i*) *>*R pos(*j*). To extend *M* to a total model of T1, we define values for the pos(*i*) that are undefined in *M* . For every *i* ∈ *M*i we check for the smallest *i*+*>i* and the greatest *i*−*<i* with pos(*i*+)*,* pos(*i*−) defined in *M* :

* + if neither such an *i*+ nor such an *i*− exists, then pos is totally undefined. Clearly, one can choose values for all indices such that Mon(pos) is satisfied.
  + if there exists an *i*+, but not an *i*−, we can choose any value for pos(*i*) which satisfies pos(*i*) *>*R pos(*i*+).
  + if there exists an *i*−, but not an *i*+, we can choose any value for pos(*i*) which satisfies pos(*i*−) *>*R pos(*i*).
  + if both *i*+ and *i*− exist, choose pos(*i*) such that it satisfies pos(*i*−) *>*R pos(*i*) *>*R

pos(*i*+).

The procedure can be repeated until pos is defined at all points between first and last. As there are finitely many positions between these two positions, the procedure terminates after a finite number of steps. We can define pos arbitrarily outside of this range. The result is a total model of T1.

1. The proof is similar to the proof of (1). Let *M* be a weak partial model of T '. Let *M*i, *M*num as above. Then for all *i, j* ∈ *M*i, if *i* − *j* = *k >* 0 and both pos(*i*) and pos(*j*) are defined then pos(*j*) − pos(*i*) ≥R *k* ∗ LengthTrain. To extend *M* to a total model of T ', we define values for the pos(*i*) that are undefined in *M* , using *i*+ and *i*− as above:

1

1

* + if neither such an *i*+ nor such an *i*− exists, then *a* is totally undefined. Clearly, one can choose values for all indices such that the condition above is satisfied.
  + if there exists an *i*+, but not an *i*−, we can fill in all the values, starting from *i*+ by defining pos(*i*+ − 1) = pos(*i*+)+ LengthTrain, and inductively, pos(*j* − 1) = pos(*j*)+ LengthTrain for all *i*+ ≤ *j* ≤ first.
  + if there exists an *i*−, but not an *i*+, we proceed similarly.
  + if both *i*+ and *i*− exist, we know that pos(*i*−) − pos(*i*+) ≤ (*i*+ − *i*−) ∗ LengthTrain.

Starting with *j* = *i*− + 1 we define for every *i*− *< j < i*+ − 1, pos(*j*) = pos(*j* −

1) + LengthTrain.

We now extend the resulting theory T1 again in two different ways, with the axiom sets for one of the two system models, respectively. A similar construction can be done starting from the theory T '.

1

Theorem 4.2 *The following extensions are local theory extensions:*

1. *The extension* T1 ⊆ T1 ∪ K*f .*
2. *The extension* T1 ⊆ T1 ∪ K*v.*

*Proof* : The idea for both proofs is to show that weak partial models can be extended to total ones, which implies locality by the results in [[8](#_bookmark22)].

1. Clauses in K*f* are flat and linear w.r.t. pos', so we can prove locality of the extension by showing that weak partial models can be extended to total ones. Let *M* be a weak partial model of T1 ∪ K*f* , where everything but pos' is totally defined. We extend *M* by defining values for all undefined pos'(*i*):
   * if *i <* 0 or *i* ≥ *n*, pos'(*i*) can be chosen arbitrarily;
   * if 0 ≤ *i* ≤ *n* − 1, the left-hand sides of the implications (F1) to (F4) are mutually exclusive, i.e. for any possible valuation of *i* we only have to satisfy the right-hand side of one implication; the other implications are true because their antecedent is false. Let *i* with 0 ≤ *i* ≤ *n* − 1 for which pos'(*i*) is undefined in *M* :
     + if the left-hand side of (F1) or (F2) is true in *M* , choose a pos'(*i*) that satisfies

pos(*i*)+ min ≤R pos'(*i*) ≤R pos(*i*)+ max. This is possible, as min ≤R max;

* + - if the the left-hand side of (F3) is true in *M* , let pos'(*i*)=pos(*i*)+Δ*t*∗min;
    - if the the left-hand side of (F4) is true in *M* , let pos'(*i*)= pos(*i*).

From the construction it is clear that we obtain a total model of T1 ∪ K.

1. is proved similarly: As above, the axioms are flat and linear w.r.t. the function symbol pos' (and of course the constants first'*,* last'), so it is enough to show that the weak partial models of T1 ∪ K*v* can be extended to total models.

Let *M* be a partial model of T1 ∪ K*v* in which everything but pos'*,* first'*,* last' is totally defined. We extend *M* to a total model of T1 ∪ K*v* in four steps:

1. As in (1), we define values for undefined pos'(*i*) within the bounds of axioms (V1) to (V4), i.e. between first and last. For values outside of the bounds, we cannot make a statement yet, as (V9) also contains pos'.
2. if first' and/or last' are undefined, axioms (V5) to (V8) are satisfied by defining

first' = first and/or last' = last.

1. if last' = last +1 in *M* and pos'(last') is undefined, define pos'(last') such that

pos'(last') *<*R pos'(last).

1. for all pos'(*i*) that are still undefined, arbitrary values can be chosen.

We thus can extend *M* to a total model of T1 ∪ K*v*.

* + 1. *Hierarchical reasoning*

Let K ∈ {K*v,* K*f* }. By the locality of T1 ⊆ T2 = T1 ∪K and by Theorem [2.1](#_bookmark5), the following are equivalent:

1. T0 ∧ Mon(pos) ∧K∧ ¬Mon(pos') |=⊥*,*
2. T0 ∧ Mon(pos) ∧ K[*G*] ∧ *G* |=*w*⊥, where *G* = ¬Mon(pos')*,*
3. T0 ∧ Mon(pos) ∧ K0 ∧ *G*0 ∧ *N*0(pos') |=⊥*,*

where K[*G*] consists of all instances of the rules in K in which the terms starting with the function symbols pos' are ground subterms already occurring in *G* or K, K0 ∧ *G*0 is obtained from K[*G*] ∧ *G* by introducing new constants for the arguments of the extension functions as well as for the (sub)terms *t* = *f* (*g*1*,... , gn*) starting with extension functions *f* ∈ Σ1, and *N*0(pos') is the set of instances of the congru- ence axioms for pos' which correspond to the definitions for these newly introduced constants.

It is easy to see that, due to the special form of the rules in K (all free variables in any clause occur as arguments of pos' both in K*f* and in K*v*), K[*G*] (hence also K0) is a set of ground clauses. By the locality of T0 ⊆ T1 = T0 ∪ Mon(pos), the following are equivalent:

1. T0 ∧ Mon(pos) ∧ K0 ∧ *G*0 ∧ *N*0(pos') |=⊥*,*
2. T0 ∧ Mon(pos)[*G*'] ∧ *G*' |=*w*⊥, where *G*' = K0 ∧ *G*0 ∧ *N*0(pos')*,*
3. T0 ∧ Mon(pos)0 ∧ *G*' ∧ *N*0(pos) |=⊥*,*

0

where Mon(pos)[*G*'] consists of all instances of the rules in Mon(pos) in which the terms starting with the function symbol pos are ground subterms already occurring

in *G*', Mon(pos)0 ∧ *G*' is obtained from Mon(pos)[*G*'] ∧ *G*' by purification and flat-

0

tening, and *N*0(pos) corresponds to the set of instances of congruence axioms for

pos which need to be taken into account. The method is illustrated in Section [4.3](#_bookmark12).

* + 1. *Application: parametric veriﬁcation*

The method for hierarchical reasoning described above allows us to reduce the prob- lem of checking whether system properties such as collision freeness are inductive invariants to deciding satisfiability of corresponding constraints in T0.

As a side effect, after the reduction of the problem to a satisfiability problem in the base theory, one can automatically determine constraints on the parameters (e.g. Δt*,* min*,* max*, ...*) which guarantee that the property is an inductive invariant, and are sufficient for this. (This can be achieved for instance using quantifier elimination.)

* + 1. *Bounded model checking*

In the example above we restricted attention to the problem of showing that a prop- erty of train systems (collision freeness) is an inductive invariant. Similar results can be established for bounded model checking. In this case the arguments are similar, but one needs to consider chains of extensions of length 1*,* 2*,* 3*,... ,k* for a bounded *k*, corresponding to the paths from the initial state to be analyzed. An interesting

side-effect of our approach (restricting to instances which are similar to the goal) is that it provides a possibility of systematic, goal-directed slicing: for proving (dis- proving) the violation of the safety condition we only need to consider those trains which are in a neighborhood of the trains which violate the safety condition.

* 1. *Illustration*

This section contains an illustration of the verification method based on hierarchical reasoning on the case study given in Section [3](#_bookmark6).

We indicate how to apply hierarchical reasoning on the case study given in Section [3](#_bookmark6), Model 1 [8](#_bookmark13) . We follow the steps given at the end of Section [2](#_bookmark2) and show how the sets of formulas are obtained that can finally be handed to a prover of the base theory.

To check whether T1 ∪ K*f* |= ColFree(pos'), where

ColFree(pos') ∀*i* (0 ≤ *i <* n − 1 → pos'(*i*) *>*R pos'(*i* + 1))*,*

we check whether T1∪K*f* ∪*G* |= ⊥, where *G* = {0 ≤ *k <* n−1*, k*' = *k*+1*,* pos'(*k*) ≤R pos'(*k*')} is the (skolemized) negation of ColFree(pos'), flattened by introducing a new constant *k*'.

Reduction from T1 ∪K*f* to T1. This problem is reduced to a satisfiability problem over T1 as follows:

*Step 1: Use locality.* We construct the set K*f* [*G*]: There are no ground subterms with pos' at the root in K*f* , and only two ground terms with pos' in *G*, pos'(*k*) and pos'(*k*'). This means that K*f* [*G*] consists of two instances of K*f* : one with *i* instantiated to *k*, the other instantiated to *k*'. E.g., the two instances of F2 are:

(F2[G]) (0 *< k <* n ∧ pos(*k* − 1) *>*R 0 ∧ pos(*k* − 1) − pos(*k*) ≥R lalarm

→ pos(*k*)+ Δt ∗ min ≤R pos'(*k*) ≤R pos(*k*)+ Δt∗max)

(0 *< k*' *<* n ∧ pos(*k*' − 1) *>*R 0 ∧ pos(*k*' − 1) − pos(*k*') ≥R lalarm

→ pos(*k*')+ Δt ∗ min ≤R pos'(*k*') ≤R pos(*k*')+ Δt∗max)

The construction of (F1[G]), (F3[G]) and (F4[G]) is similar. In addition, we specify the known relationships between the constants of the system:

(Dom) Δt *>*R 0 ∧ 0 ≤R min ∧ min ≤R max

*Step 2: Flattening and puriﬁcation.* K*f* [*G*] ∧ *G* is already flat w.r.t. pos'. We replace all ground terms with pos' at the root with new constants: we replace pos'(*k*) by *c*1 and pos'(*k*') by *c*2. We obtain a set of definitions *D* = {pos'(*k*)= *c*1*,* pos'(*k*')= *c*2} and a set K*f*0 of clauses which do not contain occurrences of pos', consisting of (Dom) together with:

(G0) 0 ≤ *k <* n − 1 ∧ *k*' = *k* +1 ∧ *c*1 ≤R *c*2

8 We illustrate our approach for the simplest model. For a variable number of trains or the other definition of collision-freeness, the approach is the same.

(F20) (0 *< k <* n ∧ pos(*k* − 1) *>*R 0 ∧ pos(*k* − 1) − pos(*k*) ≥R lalarm

→ pos(*k*)+ Δt ∗ min ≤R *c*1 ≤R pos(*k*)+ Δt∗max)

(0 *< k*' *<* n ∧ pos(*k*' − 1) *>*R 0 ∧ pos(*k*' − 1) − pos(*k*') ≥R lalarm

→ pos(*k*')+ Δt ∗ min ≤R *c*2 ≤R pos(*k*')+ Δt∗max)

The construction can be continued similarly for F1, F3 and F4.

*Step 3: Reduction to satisﬁability in* T1*.* We add the functionality clause *N*0 =

{*k* = *k*' → *c*1 = *c*2} and obtain a satisfiability problem in T1: K*f*0 ∧ *G*0 ∧ *N*0.

Reduction from T1 to T0. To decide satisfiability of T1 ∧ K*f*0 ∧ *G*0 ∧ *N*0, we have to do another transformation w.r.t. the extension T0 ⊆ T1. The resulting set of ground clauses can directly be handed to a decision procedure for the combination of the theory of indices and the theory of reals. We flatten and purify the set K*f*0 ∧ *G*0 ∧ *N*0 of ground clauses w.r.t. pos by introducing new constants denoting *k* − 1 and *k*' − 1, together with their definitions *k*'' = *k* − 1*, k*''' = *k*' − 1; as well as constants *di* for pos(*k*)*,* pos(*k*')*,* pos(*k*'')*,* pos(*k*'''). Taking into account only the corresponding instances of the monotonicity axiom for pos we obtain a set of clauses consisting of (Dom) together with:

0

|  |  |  |  |
| --- | --- | --- | --- |
| (G' ) | *k*'' = *k* − 1 ∧ | *k*''' = *k*' − 1 |  |
| (G0) | 0 ≤ *k < n* − 1 | ∧ *k*' = *k* +1 ∧ | *c*1 ≤R *c*2 |

(GF10) *k* =0 → *d*1 + Δ*t*∗min ≤R *c*1 ≤R *d*1 + Δ*t*∗max

*k*' =0 → *d*2 + Δ*t*∗min ≤R *c*2 ≤R *d*2 + Δ*t*∗max

(GF20) 0*<k<n* ∧ *d*3*>*R0 ∧ *d*3−*d*1 ≥R lalarm → *d*1+Δt∗min ≤R *c*1 ≤R *d*1+Δt∗max

0*<k*'*<n* ∧ *d*4*>*R0 ∧ *d*4−*d*2 ≥R lalarm → *d*2+Δt∗min ≤R *c*2 ≤R *d*2+Δt∗max

(GF30) 0*<k<n* ∧ *d*3*>*R0 ∧ *d*3−*d*1 *<*R lalarm → *c*1 = *d*1+Δ*t*∗min

0*<k*'*<n* ∧ *d*4*>*R0 ∧ *d*4−*d*2 *<*R lalarm → *c*2 = *d*2+Δ*t*∗min

(GF40) (0*<k<n* ∧ *d*3 ≤R 0 → *c*1=*d*1) ∧ (0*<k*'*<n* ∧ *d*4 ≤R 0 → *c*2=*d*2)

Mon(pos)[G'] *k < k*' → *d*1 *>*R *d*2 ∧ *k*' *< k* → *d*2 *>*R *d*1 ∧ *k*'' *< k*''' → *d*3 *>*R *d*4 *k < k*'' → *d*1 *>*R *d*3 ∧ *k*' *< k*''' → *d*2 *>*R *d*4 ∧ *k*''' *< k* → *d*4 *>*R *d*1 *k < k*''' → *d*1 *>*R *d*4 ∧ *k*'' *< k* → *d*3 *>*R *d*1 ∧ *k*''' *< k*' → *d*4 *>*R *d*2 *k*' *< k*'' → *d*2 *>*R *d*3 ∧ *k*'' *< k*' → *d*3 *>*R *d*2 ∧ *k*''' *< k*'' → *d*4 *>*R *d*3

*N*0(pos') *k* = *k*' → *c*1 = *c*2

*N*0(pos) *k* = *k*' → *d*1 = *d*2 ∧ *k* = *k*'' → *d*1 = *d*3 ∧ *k*' = *k*''' → *d*2 = *d*4 *k* = *k*''' → *d*1 = *d*4 ∧ *k*' = *k*'' → *d*2 = *d*3 ∧ *k*'' = *k*''' → *d*3 = *d*4

In fact, the constraints on indices can help to further simplify the instances of monotonicity of Mon(pos)[*G*'] ∧ *N*0(pos) ∧ *N*0(pos'): *k*' *> k, k*'' *< k, k*'' *< k*'*, k*''' *< k*'*, k*''' = *k*. The set of clauses equivalent to Mon(pos)[*G*'] ∧ *N*0(pos) ∧ *N*0(pos') is given below. (Here we do these simplifications by hand; this can be done as well by

a pre-simplification program which detects obviously true relationships between the premises of these rules.) After making these simplifications we obtain the following set of (many-sorted) constraints:

*C*Definitions

*C*Indices

(sort i)

*C*Reals

(sort num)

*C*Mixed

pos'(*k*)= *c*1 ∧ pos(*k*')= *d*2 *k*' = *k* +1

pos'(*k*')= *c*2 ∧ pos(*k*'')= *d*3 *k*'' = *k* − 1

pos(*k*)= *d*1 ∧ pos(*k*''')= *d*4 *k*''' = *k*' − 1

*d*1 *>*R *d*2 ∧ *d*3 *>*R *d*4 (GF10)

*d*3 *>*R *d*2 ∧ *d*4 *>*R *d*2 (GF20)

*d*3 *>*R *d*1 ∧ *d*1 = *d*4 (GF30)

0 ≤ *k, k*' *< n* − 1 *c*1 ≤R *c*2 ∧ (Dom) (GF40)

For checking the satisfiability of *C*Indices ∧ *C*Reals ∧ *C*Mixed we can use a prover for the two-sorted combination of the theory of integers and the theory of reals, possibly combined with a DPLL methodology for dealing with full clauses.

We present below an alternative method, somewhat similar to DPLL(T0), but which uses only branching on the literals containing terms of sort i, and thus reduces the verification problem to the problem of checking the satisfiability of a set of linear constraints over the reals. This idea, we think, may be used to simplify automated verification for a whole class of problems in which the axioms are guarded by simple, mutually disjoint and exhaustive premises expressed in a specific theory. Such examples occur very often in verification where several disjoint cases need to be taken into account.

*k* = 0: Due to the constraints in *C*Indices, the premises of all the other rules in GF10 −*GF* 40 become false, so all the rules except for the first rule in GF10 become trivially true. We thus only need to check the satisfiability (in the theory T num) of *d*1+Δ*t*∗min ≤R *c*1 ≤R *d*1+Δ*t*∗max ∧ *C*Reals. This is obviously satisfiable.

0

*k* /= 0: We distinguish the following possibilities:

*k <* 0: unsatisfiable because of 0 ≤ *k* in G0.

*k >* 0: We distinguish the following possibilities:

*k > n* − 1: unsatisfiable because of *k* ≤ *n* − 1 in G0.

*k* ≤ *n* − 1: We distinguish the following possibilities:

*k*' = 0: We need to check the satisfiability of the following set of constraints over the reals:

⎧ GF10 : *d*2 + Δ*t*∗min ≤R *c*2 ≤R *d*2 + Δ*t*∗max

⎪

GF20 : *d*3*>*R0 ∧ *d*3−*d*1 ≥R lalarm → *d*1+Δ*t*∗min ≤R *c*1 ≤R *d*1+Δ*t*∗max GF30 : *d*3*>*R0 ∧ *d*3−*d*1 *<*R lalarm → *c*1 = *d*1+Δ*t*∗min

⎪⎪

⎨

GF40 : *d*3 ≤R 0 → *c*1=*d*1

⎪ *C*Reals *c*1 ≤R *c*2 ∧ *d*1 *>*R *d*2 ∧ *d*3 *>*R *d*1 ∧ *d*3 *>*R *d*2 ∧ *d*4 *>*R *d*2 ∧

⎪

⎪ *d*3 *>*R *d*4 ∧ *d*1 = *d*2

⎩

By using quantifier elimination for the variables *c*1*, c*2*, d*1*, d*2*, d*3*, d*4, we can obtain a direct relationship between Δ*t,* min*,* max*, l*alarm which guarantees satisfiability. We did this by using the REDLOG system [[2](#_bookmark16)].

*k*' *<* 0: unsatisfiable because of *C*Indices.

*k*' *>* 0: We distinguish the following possibilities:

*k*' *> n* − 1: unsatisfiable because of *C*Indices.

*k*' ≤ *n*: We need to check the satisfiability of the following set of con- straints over the reals:

⎧⎪ GF20 : *d*3*>*R0 ∧ *d*3−*d*1 ≥R lalarm → *d*1+Δ*t*∗min ≤R *c*1 ≤R *d*1+Δ*t*∗max

⎪ *d*4*>*R0 ∧ *d*4−*d*2 ≥R lalarm → *d*2+Δ*t*∗min ≤R *c*2 ≤R *d*2+Δ*t*∗max

⎪ GF30 : *d*3*>*R0 ∧ *d*3−*d*1 *<*R lalarm → *c*1 = *d*1+Δ*t*∗min

⎪⎨ *d*4*>*R0 ∧ *d*4−*d*2 *<*R lalarm → *c*2 = *d*2+Δ*t*∗min

GF40 : *d*3 ≤R 0 → *c*1=*d*1

⎪

*d*4 ≤R 0 → *c*2=*d*2

⎪

⎪

*C*Reals *c*1 ≤R *c*2 ∧ *d*1 *>*R *d*2 ∧ *d*3 *>*R *d*1 ∧ *d*3 *>*R *d*2 ∧ *d*4 *>*R *d*2 ∧

*d*3 *>*R *d*4 ∧ *d*1 = *d*2

⎩

Again, by using quantifier elimination we can obtain a relationship between Δ*t,* min*,* max*, l*alarm which guarantees satisfiability.

Although the proofs above are generated by hand, the method is easy to implement. An implementation of the hierarchical method described in Section 2 is in progress.

# Conclusions

In this paper we described a case study concerning a system of trains on a rail track, where trains may enter and leave the area. An example of a safety condition for such a system (collision freeness) was considered. The problem above can be reduced to testing satisfiability of *quantiﬁed formulae* in complex theories. However, the existing results on reasoning in combinations of theories are restricted to testing satisfiability for *ground formulae*.

This paper shows that, in the example considered, we can reduce satisfiabil- ity checking of universally quantified formulae to the simpler task of satisfiability checking for ground clauses. For this, we identify corresponding chains of theory extensions T0 ⊆ T1 ⊆ ··· ⊆ T*i,* such that T*j* = T*j*−1 ∪ K*j* is a local extension of T*j*−1 by a set K*j* of (universally quantified) clauses. This allows us to reduce, for instance, testing collision freeness in theories containing arrays to represent the train positions, to checking the satisfiability of a set of sets of ground clauses over the combination of the theory of reals with a theory which expresses precedence between trains. However, the applicability of the method is far more general: the challenge is, at the moment, to recognize classes of local theories occurring in vari- ous areas of application. The method can be used for parametric verification: after the reduction of the problem to a satisfiability problem in the base theory, one can automatically determine constraints on the parameters (Δt*,* min*,* max*, ...*) which guarantee that the property is an inductive invariant. The implementation of the procedure described here is in progress; the method is clearly easy to implement. Our results also open a possibility of using abstraction-refinement deductive model checking in a whole class of applications including the examples presented here –

these aspects are not discussed in this paper, and rely on results we obtained in [[9](#_bookmark23)]. The results we present here also have theoretical implications: In one of the models we considered here, collision-freeness is expressed as a monotonicity condi- tion. Limits of decidability in reasoning about sorted arrays were explored in [[1](#_bookmark15)]. The decidability of satisfiability of ground clauses in the fragment of the theory of sorted arrays which we consider here is an easy consequence of the locality of

extensions with monotone functions.

Acknowledgement. Many thanks to Johannes Faber for sending us a case study taken from the specification of the European Train Control System which we used as a starting point for the specification considered in this paper.

# References

1. A. Bradley, Z. Manna, and H. Sipma. What’s decidable about arrays? In E. Emerson and K. Namjoshi, editors, *Verification, Model-Checking, and Abstract-Interpretation, 7th Int. Conf. (VMCAI 2006)*, LNCS 3855, pp. 427–442. Springer, 2006.
2. A. Dolzmann and T. Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, 1997.
3. J. Faber. Verifying real-time aspects of the European Train Control System. In *Proceedings of the 17th Nordic Workshop on Programming Theory*, pp. 67–70. University of Copenhagen, Denmark, 2005.
4. H. Ganzinger, V. Sofronie-Stokkermans, and U. Waldmann. Modular proof systems for partial functions with Evans equality. *Information and Computation*, 204(10): 1453-1492, 2006.
5. S. Ghilardi. Model theoretic methods in combined constraint satisfiability. *Journal of Automated* *Reasoning*, 33(3–4):221–249, 2004.
6. S. McPeak and G. Necula. Data structure specifications via local equality axioms. In K. Etessami and

S. Rajamani, editors, *Computer Aided Verification, 17th International Conference, CAV 2005*, LNCS 3576, pp. 476–490, 2005.

1. G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, 1979.
2. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In R. Nieuwenhuis, editor, *Automated deduction - CADE-20. Proceedings of the 20th International Conference on Automated Deduction*, LNCS 3632, pp. 219–234. Springer, 2005.
3. V. Sofronie-Stokkermans. Interpolation in local theory extensions. In U. Furbach and N. Shankar editors, *Automated Reasoning. Third International Joint Conference, IJCAR 2006*, LNAI 4130, pp. 235–250. Springer, 2006.