Verifying CSP-OZ-DC Specifications with Complex Data Types and Timing Parameters[*]

Johannes Faber[1], Swen Jacobs[2], and Viorica Sofronie-Stokkermans[2]

[1] Department of Computing Science, University of Oldenburg, Germany
j.faber@uni-oldenburg.de
[2] Max-Planck-Institut Informatik, Saarbrücken, Germany
{sjacobs,sofronie}@mpi-sb.mpg.de

Abstract. We extend existing verification methods for CSP-OZ-DC to reason about real-time systems with complex data types and timing parameters. We show that important properties of systems can be encoded in well-behaved logical theories in which hierarchic reasoning is possible. Thus, testing invariants and bounded model checking can be reduced to checking satisfiability of ground formulae over a simple base theory. We illustrate the ideas by means of a simplified version of a case study from the European Train Control System standard.

1 Introduction

Complex real-time systems, consisting of several components that interact, arise in a natural way in a wide range of applications. In order to verify these systems, one needs, on the one hand, to find a suitable specification language, and on the other hand, to develop efficient techniques for their verification.

In the *specification of complex systems*, one needs to take several aspects into account: control flow, data changes, and timing aspects. Motivated by this necessity, in [HO02, Hoe06] a specification language CSP-OZ-DC (COD) is defined, which combines Communicating Sequential Processes (CSP), Object-Z (OZ) and the Duration Calculus (DC). *Verification tasks* (e.g., invariant checking or bounded model checking) can usually be reduced to proof tasks in theories associated to the COD specification. These theories can be combinations of concrete theories (e.g., integer, rational or real numbers) and abstract theories (e.g., theories of functions or of data structures). Existing verification techniques for COD [HM05, MFR06] do not incorporate efficient reasoning in complex theories, which is essential to perform such verification tasks efficiently.

In this paper, we analyse both aspects mentioned above. We use COD specifications of systems, with complex data types and timing parameters, and analyse possibilities for efficient invariant checking and bounded model checking in these systems. The main contributions of the paper can be described as follows.

[*] This work was partly supported by the German Research Council (DFG) under grant SFB/TR 14 AVACS. See http://www.avacs.org for more information.

Specification: We extend existing work in which COD specifications were used [HO02, Hoe06, MFR06] in two ways:

(i) We use abstract data structures for representing and storing information about an unspecified *parametric* number of components of the systems. This allows us to pass in an elegant way from verification of several finite instances of a verification problem (for 2, 3, 4, ... components) to general verification results, in which the number of components is a parameter.

(ii) In order to refer to time constants also within the specification's data (OZ) part, we introduce timing parameters. This allows for more flexible specifications of timing constraints.

Verification: We show that, in this context, invariant checking or bounded model checking can be reduced to proving in complex theories. We analyse the theories that occur in relationship with a given COD specification, and present a sound and efficient method for hierarchic reasoning in such theories. We identify situations where the method is sound and complete (i.e., where the specific properties of systems define chains of local theory extensions).

Applications: Our running example is an extension of a case study that we considered in [JSS07] (in which we first applied hierarchic reasoning in the verification of train control systems). Here, we additionally encompass efficient handling of emergency messages and illustrate the full procedure – starting from a COD description of the case study to the verification.

Structure of the paper. We illustrate the idea of our approach by means of a case study, which will be our running example (Sect. 1.1). Section 2 introduces the specification language COD and discusses an extension with timing parameters. Section 3 presents an operational semantics of COD specifications, in terms of Phase Event Automata (PEA), and discusses some simplifications for PEA. Section 4 presents a verification method for COD specifications: the associated PEA are translated into transition constraint systems; verification is reduced to satisfiability checking in combinations of theories. We identify some theories, in which hierarchic reasoning is possible, occurring frequently in applications.

1.1 Illustration

We here give a general description of a case study inspired by the specification of the European Train Control System (ETCS) standard [ERT02]. We explain the tools we used for modelling the example and give the idea of the method for checking safety. This will be used as a running example throughout the paper. Related ETCS scenarios have been studied in [HJU05, FM06, MFR06, TZ06]. The example we consider has a less complicated control structure than those in [FM06, MFR06]. Instead, it considers an arbitrary number of trains, and hence, needs to use more realistic and sophisticated data types.

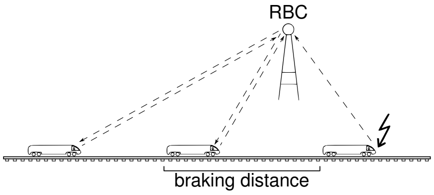


Fig. 1. Emergencies in the ETCS

The RBC Case Study. We consider a radio block centre (RBC), which communicates with all trains on a given track segment. The situation is sketched in Fig. 1. 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; the trains adjust their speed between given minimum and maximum speeds. If a train has to stop suddenly, it sends an emergency message. The RBC handles the message sent by a train (which we refer to as emergency train) by instructing each train behind the emergency train on the track to stop too.

Idea. In this case study, the following aspects need to be considered:

(1) The scenario describes processes running in parallel.
(2) We need to specify the state space and the pre- and postconditions of actions.
(3) There are timing constraints on the duration of system states.

For encompassing all these aspects, we use the specification language COD, that allows to express the control flow of the systems (expressed in CSP), data structures used for modelling state and state change (OZ) and time constraints (DC). We pass from specification to verification as follows:

- We associate so-called *Phase Event Automata* A_{CSP}, A_{OZ}, A_{DC} with the CSP, OZ and DC part, respectively. Their parallel composition A represents the semantics of the COD specification.
- From A we derive a family of transition constraints that describe the properties of the transitions in the system.
- We use this set of transition constraints for checking given safety properties.

This last verification step is highly non-trivial. Transition constraints may combine constraints over various theories. In our case, we need to reason in a combination of a theory of integers (indices of trains), reals (for modelling speeds or distances), arrays (in which the current speed and reported positions of the trains are stored), and functions (e.g., associating with each speed an upper bound for the optimal braking distance at that speed). Many of these data structures have additional properties, which need to be specified as axioms in first-order logic with quantifiers. We show that properties of systems can be encoded in well-behaved logical theories in which efficient reasoning is possible.

2 CSP-OZ-DC: A High-Level Specification Language

In order to capture the control flow, data changes, and timing aspects of the systems we want to verify, we use the high-level specification language CSP-OZ-DC (COD) [HM05, Hoe06], which integrates three well-investigated formalisms: *Communicating Sequential Processes* [Hoa85], *Object-Z* [Smi00], and *Duration Calculus* [ZH04], allowing the compositional and declarative specification of each aspect by means of the best-suited formalism. In particular, data and data changes are specified in a constraint-based representation (using OZ). In this paper, we use this advantage of the COD representation and extend the known verification procedures for COD [HM05, MFR06] to combination of theories.

We give an intuition of the formalism and its advantages using our case study: We model a *radio block centre* (RBC) that controls the railway traffic on a single (to simplify matters infinite) track segment. This RBC controls n consecutive trains, represented by their position and speed values. The full COD specification is given in Fig. 2, that we explain in the remainder of this section.

The specification begins with the declaration of a timing parameter T_PR (cf. Sect. 2.1), followed by the *interface part*, in which methods are declared. These methods are used in all parts (CSP, OZ, DC) of the COD specification, and provide the functionality of the COD class.

Interface:

`method` *positionReport*

`method` *detectEmergency* : $[\mathit{trainNumber} : \mathbb{N}]$

CSP. We use CSP [Hoa85] to specify the control flow of a system using processes over events. The interface part declares all possible events.

CSP:

$$\mathtt{main} \overset{c}{\widehat{=}} \mathit{Driveability} \mid\mid\mid \mathit{Detection}$$

$$\mathit{Driveability} \overset{c}{\widehat{=}} \mathit{positionReport} \rightarrow \mathit{Driveability}$$

$$\mathit{Detection} \overset{c}{\widehat{=}} \mathit{detectEmergency} \rightarrow \mathit{Detection}$$

The `main` process of our specification comprises an interleaving of two subprocesses, *Driveability* and *Detection*, for controlling the trains and incoming emergency messages synchronously. The *Detection* process detects emergencies using *detectEmergency* events, while the *Driveability* process regularly updates the train positions using *positionReport* events.

OZ. The data space and its changes are specified with OZ schemata [Smi00]. The OZ part of the running example begins with the state schema defining the state space of the RBC. Positions and speed values of the trains are given by sequences $\mathit{train} : \mathrm{seq}\,\mathit{Position}$ and $\mathit{speed} : \mathrm{seq}\,\mathit{Speed}$, where the types are given by reals: $\mathit{Position} == \mathbb{R}$, $\mathit{Speed} == \mathbb{R}^+$. Sequences, in the sense of OZ, are partial functions $\mathit{train} : \mathbb{N} \rightarrow \mathit{Position}$, that are defined for all $i \leq n$. A third data type is *Acceleration*, which is also real-valued: $\mathit{Acceleration} == \mathbb{R}^+$.

State space:

$\mathit{train} : \mathit{seq}\,\mathit{Position}$

$\mathit{speed} : \mathit{seq}\,\mathit{Speed}$

$\mathit{maxSpeed}, \mathit{minSpeed} : \mathit{Speed}$

$\mathit{brakingDist} : \mathit{Speed} \rightarrow \mathit{Position}$

$\mathit{emergencyTrain} : \mathbb{N}$

$\mathit{maxDec} : \mathit{Acceleration}$

$d : \mathit{Position}$

$n : \mathbb{N}$

The variable *emergencyTrain* is a pointer to the first train on the track that reported an emergency. We also define some important constants, for the maximal speed, for the minimal speed (without emergencies), the number of trains n, and the safety margin between trains d. Next follow axioms for the data structures defined in the state schema.

Axioms:

$0 < minSpeed < maxSpeed$

$n = \#train = \#speed$

$0 < d = brakingDist(maxSpeed)$

$\forall\, s : Speed \bullet brakingDist(s) \geq \frac{s^2}{2 * maxDec}$

$\forall\, s_1, s_2 : Speed \mid s_1 < s_2 \bullet brakingDist(s_1) < brakingDist(s_2)$

$brakingDist(0) = 0$

The latter three axioms ensure a safety distance between the trains. The function *brakingDist* yields for a given speed value the distance needed by a train in order to stop if the emergency brakes are applied. For the constant maximal deceleration *maxDec*, the minimal braking distance for a speed value *spd* is $\frac{spd^2}{2*maxDec}$. Since the trains can not always reach their maximal deceleration, we define this term as a lower bound for our braking function. We require monotonicity of *brakingDist* and specify its value for a speed value of 0.

Every COD class has an `Init` schema (cf. Fig. 2) that constrains initial values of state variables, and communication schemata which define state changes. Every communication schema (prefix `com`) belongs to a CSP event as given by the interface of a class. Every time a CSP event occurs the state space is changed according to the constraints of the appropriate communication schema.

`com`_*detectEmergency*

$\Delta(speed, emergencyTrain)$

$newEmergencyTrain? : \mathbb{N}$

$newEmergencyTrain? \leq n$

$emergencyTrain' = min\{newEmergencyTrain?, emergencyTrain\}$

$speed'(emergencyTrain') = 0$

$\forall\, i \in \mathbb{N} \mid i \neq emergencyTrain' \bullet speed'(i) = speed(i)$

Consider for instance the schema for *detectEmergency*. The first line identifies state variables that are changed by this schema, the remaining variables implicitly stay unchanged. The expression *newEmergencyTrain*? (second line) represents an input variable. The following lines constrain state changes (primed variables denote the post-state while unprimed variables refer to the pre-state). For example $emergencyTrain' = min\{newEmergencyTrain?, emergencyTrain\}$ sets the new value for *emergencyTrain*. (The train with the lowest number is the first on the track. So, *emergencyTrain* always points to the first train on the track that reported an emergency.) The schema `com`_*positionReport* (Fig. 2) sets the speed values for all trains and calculates their new positions: without an emergency train in front, the speed can be arbitrary between *minSpeed* and *maxSpeed*, unless the distance to the previous train is too small ($< d$); in this case the speed is set to *minSpeed*. In case of an emergency, the trains behind the emergency train brake with maximal deceleration.

DC. The duration calculus (DC) is an interval-based dense real-time logic [ZH04]. Important operators of the DC are the *chop* operator $\frown$ that splits an interval into subintervals, the operator ℓ yielding the length of an interval, and the *everywhere* operator $\lceil p \rceil$ specifying that a predicate p holds everywhere on an interval. An explicit time constant $t \in \mathbb{Q}^+$ or a symbolic constant T are used to define interval lengths. Since DC is undecidable we use a decidable sub-class (counterexample formulae). We apply the algorithm of [Hoe06] to generate automata from DC specifications of this subclass.

DC:

$\neg(\mathit{true} \frown \updownarrow \mathit{positionReport} \frown (\ell < T\_PR) \frown \updownarrow \mathit{positionReport} \frown \mathit{true})$

$\neg(\mathit{true} \frown \boxminus \mathit{positionReport} \wedge (\ell > T\_PR) \frown \mathit{true})$

In the DC specification above, the first formula specifies that it will never be the case ($\neg$) that two *positionReport* events ($\updownarrow$) are separated ($\frown$) by an interval with a length (ℓ) smaller than T_PR. (So there will be at least T_PR time units between two position reports.) In the second formula, $\boxminus$ describes an interval in which no position report event is detected. The formula states that there is no interval of a length greater than T_PR without a *positionReport* event. Together the formulae define the exact periodicity of *positionReport*.

2.1 Timing Parameters in COD

The original definition of COD in [Hoe06] only allows for using rational numbers to define interval lengths. This restriction results in a loss of generality: a developer always has to define exact values for every interval, even if the specification does not depend on an exact length. In our example, one has to replace the T_PR constant in the DC formulae with a fixed rational to get a valid COD specification. To overcome this problem, we introduce *timing parameters* as an extension for COD. That is, we allow the usage of symbolic constants for the interval definitions in the DC part, like T_PR in our example. These symbolic constants are declared as generic constants (parameters of the class) and also are accessible in the OZ part. For instance, we use T_PR in the schema of *positionReport*. That allows us to use the same (undetermined) interval length in the OZ and in the DC part.

3 Operational Semantics of COD Specifications

In this section, we present a translation from COD specifications to PEA. We extend existing translations from COD to PEA to also deal with timing parameters and study possibilities of simplifying the PEA obtained this way.

3.1 Translation of COD Specifications into PEA

Phase Event Automata (PEA) are timed automata [AD94] involving both data and timing aspects. Our definition of PEA is based on [Hoe06], but differs in that we also allow symbolic constants to occur in clock invariants $\mathcal{L}^c(C)$. In

$RBC[T\_PR : \mathbb{Q}^{+}]$

$\texttt{method}\ positionReport$
$\texttt{method}\ detectEmergency : [trainNumber : \mathbb{N}]$

Inter-face

$main \overset{c}{=} Driveability\ |||\ Detection$
$Driveability \overset{c}{=} positionReport \rightarrow Driveability$
$Detection \overset{c}{=} detectEmergency \rightarrow Detection$

CSP part

$train : \operatorname{seq} Position$ $\qquad emergencyTrain : \mathbb{N}$
$speed : \operatorname{seq} Speed$ $\qquad maxDec : Acceleration$
$maxSpeed, minSpeed : Speed$ $\qquad d : Position$
$brakingDist : Speed \rightarrow Position$ $\qquad n : \mathbb{N}$

$0 < minSpeed < maxSpeed$
$n = \#train = \#speed$
$0 < d = brakingDist(maxSpeed)$ (1)
$\forall s : Speed \bullet brakingDist(s) \geq \frac{s^{2}}{2 * maxDec}$
$\forall s_1, s_2 : Speed \mid s_1 < s_2 \bullet brakingDist(s_1) < brakingDist(s_2)$
$brakingDist(0) = 0$

$\texttt{Init}$

$emergencyTrain > n$
$\forall i : \operatorname{dom} speed \bullet minSpeed \leq speed(i) \leq maxSpeed$
$\forall i : \operatorname{dom} train \mid i \neq 1$
$\quad \bullet train(i) < train(i-1) - brakingDist(speed(i))$

$\texttt{com\_}positionReport$

$\Delta(train, speed)$

$\forall i : \operatorname{dom} train \mid i = 1 \land i < emergencyTrain$
$\quad \bullet minSpeed \leq speed'(i) \leq maxSpeed$
$\forall i : \operatorname{dom} train \mid 1 < i < emergencyTrain \land train(i-1) - train(i) \geq d$
$\quad \bullet minSpeed \leq speed'(i) \leq maxSpeed$
$\forall i : \operatorname{dom} train \mid 1 < i < emergencyTrain \land train(i-1) - train(i) < d$
$\quad \bullet minSpeed = speed'(i)$
$\forall i : \operatorname{dom} train \mid i \geq emergencyTrain$
$\quad \bullet speed'(i) = max\{speed(i) - maxDec * T\_PR, 0\}$
$\forall i : \operatorname{dom} train \bullet train'(i) = train(i) + speed'(i) * T\_PR$

$\texttt{com\_}detectEmergency$

$\Delta(speed, emergencyTrain)$
$newEmergencyTrain? : \mathbb{N}$

$newEmergencyTrain? \leq n$
$emergencyTrain' = min\{newEmergencyTrain?, emergencyTrain\}$
$speed'(emergencyTrain') = 0$
$\forall i \in \mathbb{N} \mid i \neq emergencyTrain' \bullet speed'(i) = speed(i)$

OZ part

$\neg(true \frown \updownarrow positionReport \frown (\ell < T\_PR) \frown \updownarrow positionReport \frown true)$
$\neg(true \frown \boxminus positionReport \land (\ell > T\_PR) \frown true)$

DC part

Fig. 2. The COD specification for the emergency case study

what follows, let $\mathcal{L}(V)$ be a subset of the language of OZ predicates.[1] For a set C of clock variables and timing parameters T, the set $\mathcal{L}^c(C,T)$ of (convex) *clock constraints* with constants is defined by the following BNF grammar:

$$\delta ::= c < t \mid c \leq t \mid c < z \mid c \leq z \mid \delta \wedge \delta,$$

where $c \in C$ is a clock, $t \in \mathbb{Q}^+$ is a rational constant, and $z \in T$ is a timing parameter. The semantics is given by clock valuations $\gamma : C \to \mathbb{R}^+$ assigning non-negative reals to clocks. The semantics of a timing parameter z is an interpretation $\mathcal{I} : T \to \mathbb{Q}^+$. We write $\gamma, \mathcal{I} \models \delta$ iff δ holds for γ and $\mathcal{I}$. For a set of clocks X, we denote by $(\gamma + t)$ the increasing of clocks, i.e., $(\gamma + t)(c) := \gamma(c) + t$, and by $\gamma[X := 0]$ the valuation, where each clock in X is set to zero and the values of the remaining clocks are given by γ.

Definition 1 (Phase Event Automaton). *A* phase event automaton (PEA) *is a tuple $(P, V, A, C, E, s, I, P^0)$, where P is a finite set of locations (phases) with initial locations $P^0 \subseteq P$; V, A, C are finite sets of real-valued variables, events, and real-valued clocks, respectively; $s : P \to \mathcal{L}(V)$, $I : P \to \mathcal{L}^c(C,T)$ assign state invariants resp. clock invariants to phases. The set of edges is $E \subseteq P \times \mathcal{L}(V \cup V' \cup A \cup C) \times \mathbb{P}(C) \times P$. We assume that a* stuttering edge *$(p, \bigwedge_{e \in A} \neg e \wedge \bigwedge_{v \in V} v' = v, \varnothing, p)$ (empty transition) exists for every phase p.*

The operational semantics of PEA is defined by infinite runs of configurations $\langle (p_0, \beta_0, \gamma_0, t_0, Y_0), (p_1, \beta_1, \gamma_1, t_1, Y_1), \cdots \rangle$, where initially $p_0 \in P^0$ and $\gamma_0(c) = 0$ for $c \in C$. For $i \in \mathbb{N}$ and variable valuations β_i (with $\beta_i(v) = \beta_i'(v')$) we demand $\beta(i) \models s(p_i)$ and $\gamma_i + t_i, \mathcal{I} \models I(p_i)$ and $t_i > 0$. For transitions $(p_i, g, X, p_{i+1}) \in E$ we further require $\beta_i, \beta_{i+1}', \gamma_i + t_i, Y_i \models g$ and $\gamma_{i+1} = (\gamma_i + t_i)[X := 0]$.

Thus, a PEA is an automaton enriched by constraints to define data changes and clocks to measure time (similar to a timed automaton). An edge (p_1, g, X, p_2) represents a transition from p_1 to p_2 with a guard g over (possibly primed) variables, clocks, and events, and a set X of clocks that are to be reset. Primed variables v' denote the post-state of v whereas the unprimed v always refers to the pre-state. In the parallel composition of PEA, we consider conjunctions of guards of transitions and invariants of locations.

Definition 2 (Parallel Composition). *The parallel composition of two PEA $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A}_i = (P_i, V_i, A_i, C_i, E_i, s_i, I_i, P_i^0)$, is defined by*

$$\mathcal{A}_1 \mid\mid \mathcal{A}_2 := (P_1 \times P_2, V_1 \cup V_2, A_1 \cup A_2, C_1 \cup C_2, E, s_1 \wedge s_2, I_1 \wedge I_2, P_1^0 \times P_2^0),$$

where $((p_1, p_2), g_1 \wedge g_2, X_1 \cup X_2, (p_1', p_2')) \in E$ iff $(p_i, g_i, X_i, p_i') \in E_i$ with $i = 1, 2$.

The translation of COD specifications into PEA is compositional: every part of the specification is translated separately into PEA; the semantics for the entire specification is the parallel composition of the automata for every part: $\mathcal{A}(COD) = \mathcal{A}(CSP) \parallel \mathcal{A}(OZ) \parallel \mathcal{A}(DC)$.

[1] Ideally, $\mathcal{L}(V)$ should be expressive enough so that specifications with complex data types can be translated, but should permit automatic verification.

Translation of the CSP part. The translation of the CSP part into PEA is based on the structured operational semantics of CSP [Ros98]. If this semantics of the CSP part is given as a labelled transition system $(Q, A, q_0, \longrightarrow)$ with locations Q and events A from the COD specification, its PEA is $\mathcal{A}(\mathit{CSP}) = (Q, \varnothing, A, \varnothing, E, s, I, \{q_0\})$, where $s(q) = \mathit{true}$, $I(q) = \mathit{true}$ for all $q \in Q$ and

$$E = \{(p, \mathit{only}(e), \varnothing, p') \mid p \xrightarrow{e} p'\} \cup \{(p, \mathit{only}(\tau), \varnothing, p) \mid p \in Q\}.$$

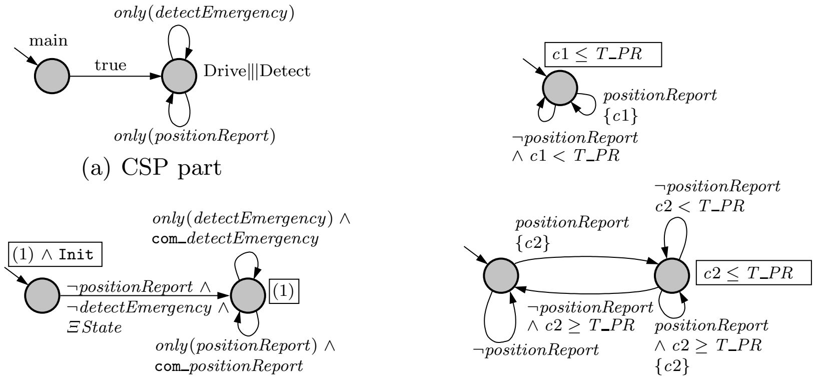
The predicate $\mathit{only}(e)$ demands that only the event e is communicated whereas $\mathit{only}(\tau)$ demands that no event is communicated. That is, E consists of transitions for every transition in the original transition system and of stuttering edges for every location. The PEA of our example's CSP part is pictured in Fig. 3.

Translation of the OZ part. The OZ part of a COD specification is translated into a PEA with two locations: one for setting the initial values of state variables and one for the running system, with a transition for each state changing event. The variables of the PEA are the variables $\mathit{Var}(\mathit{State})$ declared in the state schema. The set A of events of the PEA consists of all COD events for which a communication schema $\mathtt{com\_}c$ exists. For each such event the automaton has a transition executing the state change as defined in the associated communication schema. The resulting PEA is $\mathcal{A}(\mathit{OZ}) = (\{p_0, p_1\}, \mathit{Var}(\mathit{State}), A, \varnothing, E, s, I, \{q_0\})$, where $s(p_0) = \mathtt{Init}$ (invariant from the initial schema), $s(p_1) = \mathit{State}$ (invariant from the state schema), $I(p_i) = \mathit{true}$ for i = 1,2, and

$$\begin{aligned} E = \; & \{(p_1, \mathit{only}(c) \wedge \mathtt{com\_}c, \varnothing, p_1) \mid c \in A\} \cup \\ & \{(p_i, \mathit{only}(\tau) \wedge \Xi \mathit{State}, \varnothing, p_i) \mid i = 1, 2\} \cup \{(p_0, \mathit{only}(\tau) \wedge \Xi \mathit{State}, \varnothing, p_1)\}. \end{aligned}$$

The OZ predicate $\Xi \mathit{State}$ demands that the state space is not changed: $\Xi \mathit{State} :\Leftrightarrow \bigwedge_{v \in \mathit{Var}(\mathit{State})} v' = v$. The formula $\mathtt{com\_}c$ only changes the state of the variables occurring in the Δ list of the corresponding operation schema; the remaining variables remain implicitly unchanged. The OZ part PEA of the RBC is illustrated in Fig. 3. Formula (1) refers to the state schema from Fig. 2. The operation schemata, e.g., $\mathtt{com\_}\mathit{positionReport}$ refer to the constraints of the specification.

Translation of the DC part. Each formula of the DC part is translated into an individual PEA. The translation of counter-example formulae (cf. Sect. 2), e.g., $\neg(\mathit{phase}_0 \frown \mathit{event}_1 \frown \mathit{phase}_1 \frown \cdots \frown \mathit{phase}_n)$, into PEA is similar to the translation of a non-deterministic finite automaton into a deterministic one: every location of the resulting automaton represents a subset of DC phases. Every run of the automaton leading to a location labelled with phase_i accepts the prefix of the DC counter-example up to phase_i. In addition, phase_i may have an upper or lower time bound. In this case, the automaton includes a clock c_i measuring the duration of the phase. Event expressions event_i separating two DC phases constitute the guards that restrict transitions from phase_{i-1} to phase_i. Technical details of the construction can be found in [Hoe06]. The automata for the DC part of the RBC specification are displayed in Fig. 3(c). For instance, the upper automaton enforces the behaviour defined by the second DC formulae



(a) CSP part

(b) OZ part

(c) DC part

Fig. 3. PEA for the RBC case study. Boxes around formulae indicate state invariants; braces ($\{c1\}$,$\{c2\}$) indicate clock resets.; Ξ is defined on page 9.

of our example (Fig. 2). It consists of one location with a clock invariant, i.e, the automaton stays in this location for at most T_PR time units – the only way to reset the clock $c1$ is the transition that synchronises on *positionReport*. By this, every *positionReport* event has to occur in time.

3.2 PEA with Timing Parameters

As we allow timing parameters to occur in the DC part and in convex clock expressions, we need to adapt the translation of the DC part into PEA given in [Hoe06]: since the original translation does not depend on concrete values of rational constants, we can treat timing parameters exactly like rational constants. The clock constraints generated by the translation are then convex as before. Important properties of PEA, e.g., the translation into TCS (cf. Sect. 4), only depend on the convexity of clock constraints. We thus obtain:

Theorem 1. *The translation from COD with complex data types and timing parameters to PEA described here is sound (i.e., the PEA runs are exactly the system runs admitted by the CSP, OZ, and DC part) and compositional.*

3.3 Simplifications of PEA

As mentioned in Section 3.1, the operational semantics of the whole COD specification is given by the parallel product of the individual PEA. This product can grow very large: theoretically, its size is the product of the size of all individual automata, both in terms of locations and in terms of transitions. We propose the following simplifications for the product PEA:

- Transitions whose combined guards evaluate to `false` can be removed, as can be locations that are not connected to an initial state. We can also remove all events from the PEA, as these are only used for synchronising the individual automata [Hoe06].

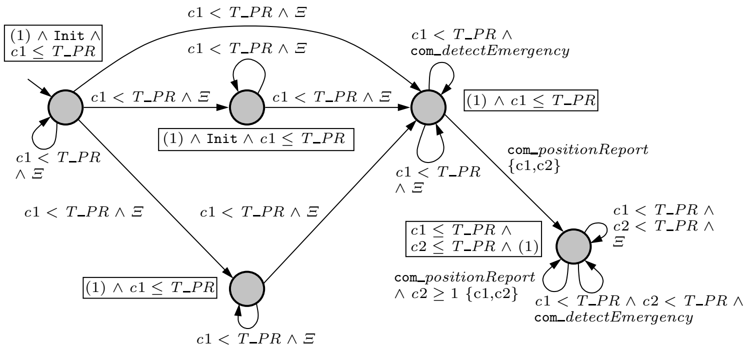


Fig. 4. Simplified product PEA

- Consider the case that we have several clocks in the product automaton. All clocks run at the same speed and can be reset to 0 by transitions. If we have two (or more) clocks for which the set of transitions that resets them is the same, we can identify them, i.e., remove one clock and replace it by the other in all guards. This not only gives us a system with one variable less, but may also remove additional transitions by making their guards `false`.

Theorem 2. *Let* $\mathcal{A} = (P, V, A, C, E, s, I, P^0)$ *be a PEA,* $c_1, c_2 \in C$*. If* $\{(p_1, g, X, p_2) \in E \mid c_1 \in X\} = \{(p_1, g, X, p_2) \in E \mid c_2 \in X\}$*, then* $\mathcal{A}$ *is equivalent to* $\mathcal{A}' = (P, V, A, C \setminus \{c_2\}, E', s', I', P^0)$*, where* E', s' *and* I' *result from* E, s *and* I *by replacing all occurrences of* c_2 *by* c_1*.*

Applied to the running example, the first simplification reduces the product automaton from 8 locations and 90 transitions (without stuttering edges) to 5 locations and 10 transitions. With the second one, we remove one clock variable and one additional transition from the product automaton. The entire product automaton with simplifications is pictured in Fig. 4. It basically comprises four locations representing slightly different possibilities to initialise the system and one dead end location in the lower right corner. The latter is eventually entered and can be interpreted as the main state of the system. Here the trains periodically report their positions to the RBC and emergencies may be detected.

4 Verification of COD Specifications

In this section, we elaborate on verifying safety properties of systems with complex data types specified in COD, based on their translation to PEA. We define transition constraint systems (TCS) and show how to extract a TCS from a PEA. We introduce the verification tasks that we consider. After an intuitive presentation of our idea for efficient verification, we formally analyse situations where verification problems can be reduced to checking satisfiability of ground formulae over a simple theory. The method is illustrated on the running example.

Language and theory associated with a COD specification. Let S be a COD specification. The *signature of* S, Σ_S consists of all sorts, functions and predicates declared in the OZ specification either implicitly (by mentioning standard theories) or explicitly. The theory of S, $\mathcal{T}_S$ is constructed by extending the (many-sorted) combination $\mathcal{T}_0$ of all standard theories used in the OZ and DC specification with the functions declared in the OZ part and the axioms for the data structures specified at the beginning of the OZ part (which we will denote by Ax). In what follows, the theory $\mathcal{T}_S$ will be considered to be a background theory: even if we do not refer to it explicitly, it is always taken into account.

4.1 Translation of PEA to TCS

Let Σ be a signature, $\mathcal{V}$ a set of (typed) variables, and $\mathcal{V}'$ a copy of corresponding primed variables. Let $\mathcal{F}(X)$ be the family of all Σ-formulae in the variables X.

Definition 3 (Transition Constraint Systems). *A* transition constraint system *T is a tuple $(\mathcal{V}, \Theta, \Phi)$ with $\Theta \in \mathcal{F}(\mathcal{V})$ and $\Phi \in \mathcal{F}(\mathcal{V} \cup \mathcal{V}')$. The formula Θ characterises the* initial states *of T, Φ the* transition constraints *of T (i.e., the relationships between the variables $\mathcal{V}$ and $\mathcal{V}'$ – before and after transitions).*

A translation from PEA to TCS has been developed by Hoenicke and Maier [HM05, Hoe06]. We use the simplified translation from [Hoe06].

Let S be a COD specification and $\mathcal{A}_S = (P, V, A, C, E, s, I, P_0)$ the PEA associated with S. We associate with $\mathcal{A}_S$ the TCS $T(\mathcal{A}_S) = (\mathcal{V}, \Theta, \Phi)$, where:

- $\mathcal{V} = V \cup A \cup C \cup \{\mathsf{len}, \mathsf{pc}\}$, where len is a real-valued variable representing the time spent in the current state, and pc is the *program counter* that is interpreted over P and represents the current location (phase) of the system.
- $\Theta = \bigvee_{p \in P_0} \mathsf{pc}{=}p \wedge \mathsf{len}{>}0 \wedge \bigwedge_{c \in C} c{=}\mathsf{len} \wedge s(p) \wedge I(p)$. Θ requires that there exists an initial location p in which a positive time len is spent, all clocks are set to len and both the state and the clock invariant of the location p hold.
- $\Phi = \bigvee_{(p_1,g,X,p_2) \in E} \mathsf{pc}{=}p_1 {\wedge} \mathsf{pc}'{=}p_2 {\wedge} g {\wedge} s'(p_2) {\wedge} I'(p_2) {\wedge} \mathsf{len}'{>}0 {\wedge} \bigwedge_{c \in X} c'{=}\mathsf{len}' {\wedge} \bigwedge_{c \in C \setminus X} c'{=}c + \mathsf{len}'$, where s' and I' represent s and I with unprimed variables replaced by primed ones. The formula Φ states that there exists a transition (p_1, g, X, p_2) such that the program counter is p_1 before and p_2 after the transition, the guard g of the transition as well as the state and clock invariant of location p_2 are satisfied, the system will remain in p_2 for some positive time, and clocks are incremented by len' if they are not reset by the transition (otherwise they are set to len').

We thus obtain a representation of the original COD specification S in terms of first-order formulae over the signature Σ_S and theory $\mathcal{T}_S$ of S. We encode states of the system by formulae over $\mathcal{V}$. If σ, σ' are states (encoded as formulae over $\mathcal{V}$, and $\mathcal{V}'$ respectively), we say that σ is reachable in one step from σ in $T_S = (\mathcal{V}, \Theta, \Phi)$ w.r.t. $\mathcal{T}_S$ if $\mathcal{T}_S, \sigma, \sigma' \models \Phi$. A run of T_S is a sequence of states $\langle \sigma_1, \dots \sigma_m \rangle$ such that $\mathcal{T}_S, \sigma_1 \models \Theta$ and σ_{i+1} is reachable in one step from σ_i.

As a consequence of the results in [Hoe06] we obtain:

Corollary 1. *The translation from PEA to TCS preserves the semantics: every run in the TCS can be mapped to a run of the PEA; the mapping is surjective.*

Example 1. Consider the RBC example discussed in Sect. 1.1. We use the simplified PEA $\mathcal{A} = (P, V, A, C, E, s, I, P_0)$ developed in Sect. 3.3, where $A = \varnothing$ and $C = \{x_1\}$. The TCS $T(\mathcal{A}) = (\mathcal{V}, \Theta, \Phi)$ associated with $\mathcal{A}$ is defined as follows[2]:

(1) $\mathcal{V} = V \cup A \cup C \cup \{\mathsf{len}, \mathsf{pc}\}$. For technical reasons we model the variables of type sequence (e.g. $\mathsf{train}, \mathsf{speed}$) as functions of sort $\mathsf{i} \rightarrow \mathsf{num}$.

The following formulae (extracted from the OZ specification) help define $T(\mathcal{A})$:

$\phi_{\mathsf{input}} = \mathsf{newEmergencyTrain}' > 0,$

$\phi_{\mathsf{clock}} = x_1{<}1 \wedge \mathsf{len}'{>}0 \wedge x_1'{=}x_1 + \mathsf{len}' \wedge x_1'{\leq}1,$

$$\begin{aligned}\phi_{\mathsf{init}} = \ & (\forall\, i : 1{<}i{\leq}\mathsf{n} \rightarrow \mathsf{train}(i){<}\mathsf{train}(i-1){-}\mathsf{brakingDist}(\mathsf{speed}(i))){\wedge} \\ & (\forall\, i : 1{\leq}i{\leq}\mathsf{n} \rightarrow \mathsf{minSpeed}{\leq}\mathsf{speed}(i){\leq}\mathsf{maxSpeed}) \ \wedge \ (\mathsf{emergencyTrain}{>}\mathsf{n}),\end{aligned}$$

$$\begin{aligned}\phi_{\mathsf{emerg}} = \ & \mathsf{newEmergencyTrain} \leq \mathsf{n} \ \wedge \\ & \mathsf{emergencyTrain}' = \mathsf{min}\,\{\mathsf{newEmergencyTrain}, \mathsf{emergencyTrain}\} \ \wedge \\ & \mathsf{speed}'(\mathsf{emergencyTrain}') = 0 \ \wedge \\ & \forall\, i : i \neq \mathsf{emergencyTrain}' \rightarrow \mathsf{speed}'(i) = \mathsf{speed}(i),\end{aligned}$$

$$\begin{aligned}\phi_{\mathsf{posRep}} = \ & \forall\, i : i{=}1 \wedge \mathsf{emergencyTrain}{>}i \rightarrow \mathsf{minSpeed}{\leq}\mathsf{speed}'(i){\leq}\mathsf{maxSpeed} \ \wedge \\ & \forall\, i : 1{<}i{<}\mathsf{emergencyTrain} \wedge \mathsf{train}(i-1){-}\mathsf{train}(i){\geq}\mathsf{d} \\ & \qquad \rightarrow \mathsf{minSpeed}{\leq}\mathsf{speed}'(i){\leq}\mathsf{maxSpeed} \ \wedge \\ & \forall\, i : 1{<}i{<}\mathsf{emergencyTrain} \wedge \mathsf{train}(i-1){-}\mathsf{train}(i){<}\mathsf{d} \\ & \qquad \rightarrow \mathsf{speed}'(i){=}\mathsf{minSpeed} \ \wedge \\ & \forall\, i : i{\geq}\mathsf{emergencyTrain} \rightarrow \mathsf{speed}'(i){=}\mathsf{max}\,\{\mathsf{speed}(i){-}\mathsf{maxDec}{*}T\_PR, 0\} \wedge \\ & \forall\, i : 1{\leq}i{\leq}\mathsf{n} \rightarrow \mathsf{train}'(i){=}\mathsf{train}(i){+}\mathsf{speed}'(i){*}T\_PR,\end{aligned}$$

$\phi_{\mathsf{const}} = \bigwedge_{c \in \mathsf{const}} c' = c$, where $\mathsf{const} = \{\mathsf{maxDec}, \mathsf{maxSpeed}, \mathsf{minSpeed}, \mathsf{n}, \mathsf{d}, T\_PR\}$ is the set of all variables in $\mathcal{V}$ that do not change during execution.

(2) The initial predicate is $\Theta = \mathsf{pc} = 1 \wedge \mathsf{len} > 0 \wedge x_1 = \mathsf{len} \wedge \ \phi_{\mathsf{init}}$.

(3) We describe the transition relation Φ in terms of the individual transitions. Several transitions change only the clock, but no state variables. Let $S_1 = \{(1,1), (1,3), (3,3)\}$, $S_2 = \{(1,2), (1,4), (2,2), (2,4), (4,4)\} \subset P \times P$, and

$$\phi_{(i,j)} = \begin{cases} (\mathsf{pc}{=}i \wedge \mathsf{pc}'{=}j \wedge \phi_{\mathsf{clock}} \wedge \phi_{\mathsf{init}} \wedge \phi_{\mathsf{const}} \wedge \phi_{\mathsf{input}}) & \text{if } (i,j) \in S_1 \\ (\mathsf{pc}{=}i \wedge \mathsf{pc}'{=}j \wedge \phi_{\mathsf{clock}} \wedge \phi_{\mathsf{const}} \wedge \phi_{\mathsf{input}}) & \text{if } (i,j) \in S_2 \end{cases}.$$

Finally, we have the following transitions that change the state variables:

$\phi_1 = (\mathsf{pc}{=}4 \wedge \mathsf{pc}'{=}4 \wedge \phi_{\mathsf{emerg}} \wedge \phi_{\mathsf{clock}} \wedge \phi_{\mathsf{const}} \wedge \phi_{\mathsf{input}}),$

$\phi_2 = (\mathsf{pc}{=}4 \wedge \mathsf{pc}'{=}5 \wedge \phi_{\mathsf{posRep}} \wedge \mathsf{len}'{>}0 \wedge x_1'{=}0 \wedge x_1' \leq 1 \wedge \phi_{\mathsf{const}} \wedge \phi_{\mathsf{input}}),$

$\phi_3 = (\mathsf{pc}{=}5 \wedge \mathsf{pc}'{=}5 \wedge \phi_{\mathsf{emerg}} \wedge \phi_{\mathsf{clock}} \wedge \phi_{\mathsf{const}} \wedge \phi_{\mathsf{input}}),$

$\phi_4 = (\mathsf{pc}{=}5 \wedge \mathsf{pc}'{=}5 \wedge \phi_{\mathsf{posRep}} \wedge \mathsf{len}'{>}0 \wedge x_1'{=}0 \wedge x_1'{\leq}1 \wedge \phi_{\mathsf{const}} \wedge \phi_{\mathsf{input}}).$

Altogether, $\Phi = \bigvee_{(i,j) \in S_1 \cup S_2} \phi_{(i,j)} \vee \bigvee_{i=1}^{4} \phi_i$.

[2] We will use a sans serif font for all symbols in the signature of the TCS $T(\mathcal{A})$.

4.2 Verification of TCS

The verification problems we consider are invariant checking and bounded model checking. We explain the problems which occur in this context, and present an idea that allows to solve these problems in certain situations. We illustrate the problems as well as the verification methods on our case study.

Invariant checking. We can check whether a formula $\varPsi$ is an inductive invariant of a TCS $T{=}(\mathcal{V}, \varTheta, \varPhi)$ in two steps: (1) prove that $\mathcal{T}_S, \varTheta \models \varPsi$; (2) prove that $\mathcal{T}_S, \varPsi, \varPhi \models \varPsi'$, where $\varPsi'$ results from $\varPsi$ by replacing every $x \in \mathcal{V}$ by x'. Failure to prove (2) means that $\varPsi$ is not an invariant, or $\varPsi$ is not inductive w.r.t. T.[3]

Example 2. For the system described in Sect. 1.1, let $\varPsi$ be the formula that states that the distance between two trains must always be greater than the sum of the braking distances of the trains in between ($\varPsi$ is a safety condition):

$$\varPsi = \forall\, i : 1 < i \leq \mathsf{n} \rightarrow \mathsf{train}(i) < \mathsf{train}(i-1) - \mathsf{brakingDist}(\mathsf{speed}(i)).$$

To check that $\varPsi$ is an inductive invariant, we need to check that:

(1) The initial states of the system, given by $\varTheta$, satisfy the safety property $\varPsi$.
(2) Assuming that a given state σ satisfies $\varPsi$, any state σ' reachable from σ using the transition predicate $\varPhi$ satisfies $\varPsi'$.

Checking (1) is not a problem. For (2) we need to show $\mathcal{T}_S \models \varPsi \wedge \varPhi \rightarrow \varPsi'$, where $\mathcal{T}_S$ is the theory associated with the COD specification, an extension of $\mathcal{T}_0$ (many-sorted combination of real arithmetic (sort num) with an index theory describing precedence of trains (sort i)), with the set of definitions $\mathsf{Def} \subseteq \mathsf{Ax}$ for global constants of the system (Ax are the axioms in the OZ specification) and with function symbols $\mathsf{brakingDist}, \mathsf{train}, \mathsf{train}', \mathsf{speed}, \mathsf{speed}'$ fulfilling the axioms specified in Ax, $\varPsi$, and $\varPhi$. We need to show that $\mathcal{T}_0 \wedge \mathsf{Ax} \wedge \varPsi \wedge \varPhi \wedge \neg\varPsi' \models\perp$.

Bounded model checking. We check whether, for a fixed k, unsafe states are reachable by runs of $T{=}(\mathcal{V}, \varTheta, \varPhi)$ of length at most k. Formally, we check whether:

$$\mathcal{T}_S \wedge \varTheta_0 \wedge \bigwedge_{i=1}^{j} \varPhi_i \wedge \neg\varPsi_j \models\perp \qquad \text{for all } 0 \leq j \leq k,$$

where $\varPhi_i$ is obtained from $\varPhi$ by replacing all variables $x \in \mathcal{V}$ by x_i, and all variables $x' \in \mathcal{V}'$ by x_{i+1}; $\varTheta_0$ is $\varTheta$ with x_0 replacing $x \in \mathcal{V}$; and $\varPsi_i$ is $\varPsi$ with x_i replacing $x \in \mathcal{V}$.

[3] Proving that a $\varPsi$ is an invariant of the system in general requires to find a stronger formula $\varGamma$ (i.e., $\mathcal{T}_0 \models \varGamma \rightarrow \varPsi$) and prove that $\varGamma$ is an inductive invariant.

Problem. Standard combination methods [NO79, Ghi04] allow for testing satisfiability in certain combinations of theories, but only for ground formulae. Our problem contains several non-ground formulae: the global axioms Ax, the invariant Ψ and the transition relation Φ. Only $\neg\Psi'$ corresponds to a ground set of clauses. Thus, standard methods are not directly applicable. We want to reduce the problem above to a ground satisfiability problem over decidable theories. To this end, we may replace quantified formulae by a number of ground instances, giving a decidable ground satisfiability problem over the base theory $\mathcal{T}_0$ (plus free function symbols). This approach is sound, but in general not complete. In what follows, we identify situations when this method is complete.

Our idea. In order to overcome the problem mentioned above we proceed as follows. We start from a base theory $\mathcal{T}_0$ associated with the COD specification S (usually a many-sorted combination of standard theories, e.g., integers or reals). For the case of invariant checking we consider the following successive extensions of $\mathcal{T}_0$ and study possibilities of efficient reasoning in these extensions:

- the extension $\mathcal{T}_1$ of $\mathcal{T}_0$ with the definitions and axioms in the OZ part of the COD specification for variables which do not occur in Φ (i.e., do not change);
- the extension $\mathcal{T}_2$ of $\mathcal{T}_1$ with the remaining variables in a set $\mathcal{V}$ (including those of sort $\mathsf{sequence}$, modelled by functions) which occur in Φ and satisfy Ψ (together with the corresponding definitions and axioms);
- the extension $\mathcal{T}_3$ of $\mathcal{T}_2$ with primed variables $\mathcal{V}'$ (including primed versions of functions for the variables of sort $\mathsf{sequence}$) satisfying Φ.

Example 3. Again consider the running example. We consider successive extensions of $\mathcal{T}_0$, a many-sorted combination of real arithmetic (for reasoning about time, positions and speed, sort num) with an index theory (for describing precedence between trains, sort i). For the case of invariant checking, we have:

- the extension $\mathcal{T}_1$ of $\mathcal{T}_0$ with a monotone and bounded function $\mathsf{brakingDist}$ as well as global constants, defined by $\mathsf{Def} \subseteq \mathsf{Ax}$,
- the extension $\mathcal{T}_4$ of $\mathcal{T}_1$ with $\mathcal{V}$-variables from Φ, satisfying Ψ, defined by:
  - Let $\mathcal{T}_2$ be the extension of $\mathcal{T}_1$ with the (free) function speed.
  - Let $\mathcal{T}_3$ be the extension of $\mathcal{T}_2$ with the binary function secure defined for every $0 < i < j < n$ by $\mathsf{secure}(i,j) = \sum_{k=i+1}^{j} \mathsf{brakingDist}(\mathsf{speed}(k))$.
  - $\mathcal{T}_4$ is the extension of $\mathcal{T}_3$ with function train satisfying $\overline{\Psi}$ (equivalent to Ψ):
$$\overline{\Psi} = \forall\, i,j (0 < i < j \leq \mathsf{n} \rightarrow \mathsf{train}(j) < \mathsf{train}(i) - \mathsf{secure}(i,j)),$$
- the extension $\mathcal{T}_5$ of $\mathcal{T}_4$ with functions train' and speed' satisfying Φ.

We show that for all of these extensions hierarchic reasoning is possible (cf. Sect. 4.3).[4] This allows us to reduce problem (2) to testing satisfiability of ground clauses in $\mathcal{T}_0$, for which standard methods for reasoning in combinations of theories can be applied. A similar method can be used for bounded model checking.

[4] We consider extensions with axiom $\overline{\Psi}$ instead of Ψ since $\overline{\Psi}$ defines a local theory extension, and hence it allows for hierarchic reasoning (cf. Sect. 4.3), whereas Ψ does not have this property. We are currently studying possibilities of automatically recognising local theory extensions, and of automatically generating (equivalent) sets of axioms defining local extensions from given sets of axioms.

4.3 Efficient Reasoning in Complex Theories: Locality

In the following, we identify situations in which we can give sound, complete and efficient methods for reasoning in theory extensions.

Local theory extensions. Let $\mathcal{T}_0$ be a theory with signature $\Pi_0 = (S_0, \Sigma_0, \mathsf{Pred})$. We consider extensions with new sorts S_1 and new function symbols Σ_1 constrained by a set $\mathcal{K}$ of (universally quantified) clauses in signature $\Pi = (S, \Sigma, \mathsf{Pred})$, where $S = S_0 \cup S_1$ and $\Sigma = \Sigma_0 \cup \Sigma_1$. We are interested in checking satisfiability of sets of ground clauses G with respect to such theory extensions.

When referring to sets G of ground clauses we assume they are in the signature $\Pi^c = (S, \Sigma \cup \Sigma_c, \mathsf{Pred})$ where Σ_c is a set of new constants. An extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is *local* if satisfiability of a set G of clauses w.r.t. $\mathcal{T}_0 \cup \mathcal{K}$ only depends on $\mathcal{T}_0$ and those instances $\mathcal{K}[G]$ of $\mathcal{K}$ in which the terms starting with extension functions are in the set $\mathsf{st}(\mathcal{K}, G)$ of ground terms which already occur in G or $\mathcal{K}$. Formally, the extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is local if condition (Loc) holds:

(Loc) For every set G of ground clauses, $\mathcal{T}_0 \wedge \mathcal{K} \wedge G$ is unsatisfiable iff $\mathcal{T}_0 \wedge \mathcal{K}[G] \wedge G$ has no partial model where all terms in $\mathsf{st}(\mathcal{K}, G)$ are defined

A partial model of $\mathcal{T}_0 \wedge \mathcal{K}[G] \wedge G$ is a partial Π^c-structure P s.t. $P_{|\Pi_0}$ is a total model of $\mathcal{T}_0$ and P satisfies all clauses in $\mathcal{K}[G] \wedge G$ where all terms are defined.

We give examples of local theory extensions relevant for the verification tasks we consider. Some appear in [GSSW06, SS05, SS06], some are new.

Theorem 3. *The extension of any theory with* free function symbols *is local. In addition, assume the base theory has a reflexive partial ordering $\leq$. Then:*

(1) Extensions of $\mathcal{T}_0$ with axioms of the following type are also local:
(GBound_f^t) $\quad \forall x_1, \dots, x_n (\phi(x_1, \dots, x_n) \rightarrow f(x_1, \dots, x_n) \leq t(x_1, \dots, x_n))$
where $t(x_1, \dots, x_n)$ is a term, $\phi(x_1, \dots, x_n)$ a conjunction of literals, both in the base signature Π_0 and with variables among $x_1, \dots, x_n$.

(2) For $i \in \{1, \dots, m\}$, let $t_i(x_1, \dots, x_n)$ and $s_i(x_1, \dots, x_n)$ be terms and let $\phi_i(x_1, \dots, x_n)$ be conjunctions of literals, all of them in the base signature Π_0, with variables among $x_1, \dots, x_n$, such that for every $i \neq j$, $\phi_i \wedge \phi_j \models_{\mathcal{T}_0} \perp$. Any "piecewise-bounded" *extension $\mathcal{T}_0 \wedge (\mathsf{GBound}_f)$, where f is an extension symbol, is local. Here $(\mathsf{GBound}_f) = \bigwedge_{i=1}^m (\mathsf{GBound}_f^{[s_i,t_i],\phi_i})$;*
$(\mathsf{GBound}_f^{[s_i,t_i],\phi_i})$ $\quad \forall \overline{x} (\phi_i(\overline{x}) \rightarrow s_i(\overline{x}) \leq f(\overline{x}) \leq t_i(\overline{x}))$.

(3) For many ordered theories including the reals (for a complete list see [SS05, SS06, JSS07]), extensions with (possibly strictly) monotone functions *are local. Combinations with boundedness axioms (GBound_f^t), where t has the same monotonicity as f, do not destroy locality.*

Hierarchic reasoning in local theory extensions. Let $\mathcal{T}_0 \subseteq \mathcal{T}_1 {=} \mathcal{T}_0 \cup \mathcal{K}$ be a local theory extension. To check the satisfiability of a set G of ground clauses w.r.t. $\mathcal{T}_1$ we can proceed as follows (for details cf. [SS05]):

Step 1: Use locality. By the locality condition, G is unsatisfiable w.r.t. $\mathcal{T}_1$ iff $\mathcal{K}[G] \wedge G$ has no partial model in which all the subterms of $\mathcal{K}[G] \wedge G$ are defined, and whose restriction to Π_0 is a total model of $\mathcal{T}_0$.

Step 2: Flattening and purification. We purify and flatten $\mathcal{K}[G] \wedge G$ by introducing new constants for the arguments of the extension functions as well as for the (sub)terms $t = f(g_1, \ldots, g_n)$ starting with extension functions $f \in \Sigma_1$, together with corresponding new definitions $c_t \approx t$. The set of clauses thus obtained has the form $\mathcal{K}_0 \wedge G_0 \wedge D$, where D is a set of ground unit clauses of the form $f(c_1, \ldots, c_n) \approx c$, where $f \in \Sigma_1$ and $c_1, \ldots, c_n, c$ are constants, and $\mathcal{K}_0, G_0$ are clause sets without function symbols in Σ_1.

Step 3: Reduction to testing satisfiability in $\mathcal{T}_0$. We reduce the problem to testing satisfiability in $\mathcal{T}_0$ by replacing D with the following set of clauses:

$$N_0 = \bigwedge \{ \bigwedge_{i=1}^{n} c_i = d_i \rightarrow c = d \mid f(c_1, \ldots, c_n) = c, f(d_1, \ldots, d_n) = d \in D \}.$$

Theorem 4 ([SS05]). *Assume that $\mathcal{T}_0 \cup \mathcal{K}$ is a local extension of $\mathcal{T}_0$. With the notations above, G is satisfiable in $\mathcal{T}_0 \cup \mathcal{K}$ iff $\mathcal{K}_0 \wedge G_0 \wedge N_0$ is satisfiable in $\mathcal{T}_0$.*

The method above is easy to implement and efficient. If all the variables in $\mathcal{K}$ are guarded by extension functions then the size of $\mathcal{K}_0 \wedge G_0 \wedge N_0$ is polynomial in the size of G. Thus, the complexity of checking the satisfiability of G w.r.t. $\mathcal{T}_0 \wedge \mathcal{K}$ is $g(n^k)$ (where k depends on $\mathcal{K}$ cf. e.g. [SS05]) where $g(n)$ is the complexity of checking the satisfiability of a set of ground clauses of size n in $\mathcal{T}_0$.

Application to parametric verification of COD specifications. A sound but potentially incomplete method for checking whether $\mathcal{T}_0 \wedge \mathsf{Ax} \wedge \Psi \wedge \Phi \wedge \neg \Psi' \models \perp$, which can always be used, is to take into account only certain ground instances of the universally quantified formulae in $\mathsf{Ax} \wedge \Psi \wedge \Phi$, related to the ground formula $G = \neg \Psi'$. However *complete* approaches can often be obtained, because many axioms used in verification problems define chains of local extensions of $\mathcal{T}_0$:

- definitions for constants can be expressed by axioms of the type GBound^t_f;
- often, transition relations which reflect updates of variables or of a sequence f according to mutually exclusive "modes of operation" are axiomatised by axioms of the form (GBound_f) as defined in Theorem 3(2) and 3(3).

If a complete approach can be given, the method for hierarchic reasoning described above can be used in two ways. If the constraints on the parameters of the systems are completely specified in the COD specification, then it allows us to reduce the problem of checking whether a system property Ψ is an inductive invariant to the problem of deciding satisfiability of a set of constraints in $\mathcal{T}_0$. Alternatively, we may choose not to specify all constraints on the parameters. 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 (in the running example these are, e.g., T_PR, $\mathsf{minSpeed}, \mathsf{maxSpeed}, ...$), which guarantee that the property is an inductive invariant and are sufficient for this. (This can be achieved for instance using quantifier elimination.)

4.4 Example: The RBC Case Study

We show how the verification method based on hierarchic reasoning can be applied to our case study. The following is a consequence of Theorem 3.

Theorem 5. *Let $\mathcal{T}_0$ be the (many-sorted) combination of real arithmetic (for reasoning about time, positions and speed, sort* num*) with an index theory (for describing precedence between trains, sort* i*).*

(1) The extension $\mathcal{T}_1$ of $\mathcal{T}_0$ with a monotone and bounded function $\mathsf{brakingDist}$ *as well as global constants, with definitions* $\mathsf{Def} \subseteq \mathsf{Ax}$*, is local.*
(2) The extension $\mathcal{T}_2$ of $\mathcal{T}_1$ with the (free) function speed *is local.*
(3) The extension $\mathcal{T}_3$ of $\mathcal{T}_2$ with the function secure *(cf. Example 3) is local.*
(4) The extension $\mathcal{T}_4$ of $\mathcal{T}_3$ with functions train *satisfying $\overline{\Psi}$ (Example 3) is local.*
(5) The extension $\mathcal{T}_5$ of $\mathcal{T}_4$ with functions train' *and* speed' *satisfying Φ is local.*

We follow the steps in the method for hierarchic reasoning in Sect. 4.3 and reduce the verification task to a satisfiability problem in the base theory $\mathcal{T}_0$. To make our task slightly simpler, we split the transition relation and look at every ϕ_i separately. Those ϕ_i that do not change the state variables train and speed are sure to preserve the invariant. Furthermore, the program counters do not interfere with the invariant. As a result, we have two interesting cases:

$$\Phi_1 = \phi_{\mathsf{emerg}} \wedge \phi_{\mathsf{clock}} \wedge \phi_{\mathsf{const}} \wedge \phi_{\mathsf{input}},$$
$$\Phi_2 = \phi_{\mathsf{posRep}} \wedge \mathsf{len}' > 0 \wedge x_1' = 0 \wedge x_1' \leq 1 \wedge \phi_{\mathsf{const}} \wedge \phi_{\mathsf{input}}.$$

We start with the first transition. We have to prove $\mathcal{T}_0 \wedge \mathsf{Def} \wedge \mathsf{Def}_{\mathsf{secure}} \wedge \overline{\Psi} \wedge \Phi_1 \wedge \neg\Psi' \models \perp$. This is a satisfiability problem over $\mathcal{T}_4 \wedge \Phi_1$. In a first reduction, this problem is reduced to a problem over $\mathcal{T}_4 = \mathcal{T}_0 \wedge \mathsf{Def} \wedge \mathsf{Def}_{\mathsf{secure}} \wedge \overline{\Psi}$:

Step 1: Use locality. For this step, the set of ground clauses we consider is $G = \neg\Psi' = \{1 < k_1, k_1 \leq \mathsf{n}, k_2 = k_1 + 1, s = \mathsf{speed}'(k_1), \mathsf{train}'(k_1) \geq \mathsf{train}'(k_2) - \mathsf{brakingDist}(s)\}$. Of the extension symbols train' and speed', only speed' occurs in Φ_1. Ground terms with speed' are $\mathsf{speed}'(k_1)$ in G and $\mathsf{speed}'(\mathsf{emergencyTrain}')$ in Φ_1. Thus, $\Phi_1[G]$ consists of two instances of Φ_1: one with i instantiated to k_1, the other with i instantiated to $\mathsf{emergencyTrain}'$ (we remove clauses that are generated in both instantiations such that they only appear once):

$$\begin{aligned}\Phi_1[G] = \;& \phi_{\mathsf{clock}} \;\wedge\; \phi_{\mathsf{input}} \;\wedge\; \phi_{\mathsf{const}} \;\wedge\; \mathsf{newEmergencyTrain} \leq \mathsf{n} \;\wedge \\ & \mathsf{emergencyTrain}' = \mathit{min}\,\{\mathsf{newEmergencyTrain}, \mathsf{emergencyTrain}\} \;\wedge \\ & \mathsf{speed}'(\mathsf{emergencyTrain}') = 0 \;\wedge \\ & k_1 \neq \mathsf{emergencyTrain}' \rightarrow \mathsf{speed}'(k_1) = \mathsf{speed}(k_1) \;\wedge \\ & \mathsf{emergencyTrain}' \neq \mathsf{emergencyTrain}' \\ & \quad \rightarrow \mathsf{speed}'(\mathsf{emergencyTrain}') = \mathsf{speed}(\mathsf{emergencyTrain}').\end{aligned}$$

Step 2: Flattening and purification. $\Phi_1[G] \wedge G$ is already flat with respect to speed' and train'. We purify the set of clauses by replacing every ground term with speed' or train' at the root with new constants $c_1, \ldots, c_4$ and obtain a set of definitions $D = \{\mathsf{speed}'(\mathsf{emergencyTrain}') = c_1, \mathsf{speed}'(k_1) = c_2, \mathsf{train}'(k_1) = c_3, \mathsf{train}'(k_2) = c_4\}$, together with the purified sets of clauses

$$\begin{aligned}
G_0 = \ & \{1 < k_1, k_1 \leq \mathsf{n}, k_2 = k_1 + 1, s = c_2, c_3 \geq c_4 - \mathsf{brakingDist}(s)\} \\
\Phi_1[G]_0 = \ & \phi_{\mathsf{clock}} \ \wedge \ \phi_{\mathsf{input}} \ \wedge \ \phi_{\mathsf{const}} \ \wedge \ \mathsf{newEmergencyTrain} \leq \mathsf{n} \ \wedge \\
& \mathsf{emergencyTrain}' = \mathit{min} \, \{\mathsf{newEmergencyTrain}, \mathsf{emergencyTrain}\} \ \wedge \\
& c_1 = 0 \ \wedge \ k_1 \neq \mathsf{emergencyTrain}' \rightarrow c_2 = \mathsf{speed}(k_1) \ \wedge \\
& \mathsf{emergencyTrain}' \neq \mathsf{emergencyTrain}' \rightarrow c_1 = \mathsf{speed}(\mathsf{emergencyTrain}').
\end{aligned}$$

Step 3: Reduction to satisfiability in $\mathcal{T}_4$. We add the set of clauses $N_0 = \{\mathsf{emergencyTrain}' = k_1 \rightarrow c_1 = c_2, k_1 = k_2 \rightarrow c_3 = c_4\}$. This allows us to remove D and obtain a ground satisfiability problem in $\mathcal{T}_4 : \Phi_1[G]_0 \wedge G_0 \wedge N_0$. In four further reduction steps, we reduce this problem (using a similar procedure) to a ground satisfiability problems over $\mathcal{T}_0$. This set of clauses can now directly be handed to a decision procedure for the combination of the theories of reals and indices. In the same way, the transition Φ_2 can be handled.

5 Conclusions

In this paper, we presented a method for invariant checking and bounded model checking for complex specifications of systems containing information about processes, data, and time. In order to represent these specifications in full generality, we used the specification language CSP-OZ-DC (COD) [HO02, Hoe06]. Similar combined specification formalisms are, e.g., [MD99, Smi02, Süh02] but we prefer COD due to its strict separation of control, data, and time, and to its compositionality (cf. Sect. 3), that is essential for automatic verification.

One of our goals was to model complex systems with a parametric number of components. For this, it was essential to use complex data structures (e.g., arrays, functions). Therefore, in this paper we needed to extend existing verification techniques for COD [HM05, MFR06] to situations when abstract data structures appear. Also, in order to achieve a tighter binding of the OZ to the DC part, we introduced timing parameters, allowing for more flexible specifications.

We showed that, in this context, invariant checking or bounded model checking can be reduced to proving in complex theories. This was done using translations from COD to PEA (and then to simplified PEA) and from PEA to TCS (these translations can be fully automated – we already have tool support for them). We then analysed the type of theories that occur in relationship with a given COD specification, and presented a sound method for efficient reasoning in these theories. At the same time, we identified situations when the method is sound and complete (i.e., when the specific properties of “position updates” can be expressed by using chains of local theory extensions). All these ideas were illustrated by means of a running example complementing scenarios studied in [FM06, MFR06] (as now we consider an arbitrary number of trains) and in [JSS07] (as now we also encompass efficient handling of emergency messages). We kept the running example relatively easy in order to ensure clarity of presentation. More complicated scenarios can be handled similarly (we also considered, e.g., situations in which time passes between position and speed updates).

In ongoing work, we investigate possibilities to use methods for abstraction-based model checking and invariant generation for this type of models.

References

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[ERT02] ERTMS User Group, UNISIG. ERTMS/ETCS System requirements specification. `http://www.aeif.org/ccm/default.asp`, 2002. Version 2.2.2.

[FM06] J. Faber and R. Meyer. Model checking data-dependent real-time properties of the European Train Control System. In *FMCAD*, pages 76–77. IEEE Computer Society, 2006.

[Ghi04] S. Ghilardi. Model theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning*, 33(3–4):221–249, 2004.

[GSSW06] 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.

[HJU05] H. Hermanns, D.N. Jansen, and Y.S. Usenko. From StoCharts to MoDeST: a comparative reliability analysis of train radio communications. In *Workshop on Software and Performance*, pages 13–23. ACM Press, 2005.

[HM05] J. Hoenicke and P. Maier. Model-checking of specifications integrating processes, data and time. In *FM 2005*, volume 3582 of *LNCS*. Springer, 2005.

[HO02] J. Hoenicke and E.-R. Olderog. CSP-OZ-DC: A combination of specification techniques for processes, data and time. *Nordic Journal of Computing*, 9(4):301–334, 2002. Appeared March 2003.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Hoe06] J. Hoenicke. *Combination of Processes, Data, and Time*. PhD thesis, University of Oldenburg, Germany, 2006.

[JSS07] S. Jacobs and V. Sofronie-Stokkermans. Applications of hierarchic reasoning in the verification of complex systems. *ENTCS (special issue dedicated to PDPAR 2006)*, 2007. 15 pages. To appear.

[MD99] B. P. Mahony and J. S. Dong. Overview of the semantics of TCOZ. In *IFM*, pages 66–85. Springer, 1999.

[MFR06] R. Meyer, J. Faber, and A. Rybalchenko. Model checking duration calculus: A practical approach. In *ICTAC*, volume 4281 of *LNCS*, pages 332–346. Springer, 2006.

[NO79] G. Nelson and D.C. Oppen. Simplification by cooperating decision procedures. *ACM TOPLAS*, 1(2):245–257, 1979.

[Ros98] A.W. Roscoe. *Theory and Practice of Concurrency*. Prentice Hall, 1998.

[Smi00] G. Smith. *The Object Z Specification Language*. Kluwer Academic, 2000.

[Smi02] G. Smith. An integration of real-time Object-Z and CSP for specifying concurrent real-time systems. In *IFM*, volume 2335 of *LNCS*, pages 267–285. Springer, 2002.

[SS05] V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE*, LNCS 3632, pages 219–234. Springer, 2005.

[SS06] V. Sofronie-Stokkermans. Interpolation in local theory extensions. In *IJCAR*, LNCS 4130, pages 235–250. Springer, 2006.

[Süh02] Carsten Sühl. An overview of the integrated formalism RT-Z. *Formal Asp. Comput*, 13(2):94–110, 2002.

[TZ06] J. Trowitzsch and A. Zimmermann. Using UML state machines and petri nets for the quantitative investigation of ETCS. In *VALUETOOLS*, pages 1–34. ACM Press, 2006.

[ZH04] C. Zhou and M. R. Hansen. *Duration Calculus*. Springer, 2004.