*Journal of Universal Computer Science, vol. 6, no. 4 (2000), 460-473*
*submitted: 30/4/99, accepted: 28/1/00, appeared: 28/4/00* *© Springer Pub. Co.*

# Specifying and Verifying Real–Time Systems using Second–Order Algebraic Methods: A Case Study of the Railroad Crossing Controller

L.J. Steggles
Department of Computer Science, University of Newcastle, UK.
email: L.J.Steggles@newcastle.ac.uk

**Abstract:** Second–order algebraic methods provide a natural and expressive formal framework in which to develop correct computing systems. In this paper we consider using second–order algebraic methods to specify real–time systems and to verify their associated safety and utility properties. We demonstrate our ideas by presenting a detailed case study of the railroad crossing controller, a benchmark example in the real–time systems community. This case study demonstrates how real-time constraints can be modelled naturally using second–order algebras and illustrates the substantial expressive power of second–order equations.

**Keywords**: real–time systems, algebraic specification methods, formal verification.
**Category**: F.3.1, C.3

## 1 Introduction

Standard algebraic specification methods are a well–understood and extensively investigated formal technique for modelling and reasoning about computing systems (see for example [Ehrig and Mahr, 1985] and [Loeckx et al, 1996]). *Second–order algebraic methods* extend the standard techniques by providing explicit support for second–order functions. Second–order algebras provide a very natural way of modelling computing systems based on streams (see [Meinke and Steggles, 1994]) and turn out to be substantially more expressive than their first–order counter parts (see [Kosiuczenko and Meinke, 1995] and [Meinke, 1997]). Second–order algebraic specifications benefit from retaining a simple initial algebra semantics and a simple proof theory based on equational reasoning which is straightforward to automate using rewriting techniques (see [Meinke, 1996a]).

The theory of second–order algebra and their generalisation to higher–order algebras has been developed in a number of papers including: [Maibaum and Lucena, 1980], [Poigné, 1986], [Broy, 1987], [Möller, 1987], and [Qian, 1993]. In this paper we use the finite type theory for second–order algebra developed in [Meinke, 1992]. Despite increasing interest in second–order algebraic methods there is still relatively few case studies to be found in the literature: see [Meinke and Steggles, 1994], [Meinke and Steggles, 1996] and [Steggles, 1995] where a range of dataflow and systolic algorithms are verified correct.

In this paper we consider applying second–order algebraic methods to the formal design of real–time systems. We model the behaviour of entities in time using timed streams, that is elements of the function space $[\mathbf{N} \to A]$, where $\mathbf{N}$ represents discrete time (natural numbers) and $A$ is the set of possible values or states of the entity in question. Thus, for any $a \in [\mathbf{N} \to A]$, and any $t \in \mathbf{N}$, we have that $eval(a, t)$ represents the value of entity $a$ at time $t$ (where $eval : [\mathbf{N} \to A] \times \mathbf{N} \to A$ is the standard evaluation operation). We can view components of a real–time system as simply stream transformers (i.e. second–order functions) of the form $comp : [\mathbf{N} \to A]^n \to [\mathbf{N} \to A]$ which take $n$ timed input streams and produce a timed stream of outputs. Thus a real–time system can be naturally modelled as a second–order algebra

$$(\mathbf{N}, A, [\mathbf{N} \to A];\ eval : [\mathbf{N} \to A] \to A,\ comp : [\mathbf{N} \to A]^n \to [\mathbf{N} \to A]).$$

We present a detailed case study of the specification and verification of a benchmark real–time system. The so called *railroad crossing controller problem* has been widely considered: see for example [Heitmeyer et al, 1993] and [Heitmeyer and Lynch, 1994]. Following a simple refinement methodology we model the safety and utility requirements of the system abstractly making use of a second–order equational encoding of first–order universal quantification. In particular, we identify the environment information for the system which is needed when verifying the systems correctness. We then specify a simple implementation of the crossing controller and verify its correctness against the abstract requirement specification using second–order equational reasoning.

The structure of this paper is as follows. In Section 2 we introduce the basic definitions and theoretical results of second–order algebra. In Section 3 we consider a detailed case study of the benchmark railroad crossing controller. This case study demonstrates our proposed approach to modelling and reasoning about real–time systems and in particular, illustrates the expressive power of second–order equations. Finally in Section 4 we conclude with some general remarks about the ideas introduced in this paper.

## 2 Second–Order Algebraic Methods

In this section we introduce the notion of a second–order signature, algebra and specification, and then consider what it means to correctly specify a second–order algebra. For a detailed introduction to second–order algebraic methods we refer the interested reader to [Meinke, 1992] and [Meinke, 1996a]; for examples of their use see [Meinke and Steggles, 1994], [Meinke and Steggles, 1996] and [Steggles, 1995]. In the sequel we assume that the reader is familiar with basic universal algebraic constructions and results (see [Meinke and Tucker, 1993] and [Loeckx et al, 1996]).

The theory of second–order universal algebra can be developed within the framework of many–sorted first–order universal algebra.

**2.1 Definition** Let $\mathcal{B}$ be any non–empty set, the members of which will be termed *basic types*, the set $\mathcal{B}$ being termed a *type basis.* A *type structure* $S$ over a type basis $\mathcal{B}$ is a set

$$S \subseteq \mathcal{B} \cup \{\ (\sigma \to \tau)\ \mid\ \sigma, \tau \in \mathcal{B}\ \},$$

which is closed under subtypes, i.e. for any type $(\sigma \to \tau) \in S$ we have both $\sigma \in S$ and $\tau \in S$. Each element $(\sigma \to \tau) \in S$ is termed a *second-order* or *function type.*

Given a type structure $S$, an *$S$–typed signature* $\Sigma$ is an $S$–sorted signature such that for each function type $(\sigma \to \tau) \in S$ it includes a distinguished *evaluation operation symbol* $eval^{(\sigma \to \tau)} : (\sigma \to \tau)\ \sigma \to \tau$. □

Note for brevity we often assume that the evaluation function symbol exists for each function type in a second–order signature without explicitly defining them. Next we consider the intended interpretations of a second–order signature $\Sigma$.

**2.2 Definition** Let $A$ be an $S$–sorted $\Sigma$ algebra. We say that $A$ is an *$S$–typed $\Sigma$ algebra* if, and only if, for each function type $(\sigma \to \tau) \in S$ we have (i) $A_{(\sigma \to \tau)} \subseteq [A_\sigma \to A_\tau]$, i.e. $A_{(\sigma \to \tau)}$ is a subset of the set of all (total) functions from $A_\sigma$ to $A_\tau$; and (ii) $eval_A^{(\sigma \to \tau)}: A_{(\sigma \to \tau)} \times A_\sigma \to A_\tau$ is the *evaluation operation* on the function space $A_{(\sigma \to \tau)}$ defined by $eval_A^{(\sigma \to \tau)}(a, n) = a(n)$, for each $a \in A_{(\sigma \to \tau)}$ and $n \in A_\sigma$. □

For brevity given a second–order algebra $A$ we let $a(n)$ denote $eval_A^{(\sigma \to \tau)}(a, n)$, for each function type $(\sigma \to \tau) \in S$, $a \in A_{(\sigma \to \tau)}$ and $n \in A_\sigma$.

Second-order algebras are substantially more expressive than their first–order counter parts and have been shown to be adequate for modelling any algebra of arithmetic complexity, i.e. up to $\Pi_1^1$ (see [Kosiuczenko and Meinke, 1995] and [Meinke, 1995]). This is due to the fact that first–order quantification can be modelled using second–order equations (see [Kosiuczenko and Meinke, 1995]) and we demonstrate this in the case study that follows.

The above definition can be easily extended to allow algebras of arbitrary order, so called *higher–order algebras* (see [Meinke, 1992]). However, in this paper we restrict our attention to second–order algebras since they provide a natural model of stream algebras. It also turns out that increasing the order of algebras above second–order does not increase their expressive power (see [Meinke, 1997]).

The structure of second–order algebras can be characterised by a set of first–order *extensionality sentences* $Ext = Ext_\Sigma$ over $\Sigma$:

$$\forall x \forall y \Big( \forall z \Big( eval^{(\sigma \to \tau)}(x, z) = eval^{(\sigma \to \tau)}(y, z) \Big) \Rightarrow x = y \Big),$$

for each $(\sigma \to \tau) \in S$, where $x, y \in X_{(\sigma \to \tau)}$, $z \in X_\sigma$. A $\Sigma$ algebra $A$ is *extensional* if, and only if, $A \models Ext$. We let $Alg_{Ext}(\Sigma)$ denote the class of all extensional $\Sigma$ algebras. Recall that a $\Sigma$ algebra $A$ is *minimal* if, and only if, $A$ has no proper subalgebra. We let $Min_{Ext}(\Sigma)$ denote the class of all minimal, extensional $\Sigma$ algebras. Let $S_2$ be a type structure such that $S_2 \subseteq S$, and let $\Sigma^2$ be an $S_2$–typed signature such that $\Sigma^2 \subseteq \Sigma$. Given an $S$-typed $\Sigma$ algebra $A$ we say $A$ is *$\Sigma_2$ minimal* if, and only if, $A|_{\Sigma^2}$ (the $\Sigma^2$ reduct of $A$) is minimal.

We are interested in specifying classes of second–order algebras by means of second–order (conditional) equations, i.e. many-sorted first–order (conditional) equations over a second–order signature $\Sigma$. We let $Eqn(\Sigma, X)$ denote the set of all second–order equations over $\Sigma$ and $X$. Given any $\Sigma$ algebra $A$, we have the usual validity relation $\models$ on an (conditional) equation or set of (conditional) equations. Let $E \subseteq Eqn(\Sigma, X)$ be any set of (second–order) equations over $\Sigma$ and $X$, referred to as a (second–order) equational theory. By a basic result of second–order universal algebra (see [Meinke, 1992]), the *extensional equational* *class* $Min_{Ext}(\Sigma,\, E)$ of all minimal, extensional $\Sigma$ algebras which are models of $E$, admits an initial algebra, denoted $I_{Ext}(\Sigma,\, E)$. We refer to $I_{Ext}(\Sigma,\, E)$ as the *second–order initial model*.

Second–order initial models can be concretely constructed from syntax using a *second–order equational calculus*. This calculus extends the many–sorted first–order equational calculus with additional inference rules for second–order types.

**2.3 Definition** *Second–order equational logic* extends the reflexivity, symmetry, transitivity and substitution rules of first–order equational logic (see [Meinke and Tucker, 1993]) with the following additional inference rules:

(1) *Evaluation rule.* For each function type $(\sigma \to \tau) \in S$, any terms $t_0, t_1 \in T(\Sigma, X)_{(\sigma\to\tau)}$ and any variable symbol $x \in X_\sigma$ not occurring in $t_0$ or $t_1$,

$$\frac{eval^{(\sigma\to\tau)}(t_0, x) = eval^{(\sigma\to\tau)}(t_1, x)}{t_0 = t_1.}$$

(2) $\omega$*–evaluation rule.* For each type $(\sigma \to \tau) \in S$ and any $t_0, t_1 \in T(\Sigma, X)_{(\sigma\to\tau)}$,

$$\frac{\langle eval^{(\sigma\to\tau)}(t_0, t) = eval^{(\sigma\to\tau)}(t_1, t) \ \mid \ t \in T(\Sigma)_\sigma \rangle}{t_0 = t_1.}$$

□

The above evaluation rules encode the extensionality axioms on function types. We let $E \ \vdash_\omega e$ denote the inference relation between equational theories $E$ and equations $e$ with respect to (infinitary) second–order equational logic. We note that *finitary second–order equational logic* in which only the (finitary) evaluation rule (1) above is allowed can be shown to be complete with respect to extensional models (see [Meinke, 1992]). The infinitary $\omega$–evaluation rule is needed to construct the second–order initial model as follows. Define the extensional congruence $\equiv^{E,\omega}$ on $T(\Sigma)$ by $t \equiv^{E,\omega}_\tau t' \Leftrightarrow E \vdash_\omega \ t = t'$ for type $\tau \in S$ and any $t, t' \in T(\Sigma)_\tau$. We let $[t]$ denote the congruence class of any term $t \in T(\Sigma)_\tau$ with respect to $\equiv^{E,\omega}$. Then we have the following result (see [Meinke, 1992]):

**2.4 Theorem** *Let $E$ be an equational theory over an $S$–typed signature $\Sigma$. Then we have $T(\Sigma)/\equiv^{E,\omega} \cong I_{Ext}(\Sigma,\, E)$. Thus $T(\Sigma)/\equiv^{E,\omega}$ is initial in the class $Min_{Ext}(\Sigma,\, E)$.* □

A *second–order algebraic specification* $Spec = (\Sigma(Spec), E(Spec))$ is simply a pair consisting of a second–order signature $\Sigma(Spec)$ and an equational (or conditional equational) theory $E(Spec)$. The *second–order initial algebra semantics* of *Spec* is given by the class $Iso(I_{Ext}(Spec))$ of second–order algebras which are isomorphic to the second–order initial model. Let $A$ be an extensional $\Sigma(Spec)$ algebra. We say that *Spec correctly specifies A under second–order initial algebra semantics* if, and only if, $A \in Iso(I_{Ext}(Spec))$. By Theorem 2.4, to establish that a second–order equational specification *Spec* is correct under second–order initial algebra semantics for an extensional $\Sigma$ algebra $A$, it suffices to show that $T(\Sigma(Spec))/\equiv^{E(Spec),\,\omega} \cong A$.

It should be clear from the definition introduced so far that algebras of streams are simply second–order algebras. In order to specify a full stream space, such as $[\mathbf{N} \to D]$, for some data set $D$, we normally add a stream constant $\hat{a} : (nat \to data)$ for each actual stream $a : \mathbf{N} \to D$. This simple approach based on the method of diagrams can be avoided by using topological methods [Meinke, 1996b] or parameterised second–order specifications [Steggles, 1997].

## 3 Case Study: Railroad Crossing Controller

In this section we demonstrate the use of second–order algebraic methods for specifying and verifying real–time systems by considering a case study of the benchmark railroad crossing controller. The railroad crossing problem is a simple real-time problem concerned with the control of a gate at a railroad crossing. The problem was proposed as a benchmark real-time example in [Heitmeyer et al, 1993]; for a detailed explanation see [Heitmeyer and Lynch, 1994].

Simply stated the problem is to design a control system to operate a gate at a railroad crossing, see figure 1. The crossing consists of three components:

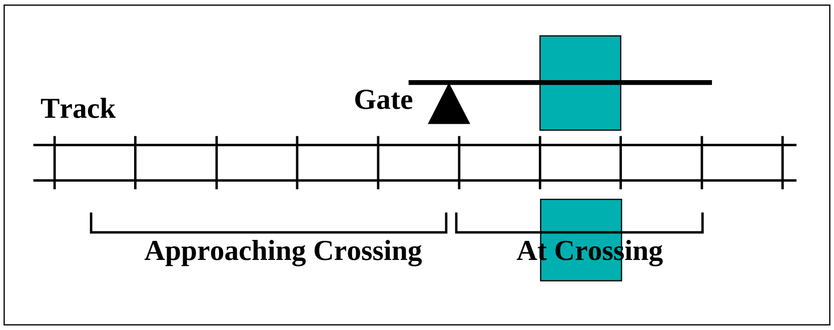


**Figure 1:** The Railroad Crossing.

**Track** We abstractly model the track as being in one of three possible states: *apprch* when a train is approaching the crossing; *atCross* when the train is at the crossing; and *other* when the track approaching and at the crossing is empty. Trains are assumed to have a maximum speed and this allows us to ensure that it takes a minimum amount of time $\epsilon_{app}$ for a train to reach the crossing once it has been detected to be approaching.

**Gate** The gate is a simple barrier that prevents vehicles crossing the track when a train is at (or very near) the crossing. The gate takes commands (either open or close) and then (after a one unit time delay) changes its state accordingly. We assume we know the maximum time to open $\epsilon_{op}$ and close $\epsilon_{cl}$ the gate.

**Controller** The controllers job is to process the track information and send signals to control the operation of the gate. It has to ensure the following two important properties:

(1) **Safety Property**: the gate is closed whenever a train is at the crossing;
(2) **Utility Property**: the gate is open as much as possible.

### 3.1 Requirement Specification Phase

In this phase we construct a second–order equational specification to abstractly specify the safety and utility requirements of the railroad crossing controller. We begin by constructing a *statement signature* which declares the components we are interested in designing (i.e. in this case the gate controller).

**3.1.1 Definition** (Statement Signature) Let $\mathcal{B} = \{Time, Com, TrState\}$ and let $S(Stat) \subseteq H(\mathcal{B})$ be the second-order type structure defined by

$$S(Stat) = \mathcal{B} \ \cup \ \{(Time \to TrState), (Time \to Com)\}.$$

Then we define the *statement specification* $Stat = (\Sigma(Stat), \emptyset)$, where $\Sigma(Stat)$ is an $S(Stat)$–typed signature defined to contain the single function symbol:

$$control : (Time \to TrState) \to (Time \to Com). \qquad \square$$

The symbol *control* represents the controller which takes a timed stream representing the state of the track and returns a stream of gate commands. Next we consider the environment in which the controller will operate.

**3.1.2 Definition** Let $\mathcal{B} = \{Time, Bool, Com, TrState, GState\}$ be a type basis and let $S(Env) \subseteq H(\mathcal{B})$ be the type structure defined by

$$S(Env) = \mathcal{B} \ \cup \ S(Stat) \ \cup \ \{(Time \to Bool), (Time \to GState)\}.$$

Define the $S(Env)$–typed signature $\Sigma(Env)$ to contain the following: first we have symbols needed for modelling discrete time and the Booleans:

$$0 : Time; \ \ tk : Time \to Time; \ \ true, false : Bool;$$

$$or, and : Bool \ \ Bool \to Bool; \ \ less : Time \ \ Time \to Bool;$$

then we have stream constants to represent the function spaces: for each function type $(\sigma \to \tau) \in S(Env)$, each $a \in EN_{(\sigma \to \tau)}$ (see Definition 3.1.3 below), we have $\hat{a} : (\sigma \to \tau)$; next we have symbols to represent the track and gate:

$$MAXop, MAXcl, MINapp : Time; \ \ open, close : Com;$$

$$other, apprch, atCross : TrState; \ \ TRUE : (Time \to Bool);$$

$$tail : (Time \to Bool) \to (Time \to Bool); \ \ eq : TrState \ \ TrState \to Bool;$$

$$interval : Time \ \ Time \ \ TrState \ \ (Time \to TrState) \to Bool;$$

$$caseV : TrState \ \ Time \ \ (Time \to TrState) \to Bool;$$

$$valid : (Time \to TrState) \to (Time \to Bool); \ \ fold : (Time \to Bool) \to Bool;$$

$$normal : (Time \to TrState) \to Bool; \ \ down, up : Time \to GState;$$

$$caseG : Com \ \ GState \to GState;$$

$$gate : (Time \to Com) \to (Time \to GState). \qquad \square$$

We now introduce a concrete model of the signature $\Sigma(Env)$ which represents out intuitive interpretation of the environment signature.

**3.1.3 Definition** Let $EN$ be the $S(Env)$–typed $\Sigma(Env)$ algebra defined as follows. Define the carrier sets

$$EN_{Time} = \mathbf{N},\ EN_{Bool} = \mathbf{B},\ EN_{Com} = \{op, cl\},$$

$$EN_{TrState} = \{A, C, O\},\ EN_{GState} = \{u(i) \mid i \in \mathbf{N}\} \cup \{d(i) \mid i \in \mathbf{N}\},$$

and for each function type $(Time \to \tau) \in S(Env)$ define $EN_{(Time \to \tau)} = [\mathbf{N} \to EN_\tau]$. We now have to define how each function symbol is interpreted in $EN$. In fact this is straightforward to do and for brevity we present only the definition of the main gate function symbol $gate_{EN} : [\mathbf{N} \to EN_{Com}] \to [\mathbf{N} \to EN_{GState}]$ on any command stream $cs \in [\mathbf{N} \to EN_{Com}]$ and $t \in \mathbf{N}$ by

$$gate_{EN}(cs)(t) = \begin{cases} u(0), & \text{if } t = 0; \\ caseG_{EN}(cs(t-1), gate_{EN}(cs)(t-1)), & \text{otherwise.} \end{cases}$$

□

In the sequel it is useful to have a term representation for the data elements in $EN$. For this reason we define the following mapping.

**3.1.4 Definition** Define the term mapping $\overline{\cdot} : EN \to T(\Sigma(Env))$ by

$$\overline{n} = tk^n(0),\quad \overline{b} = \begin{cases} true, & \text{if } b = tt; \\ false, & \text{otherwise,} \end{cases} \quad \overline{c} = \begin{cases} open, & \text{if } c = op; \\ close, & \text{otherwise,} \end{cases}$$

$$\overline{tr} = \begin{cases} other, & \text{if } tr = O; \\ apprch, & \text{if } tr = A; \\ atCross, & \text{otherwise,} \end{cases} \quad \overline{g} = \begin{cases} up(\overline{i}), & \text{if } g = u(i); \\ down(\overline{i}), & \text{if } g = d(i); \end{cases} \quad \overline{a} = \hat{a},$$

for $n \in \mathbf{N}$, $b \in \mathbf{B}$, $c \in EN_{Com}$, $tr \in EN_{TrState}$, and $g \in EN_{GState}$, for each $(\sigma \to \tau) \in S(Env)$ and $a \in EN_{(\sigma \to \tau)}$. □

We now specify the environment by constructing a second–order equational specification which we will prove correctly specifies our standard model $EN$.

**3.1.5 Definition** (Environment Specification) Define the second–order equational specification $Env$ by

$$Env = (\Sigma(Env), E(Env)),$$

where $\Sigma(Env)$ is the $S(Env)$–typed signature defined in Definition 3.1.2 and $E(Env)$ is the second–order equational theory defined over $\Sigma(Env)$ and an $S(Env)$–indexed family $X$ of sets of variables as follows.

$$MAXop = tk^{\epsilon_{op}}(0),\ MAXcl = tk^{\epsilon_{cl}}(0),\ MINapp = tk^{\epsilon_{app}}(0), \qquad (1a, b, c)$$

$$TRUE(t) = true,\ \hat{a}(\overline{n}) = \overline{a(n)}, \qquad (2, 3)$$

for each $(\sigma \to \tau) \in S(Env)$, $a \in EN_{(\sigma\to\tau)}$ and $n \in EN_\sigma$;

$$interval(t, 0, s, tr) = true, \tag{4a}$$

$$interval(t, tk(t'), s, tr) = (interval(tk(t), t', s, tr)\ and\ eq(s, tr(t))), \tag{4b}$$

$$caseV(other, t, tr) =$$
$$(eq(tr(tk(t)), other)\ or\ interval(tk(t), MINapp, apprch, tr)) \tag{5a}$$

$$caseV(apprch, t, tr) = (eq(tr(tk(t)), apprch)\ or\ eq(tr(tk(t)), atCross)) \tag{5b}$$

$$caseV(atCross, t, tr) = (eq(tr(tk(t)), atCross)\ or\ eq(tr(tk(t)), other)) \tag{5c}$$

$$valid(tr)(0) = eq(tr(0), other),\ \ valid(tr)(tk(t)) = caseV(tr(t), t, tr), \tag{6a, b}$$

$$tail(bst)(t) = bst(tk(t)),\ \ normal(tr) = fold(valid(tr)), \tag{7, 8}$$

$$fold(bst) = (bst(0)\ and\ fold(tail(bst))),\ \ fold(TRUE) = true, \tag{9a, b}$$

$$caseG(open, up(0)) = up(0),\ \ caseG(open, down(t)) = up(MAXop), \tag{10a, b}$$

$$caseG(open, up(tk(t))) = up(t),\ \ caseG(close, down(0)) = down(0), \tag{10c, d}$$

$$caseG(close, down(tk(t))) = down(t), \tag{10e}$$

$$caseG(close, up(t)) = down(MAXcl), \tag{10f}$$

$$gate(c)(0) = up(0),\ \ gate(c)(tk(t)) = caseG(c(t), gate(c)(t)), \tag{11a, b}$$

$$less(MAXop, MINapp) = true. \tag{12}$$

where $t, t' \in X_{Time}$, $b, b1, b2 \in X_{Bool}$, $s \in X_{TrState}$, $tr \in X_{(Time\to TrState)}$, $x \in X_{GState}$, $bst \in X_{(Time\to Bool)}$, and $c \in X_{(Time\to Com)}$. Note for brevity we have omitted the standard equations for *or*, *and*, *less* and the equality function *eq*. □

The function *valid* is used to axiomatise the correct behaviour of a rail track. Note that $normal(tr)$ is used to model a universally quantified sentence: $\forall t : Time\ .\ valid(tr)(t) = true$. This is an example of the power of second–order equations; using the auxiliary operation *fold* we have been able to axiomatise first–order universal quantification using second–order equations. For a detailed discussion of this point see [Kosuiczenko and Meinke, 1995].

We now need to ensure that the specification *Env* is consistent. We do this by proving that *EN* is a model of the specification *Env*.

**3.1.6 Proposition** (Consistent) $EN \models E(Env)$. □

We now show that *Env* correctly specifies the environment information as defined by *EN*.

**3.1.7 Theorem** (Correctness) *The second–order equational specification Env correctly specifies the standard model EN under second–order initial algebra semantics, i.e.* $EN \cong I_{Ext}(Env)$.

**Proof.** Since we can easily show that $I_{Ext}(Env)$ and $EN$ are both minimal extensional $\Sigma(Env)$, $E(Env)$ algebras and since $I_{Ext}(Env)$ is initial in the class of all minimal $\Sigma(Env)$, $E(Env)$ algebras it suffices to show there exists an homomorphism $\phi : EN \to I_{Ext}(Env)$. Define the family of mappings

$$\phi = \langle\ \phi : EN_\tau \to I_{Ext}(Env)_\tau \ \mid \tau \in S(Env)\ \rangle,$$

by $\phi_\tau(a) = [\overline{a}]$, for each $\tau \in S(Env)$ and each $a \in EN_\tau$. Then it is straightforward to show that $\phi$ is a homomorphism. As an example consider the function symbol $gate : (Time \to Com) \to (Time \to GState)$. We have to show that $\phi(gate_{EN}(a)) = gate_{I_{Ext}(Env)}(\phi(a))$, for any $a \in Env_{(Time \to Com)}$. Since it can be easily shown that each term of type *Time* is provably equivalent to a term of form $tk^i(0)$, for some $i \in \mathbf{N}$, it suffices to prove that for any $n \in \mathbf{N}$

$$\phi(gate_{EN}(a))([tk^n(0)]) = gate_{I_{Ext}(Env)}(\phi(a))([tk^n(0)]), \tag{I}$$

and then apply the infinitary $\omega$–evaluation rule. It is straightforward to show that (I) holds using induction on $n \in \mathbf{N}$. $\square$

We are now in a position to define the abstract requirement specification for the gate controller using the statement and environment specifications.

**3.1.8 Definition** (Requirement Specification) Define the requirement specification

$$Req = (\Sigma(Req), E(Req)),$$

as follows. Let $S(Req) = S(Env)$ and define $\Sigma(Req) = \Sigma(Stat) \cup \Sigma(Env)$. Let the second–order equational theory $E(Req)$ consist of the equations in $E(Env)$ (see Definition 3.1.5 above) and the following second–order conditional equations: first a second–order conditional equation representing the *safety property*:

$$(normal(tr)\ and\ eq(tr(t), atCross)) = true \Longrightarrow$$

$$gate(control(tr))(t) = down(0); \tag{13}$$

finally a second–order conditional equation representing the *utility property*:

$$(normal(tr)\ and\ interval(t, tk(MAXop), other, tr)) = true \Longrightarrow$$

$$gate(control(tr))(tk^{\epsilon_{op}+1}(t)) = up(0); \tag{14}$$

where $t \in X_{Time}$ and $tr \in X_{(Time \to TrState)}$. $\square$

### 3.2 Design Specification Phase

We now consider a simple design for the railroad gate controller and construct a second–order equational specification *Des* (extending the statement specifcation) which specifies this design.

**3.2.1 Definition** Let $S(Des)$ be the type structure defined in Definition 3.1.1, i.e. $S(Des) = S(Stat)$. Define the $S(Des)$–typed signature $\Sigma(Des)$ to extend the signature $\Sigma(Stat)$ with the following constant and function symbols

$$0 : Time;\ \ tk : Time \to Time;\ \ open, close : Com;$$

$$other, apprch, atCross : TrState; \; caseC : TrState \to Com;$$

for each function type $(\sigma \to \tau) \in S(Des)$, each $a \in DC_{(\sigma \to \tau)}$ (see Definition 3.2.2 below), we have the stream constant $\hat{a} : (\sigma \to \tau)$. $\square$

Next we define a second–order model which represents our standard interpretation of the above design components.

**3.2.2 Definition** Let $DC$ be the $S(Stat)$–typed $\Sigma(Des)$ algebra defined as follows. Define the carrier sets

$$DC_{Time} = \mathbf{N}, \; DC_{Com} = \{op, cl\}, \; DC_{TrState} = \{A, C, O\},$$

$$DC_{(Time \to TrState)} = [\mathbf{N} \to DC_{TrState}], \; DC_{(Time \to Com)} = [\mathbf{N} \to DC_{Com}].$$

Define the constants and functions of type $Time$ and $(\sigma \to \tau) \in S(Des)$ in the standard way and define

$$caseC_{DC}(s) = \begin{cases} op, & \text{if } s = O; \\ cl, & \text{otherwise;} \end{cases} \quad control_{DC}(tr)(n) = caseC_{DC}(tr(n)).$$

for any $s \in DC_{TrState}$, $tr \in DC_{(Time \to TrState)}$ and $n \in \mathbf{N}$. $\square$

We now formulate a second–order equational specification which we will show correctly specifies the standard model $DC$ of the crossing controller.

**3.2.3 Definition** (Design Specification) Define the second–order equational specification $Des$ by

$$Des = (\Sigma(Des), E(Des)),$$

where $\Sigma(Des)$ is defined in Definition 3.2.1 and $E(Des)$ is the second–order equational theory defined over $\Sigma(Des)$ and $X$ as follows.

For each $(\sigma \to \tau) \in S(Des)$, each $a \in DC_{(\sigma \to \tau)}$ and each $n \in DC_\sigma$, we have

$$\hat{a}(\overline{n}) = \overline{a(n)}, \; caseC(other) = open, \quad caseC(apprch) = close, \qquad (1, 2a, b)$$

$$caseC(atCross) = close, \quad control(tr)(t) = caseC(tr(t)), \qquad (2c, 3)$$

where $t \in X_{Time}$ and $tr \in X_{(Time \to TrState)}$. $\square$

Again, to ensure that the equational specification $Des$ is consistent we need to show that $DC$ is a model of the specification $Des$.

**3.2.4 Proposition** (Consistent) $\quad DC \models E(Des)$. $\square$

We now show that $Des$ correctly specifies the design model $DC$ as follows.

**3.2.5 Theorem** (Correctness) $\quad DC \cong I_{Ext}(Des)$.

**Proof.** Follows along similar lines to the proof of Theorem 3.1.7. $\square$

### 3.3 Verification Phase

In the preceeding sections we have formulated an abstract requirement specification for the railroad gate controller and its environment, and a design specification for a proposed railroad crossing controller. It remains to verify that the design specification is a correct functional refinement of the requirement specification. We begin by constructing the *verification specification* which extends the design specification with the environment information.

**3.3.1 Definition** Let $Ver$ be the second–order equational specification

$$Ver = (\Sigma(Ver), E(Ver)),$$

where $\Sigma(Ver) = \Sigma(Env) \cup \Sigma(Des)$ and $E(Ver) = E(Env) \cup E(Des)$. □

We need to show that this new verification specification satisfies the so called *consistency*, *preservation* and *refinement* conditions (see [Steggles and Wirsing, 1995]).

**3.3.2 Proposition** (Consistent) *$E(Ver)$ is a consistent equational theory.*

**Proof.** We construct a non–unit $S(Ver)$–typed $\Sigma(Ver)$ algebra $A$ by combining the standard model $EN$ (Definition 3.1.3) and $DC$ (Definition 3.2.2). This is possible since we can see that $EN|_\Sigma = DC|_\Sigma$, for $\Sigma = \Sigma(Env) \cap \Sigma(Des)$. It is then straightforward to show $A \models E(Ver)$. □

In order to be able to reason about the verification specification it is useful to identify its constructors. Let $S(Cons) = S(Ver)$ and $A$ be defined as in Proposition 3.3.2. Define the $S(Cons)$–sorted signature $\Sigma(Cons)$ by

$$0 : Time;\ tk : Time \to Time;\ true, false : Bool,\ open, close : Com,$$

$$other, apprch, atCross : TrState;\ down, up : Time \to GState;\ \hat{a} : (\sigma \to \tau),$$

for each $(\sigma \to \tau) \in S(Ver)$, $a \in A_{(\sigma\to\tau)}$.

We can show that the initial algebra semantics of the verification specification is generated by the above constructor signature.

**3.3.3 Proposition** *$I_{Ext}(Ver)$ is $\Sigma(Cons)$ minimal.*

**Proof.** Straightforward using induction on the construction of $\Sigma(Ver)$ terms. □

It turns out that this is a very useful fact which is used extensively in the verification results that follow. Next we show that the semantics of the verification specification preserves the semantics of the design specification.

**3.3.4 Proposition** (Preservation) $I_{Ext}(Des) \cong I_{Ext}(Ver)|_{\Sigma(Des)}$.

**Proof.** By Proposition 3.3.3 it follows that $I_{Ext}(Ver)|_{\Sigma(Des)} \in Min_{Ext}(Des)$.

Since $I_{Ext}(Des)$ is initial in the class $Min_{Ext}(Des)$ it suffices to show there exists a homomorphism $\phi : I_{Ext}(Ver)|_{\Sigma(Des)} \to I_{Ext}(Des)$. Define the family of mappings

$$\phi = \langle\ \phi_\tau : (I_{Ext}(Ver)|_{\Sigma(Des)})_\tau \to I_{Ext}(Des)_\tau \mid \tau \in S(Des)\ \rangle,$$

by $\phi_\tau([t]) = [t]$, for each $\tau \in S(Des)$ and each $t \in T(\Sigma(Cons))_\tau$. Clearly, $\phi$ satisfies the homomorphism condition. It remains to show that $\phi$ is well–defined, i.e. for any $\tau \in S(Ver)$ and any terms $t, t' \in T(\Sigma(Cons))_\tau$,

$$E(Ver) \models_\omega t = t' \Longrightarrow E(Des) \models_\omega t = t'. \qquad (*)$$

Since we can show that $I_{Ext}(Des)$ is $\Sigma(Cons)$ minimal and since both specifications are consistent it is straightforward to show that $(*)$ must hold. □

Finally, we need to show that $Ver$ is a correct functional refinement of $Req$.

**3.3.5 Theorem** (Refinement) $I_{Ext}(Ver) \models E(Req) - E(Env)$.

**Proof.** We have to show that the *safety* (axiom 3.1.8.(13)) and *utility* (axiom 3.1.8.(14)) axioms hold in $I_{Ext}(Ver)$.

**(1) Safety.** It suffices by $\Sigma(Cons)$–minimality (Proposition 3.3.3) to prove that for each stream constant $a \in [\mathbf{N} \to \{O, A, C\}]$ and each $n \in \mathbf{N}$ we have

$$E(Ver) \models_\omega (normal(\hat{a})\ and\ eq(\hat{a}(\overline{n}), atCross)) = true \Longrightarrow$$

$$gate(control(\hat{a}))(\overline{n}) = down(0).$$

By assumption, $\Sigma(Cons)$–minimality and equational reasoning we can show using a proof by contradiction that there must exist a $k \in \mathbf{N}$, $0 < k < n$ such that $E(Ver) \vdash_\omega eq(\hat{a}(\overline{n-k}), apprch) = true$, and for $i = 0, \ldots, k-1$, $E(Ver) \models_\omega eq(\hat{a}(\overline{n-i}), atCross) = true$. We can now prove that the result holds using induction on $k \in \mathbf{N}$, $k > 0$.

**(2) Utility.** We have to prove that for each $a \in [\mathbf{N} \to \{O, A, C\}]$, $n \in \mathbf{N}$,

$$E(Ver) \models_\omega (normal(\hat{a})\ and\ interval(\overline{n}, tk(MAXop), other, \hat{a})) = true \Longrightarrow$$

$$gate(control(tr))(\overline{n + \epsilon_{op} + 1}) = up(0).$$

By $\Sigma(Cons)$–minimality, assumption and equations for *interval* we have that $E(Ver) \models_\omega \hat{a}(\overline{n+i}) = other$, for $i = 0, \ldots, \epsilon_{op}$. We can then prove by induction on $n \in \mathbf{N}$ that the result holds. □

## 4 Conclusions

In this paper we have considered using second–order algebraic methods for formally developing correct real–time systems. In particular, we have presented a detailed case study of applying second–order algebraic methods to the specification and verification of the benchmark railroad crossing controller problem. This case study illustrates that second–order techniques provide a natural framework for modelling and reasoning about real–time systems based on timed streams. It also illustrated the substantial expressive power of second–order equations and we saw an example of how we can capture the notion of first–order quantification within a second–order equational environment.

In future work we intend to consider automating verification proofs in second–order equational logic and aim to construct a range of new tools based on advanced term rewriting systems such as Elan (see [Borovanský et al, 1998]).

We would like to acknowledge the helpful advice and comments provided by Dr K. Meinke during the preparation of this paper. We are also very grateful for the financial assistance provided by the British Council and DAAD.

## References

[Broy, 1987] Broy, M.: Equational specification of partial higher–order algebras. In: M. Broy (ed) *Logic of programming and calculi of discrete design*, Springer-Verlag (1987).

[Borovanský et al, 1998] Borovanský, P., Kirchner, C., Kirchner, H., Moreau, P.–E., and Ringeissen, C.: An overview of ELAN. In: C. Kirchner and H. Kirchner (eds), Proceedings of WRLA '98, *Electronic Notes in Theoretical Computer Science*, 15 (1998).

[Ehrig and Mahr, 1985] Ehrig, H. and Mahr, B.: *Fundamentals of Algebraic Specification 1 – Equations and Initial Semantics.* EATCS Monographs on Theoretical Computer Science 6, Springer-Verlag, Berlin (1985).

[Heitmeyer et al, 1993] Heitmeyer, C., Jeffords, R. D., and Labaw, B. G.: A benchmark for comparing different approaches for specifying and verifying real-time systems. In: *Proc. of 10th Int. Workshop on Real–Time Operating Systems and Software* (1993).

[Heitmeyer and Lynch, 1994] Heitmeyer, C., and Lynch, N.: The Generalized Railroad Crossing: A Case Study in Formal Verification of Real–Time Systems. In: *Proc. of IEEE Real–Time Systems Symposium*, San Juan, Puerto Rica, December (1994).

[Kosiuczenko and Meinke, 1995] Kosiuczenko, P., and Meinke, K.: On the power of higher-order algebraic specifications. *Information and Computation* 124(1):85-101 (1995).

[Loeckx et al, 1996] Loeckx, J., Ehrich, H–D., and Wolf, M.: *Specification of Abstract Data Types*, Wiley (1996).

[Maibaum and Lucena, 1980] Maibaum, T. S. E., and Lucena, C. J.: Higher–order data types. *International Journal of Computer and Information Sciences*, 9:31–53, 1980.

[Meinke, 1992] Meinke, K.: Universal algebra in higher types. *Theoretical Computer Science*, 100:385–417 (1992).

[Meinke, 1996a] Meinke, K.: Higher-order equational logic for specification, simulation and testing. In: G. Dowek et al (eds), *Proc. of HOA '95*, LNCS 1072, Springer–Verlag (1996), 124–143.

[Meinke, 1996b] Meinke, K.: Topological methods for algebraic specification. *Theoretical Computer Science*, 166:263–290 (1996).

[Meinke, 1997] Meinke, K.: A completeness theorem for the expressive power of higher-order algebraic specifications. *Journal of Computer and System Sciences* (1997).

[Meinke and Tucker, 1993] Meinke, K. and Tucker, J. V.: Universal algebra. In: S. Abramsky, D. Gabbay and T.S.E. Maibaum, (eds) *Handbook of Logic in Computer Science*, Volume I, Oxford University Press, Oxford (1993), 189–412.

[Meinke and Steggles, 1994] Meinke, K. and Steggles, L. J.: Specification and Verification in Higher Order Algebra: A Case Study of Convolution. In: J. Heering et al (eds), *Proc. of HOA '93*, LNCS 816, Springer–Verlag (1994), 189-222.

[Meinke and Steggles, 1996] Meinke, K. and Steggles, L. J.: Correctness of Dataflow and Systolic Algorithms: Case Studies in Higher–Order Algebra. Technical Report No. 559, Department of Computing Science, University of Newcastle (1996).

[Möller, 1987] Möller, B.: Algebraic specifications with higher–order operators. In: L.G.L.T. Meertens (ed), *Program specification and transformation*, North–Holland (1987).

[Poigné, 1986] Poigné, A.: On specifications, theories and models with higher types. *Information and Control*, 68:1–46 (1986).

[Qian, 1993] Qian, Z.: An Algebraic Semantics of Higher–Order Types with Subtypes. *Acta Informatica*, 30:569–607 (1993).

[Steggles, 1995] Steggles, L. J.: *Extensions of Higher–Order Algebra: Fundamental Theory and Case Studies.* Ph.D. Thesis, University of Wales, Swansea (1995).

[Steggles, 1997] Steggles, L. J.: Parameterised Higher–Order Algebraic Specifications. In: M. Hanus et al (eds), *Proc. of ALP/HOA '97*, LNCS 1298, Springer–Verlag (1997), 76–98.

[Steggles and Wirsing, 1995] Steggles, L. J. and Wirsing, M. Formal Software and Hardware Development: A Case Study in the use of CSDM, SPECTRUM and HOLCF. Technical Report No. 9515, Ludwig–Maximilians–Universität München, Institut für Informatik (1995).