Controlling control systems: an application of evolving retrenchment

Michael Poppleton[1] and Richard Banach[2]

[1] Department of Computing, Open University, Walton Hall,
Milton Keynes MK7 6AA, UK,
m.r.poppleton@open.ac.uk,
WWW home page: http://mcs.open.ac.uk/mp529

[2] Department of Computer Science, Manchester University,
Manchester M13 9PL, UK,
banach@cs.man.ac.uk,
WWW home page: http://www.cs.man.ac.uk/~banach/

Abstract. We review retrenchment as a liberalisation of refinement, for the description of applications too rich (e.g. using continuous and infinite types) for refinement. A specialisation of the notion, *evolving retrenchment* is introduced, motivated by the need for an approximate, evolving notion of simulation. The focus of the paper is the case study, a substantial second-order linear control system. The design step from continuous to zero-order hold discrete system is expressible as an evolving retrenchment. Thus we demonstrate that the retrenchment approach can formalise the development of useful applications, which are outside the scope of refinement.
The work is presented in a data type-enriched language containing the B language of J.-R. Abrial.

1 Introduction

From early concerns about proving correctness of programs such as Hoare's [23] and Dijkstra's [15], a mature *refinement calculus* of specifications to programs has developed. Thorough contemporary discussion can be found in [14, 2]. In particular, the *simulation* proof method, appearing in *inter alia* [34, 32], is central.

In this context of model-based specifications the term "refinement" has a very precise meaning; according to Back and Butler [3] it is a "...correctness-preserving transformation...between (possibly abstract, non-executable) programs which is transitive, thus supporting stepwise refinement, and is monotonic with respect to program constructors, thus supporting piecewise refinement." Relationally, refinement is characterised as a development step requiring the concrete precondition to be weaker than the abstract (the applicability, or termination condition), and the concrete transition relation to be stronger, or less nondeterministic, than the abstract (the correctness, or transition condition). The most succinct characterisation of refinement is that of "operational indistinguishability", i.e. that every concrete behaviour be a possible abstract one.

Refinement is a strong technique in software development, both in descriptive power, and in delivering proof obligations that assert strongly coupled structure between levels of abstraction. So it is not surprising that refinement cannot be used, without simplifying or approximating assumptions and informal justifications, in many real-world system situations. Early work on clean termination of programs [11, 10] shares our concern with refining specifications on abstract, infinite domains to finite computer-oriented domains. Another approach to this finiteness problem was Neilson's thesis [24], which proposed a notion of acceptably inadequate design, i.e. that refinement over an infinite domain could be regarded as a limit of finite refinements. Partial logic approaches have also been proposed [26].

This work develops and applies the *retrenchment* method, a liberalisation of refinement. We argued, when introducing the notion [5, 6], for a weakening of the retrieve relation over the operation step, allowing concrete non-simulating behaviour in retrenchment. Concrete I/O may have different type to the abstract counterpart, and moreover the retrenchment relation may define fluidity between state and I/O components across the development step from abstract to concrete model. This early work made a more engineering-oriented than formal case for moving away from refinement.

[29] reported initial work, using transitivity and monotonicity arguments, on the development of a calculus of retrenchment in B. This calculus was completed in [31], which showed all primitive operators of the B GSL, including operation sequence, to be monotonic with respect to retrenchment. [8, 9] gave a substantial theoretical presentation and development of the retrenchment framework, exploring the landscape between refinement, simulation and retrenchment. This was done by means both of B and automata-theoretic formalisms, using specialised (if widely applicable) further assumptions to nudge the retrenchment relation closer to refinement-like structure. [7] examined what can be said when a simulation is partial, or *punctured*, rather than full. This allows the description of breaks, or punctures (e.g. failure and reset) in the simulation. [4] addressed the integration of refinement and retrenchment from a methodological perspective. In [30] we presented a generalisation, *evolving retrenchment*.

This paper is squarely in the B setting (albeit with some data-type-enrichment), and uses the generalisation (rather than specialisation) that is evolving retrenchment. The motivation for this variant of retrenchment is the need for an approximate, evolving notion of simulation. In simple retrenchment, a simulation fails at the point at which an abstract/concrete operation step fails to establish the retrieve relation. Here we propose that the retrieve relation be allowed to *evolve* in the sense that the precision of representation may change over time. We give a simple supporting example of the transformation of real to floating-point addition. As an evolving retrenchment, this transformation demonstrates the approximate nature of the simulation of real by FP addition. It also serves as a formal bound on error propagation, thus giving a link to classical numerical analysis. The introductory example is completed with a demonstration of how the monotonicity of evolving retrenchment with respect to operation sequence supports an evolving notion of simulation.

The focus of this paper is the case study, a substantial second-order linear control system. We summarise the classical control engineering design transformation from continuous- to discrete-time using the zero-order hold approximation. Two evolving retrenchment descriptions are given for this transformation, in a B framework enriched with appropriate data types.

The paper contents are as follows. In Sect. 2 we briefly recap syntactic and semantic definitions for retrenchment. Evolving retrenchment is defined and demonstrated in a small example. Section 3 introduces the case study, a second-order linear control system. This is done as two descriptions, a classical continuous control system, and the zero-order hold discrete approximation. Section 4 gives two possible design formulations in terms of evolving retrenchment. We discuss the limitations of a method such as B for such a formulation. Section 5 concludes.

2 From Simple to Evolving Retrenchment

The notion of an evolving relationship between system models emerges from the intrinsically approximate nature of modelling real-world systems which are described with continuous mathematics. For example, to model the motion of a projectile near to the earth we might use a linear differential equation relating the acceleration of the projectile to the forces of gravity and air friction. We might choose to omit, for practical reasons, the Coriolis effect, or the effect of time-varying crosswinds. So the best mathematical model available to us will always be approximate, for reasons of cost, practicality, or even lack of mathematical sophistication.

This work is, however, concerned with the relationship between mathematical models at adjacent levels of abstraction. In the projectile example we might (unwisely choosing a large reduction in abstraction level) choose to specify the next layer of model using some finite set of floating-point numbers appropriate to computer implementation. Clearly, a second level of approximation is introduced: before any system dynamics are considered, precision is limited to that determined by the floating-point set chosen. Moreover, when some abstract (continuous) operation is considered in relation to its concrete (floating point) counterpart, precision of representation is in general reduced. For simple addition, precision decays by a factor of two. For a more complex operation and its retrenchment, say exponentiation, the precision decay is far more complex.

From a refinement point of view this suggests that relaxing the invariant nature of the retrieve relation between abstract and concrete models, allowing it to *evolve*, might give a more expressive development step than either refinement or simple retrenchment. Although retrenchment allows breach of the invariant (via establishment of the concession), such breach effectively stops any simulation of the abstract operation sequence by the concrete. Specialisations of retrenchment, as indicated before, seek to combine the concession achieved with application domain structure to re-establish the retrieve relation where the simulation has failed to establish it. The approach here is more general in relaxing the immutability of the retrieve relation, and allowing it to evolve. That is, if one operation step results under retrenchment in achievement of a weaker, evolved retrieve relation, this latter relation can serve as a starting point for a subsequent retrenchment step.

2.1 Simple Retrenchment

Simple retrenchment is, loosely speaking, the strengthening of the precondition, the weakening of the postcondition, and the introduction of mutability between state and I/O at the two levels of abstraction.

Figure 1 defines the B syntax. Abstract machine M has parameter a, state variable u, and invariant predicate $I(u)$. Variable u is initialised by substitution $X(u)$, and is operated on by operation *OpName*, a syntactic wrapper for substitution $S(u, i, o)$, with input i and output o. Unlike a refinement, which in B is a construct derived from the refined machine, a retrenchment is an independent MACHINE. Thus N is a machine with parameter b (not necessarily related to a), state variable v, invariant $J(v)$, initialisation $Y(v)$, and operation *OpNameC* as wrapper for $T(v, j, p)$, a substitution with input j and output p. Viewed as an independent machine, N cannot refer to M. So the new retrenchment syntax proposed here must in this case be regarded as null text.

Here we state that machine N RETRENCHES machine M; in general either a MACHINE or a REFINEMENT may be retrenched. The RETRENCHES clause (similarly to REFINES) makes visible the contents of the retrenched construct. We further assume that the name spaces of the retrenched and retrenching constructs are disjoint, but admit an injection of (retrenched to retrenching) operation names. An *injection* allows further, independent dynamic structure in the retrenching machine. In the retrenching machine N we have state variable v under invariant $J(v)$, and the RETRIEVES clause $G(u, v)$. The existence of N as an independent machine requires that its (local) invariant be stated independently of the retrieve relation, unlike in B, where the two take the form of a joint predicate.

The relationship between concrete and abstract state is fundamentally different *before* and *after* the operation. We model this by distinguishing between a strengthened before-relation between abstract and concrete states, and a weakened after-relation. Thus the syntax of the concrete operation *OpNameC* in N is precisely as in B, with the addition of the *ramification*, a syntactic enclosure of the operation. We call this augmented syntax for the operation a *ramified generalised substitution.* Before, the relationship is constrained (precondition is strengthened) by the new *WITHIN* condition $P(u, v, i, j, A)$ which may change the balance of components between input and state, and may further define an optional "memory" variable LVAR A for reference in the after-state. The after-relation is weakened (postcondition is weakened) by the CONCEDES clause (the *concession*) $C(u, v, o, p, A)$, which specifies what the operation guarantees to achieve (in terms of after-state, output and logical variable A) if it cannot

```
MACHINE          $M(a)$           MACHINE          $N(b)$
                                  RETRENCHES       $M$
VARIABLES        $u$              VARIABLES        $v$
INVARIANT        $I(u)$           INVARIANT        $J(v)$
                                  RETRIEVES        $G(u, v)$
INITIALISATION   $X(u)$           INITIALISATION   $Y(v)$
OPERATIONS                        OPERATIONS
    $o \longleftarrow OpName(i) \widehat{=}$        $p \longleftarrow OpNameC(j) \widehat{=}$
        $S(u, i, o)$                    BEGIN
END                                         $T(v, j, p)$
                                        [LVAR
                                            $A$]
                                        WITHIN
                                            $P(i, j, u, v, A)$
                                        CONCEDES
                                            $C(u, v, o, p, A)$
                                        END
                                  END
```

Fig. 1. Syntax of simple retrenchment

maintain the retrieve relation G. The concession will describe the weaker state of representation after the operation, or preserve exception information if representation fails completely.

The operation retrenchment POB is central in this theory since it serves as the semantic definition of retrenchment. Analogously to the operation refinement POB, although with more predicate information, concrete *OpNameC* (substitution T) retrenches abstract operation *OpName* (substitution S) if:

$$I(u) \wedge G(u, v) \wedge J(v) \wedge P(i, j, u, v, A) \wedge \mathsf{trm}(T(v, j, p)) \\ \Rightarrow \mathsf{trm}(S(u, i, o)) \wedge [T(v, j, p)]\neg\, [S(u, i, o)]\neg\, (G(u, v) \vee C(u, v, o, p, A)) \quad (1)$$

The retrenchment initialisation POB is as for refinement, guaranteeing a joint starting state satisfying $G(u, v)$.

Definition (1) is justified by comparison with the refinement proof obligation. In refinement, "operational indistinguishability" of the refined operation is the central issue. When the abstract operation terminates, so does the concrete one: this is precondition weakening. In retrenchment we invert this relationship. The retrenching machine is different from the retrenched machine and more faithful to the system being built. It can contain features incompatible with those of the abstract machine. These are described explicitly in the CONCEDES clause, and implicitly in the WITHIN clause which restricts the portions of the two models being related. To the extent that the retrenching machine might not terminate in places that the retrenched machine is guaranteed to do so, we must consciously exclude such possibilities in the WITHIN clause from the relationship asserted between the two machines. Thus we strengthen the precondition by swapping the two termination clauses in the POB, and use the WITHIN clause to tailor the scope of the POB over and above what is already stated in the RETRIEVES clause.

2.2 Evolving Retrenchment

We now propose that the retrieve relation G becomes "variant" (i.e. evolves), in the sense that it becomes mediated by some "precision" parameters α and β. That is, $G_\alpha(u, v)$ is required to start the retrenchment step, and $G_\beta(u', v')$ is a possible outcome. The intuition behind this is that α, β belong to some ordered set, where increasing α denotes decreasing precision of the representation of abstract u by concrete v in $G_\alpha(u, v)$. That is, we will usually (but not always) expect that

$$\alpha \leq \beta \Rrightarrow (G_\alpha \Rightarrow G_\beta)$$

This formulation describes a typical precision-decay situation over a simulation, for example in a sequence of arithmetic steps (real to floating point). We choose the notion of *evolving* rather than *decaying* retrenchment as this suggests, because it is quite possible for precision to *increase* over a simulation step. In a control system, sensor readings provide the software with the inputs necessary to model the real system state. In control systems with a large number of sensors, it is quite likely that, at a given point in time, some sensors have failed, and that the modelled state is representing the real system state in some degraded mode[1]. Some algorithm extrapolating the failed sensor value from neighbouring sensors may be applied, to make the best of things, but accuracy is nonetheless lost. Assuming it is possible for sensors to be repaired in-flight, and for repair status to be detectable, it is manifestly possible for precision of the system state representation to improve.

The extension to syntax for evolving retrenchment involves some subscripting of clauses G, P, C in Fig. 1. Semantically, *evolving retrenchment* is defined:

$$\begin{aligned} &I(u) \wedge G_\alpha(u,v) \wedge J(v) \wedge P_\alpha(i,j,u,v,A) \wedge \mathsf{trm}(T)(v,j) \wedge \alpha \leq \beta \\ &\Rightarrow \mathsf{trm}(S)(u,i) \wedge [T(v,j,p)]\neg\, [S(u,i,o)]\neg\, (G_\beta(u,v) \vee C_\beta(u,v,o,p,A)) \end{aligned} \quad (2)$$

The following shorthand form will be more convenient:

$$S \lesssim T \;\; \text{w.r.t.} \;\; (G_\alpha, P_\alpha \longrightarrow G_\beta, C_\beta)$$

Thus the (u, v) relationship is now mediated through evolving precision parameters α and β. Notice that WITHIN and CONCEDES predicates are also parameterised: this is necessary since each of these further constrain the (u, v)

[1] This assumes reliable sensor failure determination, an issue we will not pursue here.

relationship, in context of inputs and outputs respectively. Generalising these predicates also allows these relations between abstract/concrete I/O and state to evolve. Note that the precision parameters are not free variables, they are more akin to B machine parameters. A relation of evolving retrenchment between two machines M and $N(\alpha,\beta)$ can be interpreted as a collection of simple retrenchments indexed on the carrier sets of precision parameters α,β. This collection describes the (usually decaying) evolution of the RETRIEVES relationships involved at each step.

2.3 From an Example to Simulation

A simple arithmetic example will demonstrate the use of evolving retrenchment. Floating point arithmetic is well understood, e.g. [19]. We choose the retrenchment of real to floating-point addition, simplifying the scenario to highlight the evolving approximation of representation that we wish to describe. Figure 2 shows the example: the upper line of numbers represents a sequence of three real additions and the lower line represents the corresponding floating point sequence. Δ represents the error bound at each step.

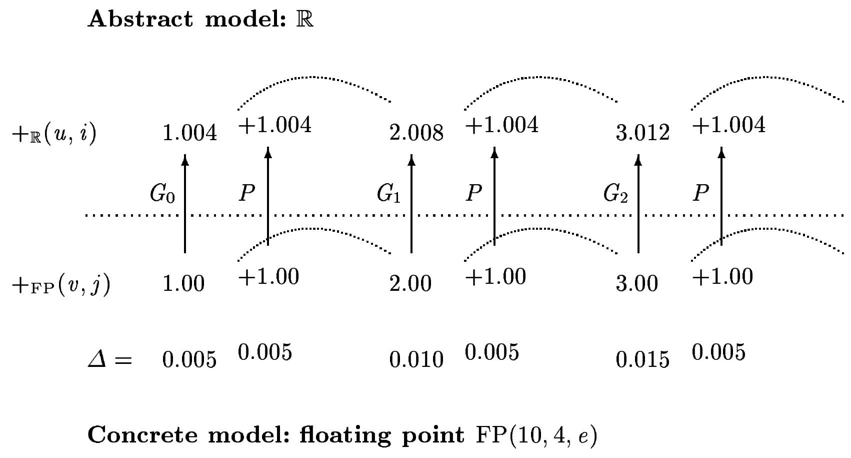

Fig. 2. Evolving retrenchment of real to floating point addition

We restrict ourselves, using the WITHIN clause, to (abstract state and input variables u,i) the real interval $[1, 9.9995)^2$, and the corresponding set of base-ten floating point numbers with four decimal places (concrete state and input variables v,j). e is the limiting exponent value for FP representation. We use a

[2] This overloaded syntax is the classical real notation for a semi-open interval including 1 and excluding 9.9995

rounding regime for FP addition. We have (using some syntactic shorthand for clause P):

$$
\begin{aligned}
& I(u) \mathrel{\widehat{=}} u \in \mathbb{R} \qquad\qquad J(v) \mathrel{\widehat{=}} v \in \mathrm{FP}(10, 4, e) \\
& OpA(u, i) \mathrel{\widehat{=}} u := +_{\mathbb{R}}(u, i) \qquad\qquad OpC(v, j) \mathrel{\widehat{=}} v := +_{\mathrm{FP}}(v, j) \\
& G_k(u, v) \mathrel{\widehat{=}} \mid u - v \mid \leq 5 * (k+1) * 10^{-3} \text{ for } k \in \mathbb{N} \\
& P(i, j, u, v) \mathrel{\widehat{=}} u, i, u + i \in [1, 9.9995) \\
& \qquad\qquad\qquad \wedge\ v, j \geq 1 \wedge v, j \leq 9.999 \wedge \mid i - j \mid \leq 5 * 10^{-3} \\
& C_k(u, v) \mathrel{\widehat{=}} G_{k+1}(u, v)
\end{aligned}
\tag{3}
$$

Because these are simple assignment operations, the operations terminate universally. Initial error G_0 is the standard state representation error $5 * 10^{-3}$ for these FP parameters; the evolving error results from the accumulation of the constant input representation error $5 * 10^{-3}$. So the evolving retrenchment POB is a succinct formal statement that the retrieve relation will degrade by at most $5 * 10^{-3}$ every step:

$$
\begin{aligned}
& I(u) \wedge G_k(u, v) \wedge J(v) \wedge P(i, j, u, v) \\
& \Rightarrow [+_{\mathrm{FP}}(v, j)]\neg\ [+_{\mathbb{R}}(u, i)]\neg\ (G_k(u, v) \vee C_k(u, v))
\end{aligned}
\tag{4}
$$

The apparently curious concession definition is explained by the POB: the last subclause could more clearly be written $(G_k(u, v) \vee G_{k+1}(u, v))$.

From [30], we repeat the monotonicity statement for operation sequence with respect to evolving retrenchment. This is a simpler form for universally substituting operations:

$$
\begin{aligned}
& S1 \lesssim T1 \ \text{ w.r.t. } \ (G_\alpha, P1_\alpha(i_1, j_1, u, v) \longrightarrow G_\beta, C1_\beta(u, v, o_1, p_1)) \\
& \wedge\ S2 \lesssim T2 \ \text{ w.r.t. } \ (G_\beta, P2_\beta(i_2, j_2, u, v) \longrightarrow G_\gamma, C2_\gamma(u, v, o_2, p_2)) \\
& \wedge\ I \wedge G_\alpha \wedge J \\
& \wedge\ P1 \wedge [T1]\neg\ [S1]\neg\ (G_\beta \wedge P2_\beta) \\
& \vdash [T1; T2]\neg\ [S1; S2]\neg\ (G_\gamma \vee C2_\gamma) \\
& \qquad\qquad\qquad \text{...that is} \\
& S1; S2 \lesssim T1; T2 \ \text{ w.r.t. } \ (G_\alpha, P_\beta{}^{;} \longrightarrow G_\gamma, C2_\gamma) \\
& \qquad\qquad\qquad \text{...where } P_\beta{}^{;} \text{ denotes last line of hypothesis}
\end{aligned}
\tag{5}
$$

The POB (4) shows that starting from G_k, we might achieve G_k, but certainly will achieve the weaker G_{k+1}. Hence the monotonicity property is directly applicable to guarantee a two-step transition from G_k to at least G_{k+2}, and so on, inductively.

This formalises the sequential composition of the OpA/OpC retrenchments into what we might call an *evolving simulation.* This precisely describes the evolution of the representation. This is a formal statement of error propagation, a subject thoroughly analysed in classical numerical analysis such as [1]. Thus evolving retrenchment points the way to the exploitation of numerical analysis in prover-supported verification of continuous system development.

3 Case Study: Second-order Linear Control System

We now test the intuition of evolving retrenchment against the kind of continuous application it is intended for. The case study is based on a classical second-order linear control system. Typical control engineering practice [27] is to design a discrete, computer based system ("digital system", in control engineering parlance) by approximation to the continuous. We will describe this approximate design step as a retrenchment.

A continuous control system is analogue in structure, and has analogue connections to the plant under control. System dynamics and control vary continuously in time. On the other hand, a digital system emphasises the discrete computer and interfaces with the continuous plant through sensors and A-D converters. These interfaces are discrete by virtue of being sampled, say every T seconds.

The discrete system is inevitably an approximation of the continuous, losing all information about state and input dynamics between the sample points. The precision of the approximation is a function of time. The design step from continuous to discrete control is thus an obvious candidate for description as a retrenchment.

We will use the "zero-order hold", or ZOH approximation [27, 18]. In practice this is the preferred choice over a number of approximation methods. It is named after the fact that the discrete controller output $y_d(kT)$, available only at discrete sample times, is converted into continuous $y(t)$ by the ZOH chip as follows:

$$y(t) \mathrel{\widehat{=}} y_d(kT) \qquad \text{for} \quad kT \leq t < kT + T \tag{6}$$

The aim of the retrenchment to be presented is to give a precise formal description of the precision of approximation of a continuous by a discrete control system. Such a description is typical of the style sought by formal methods practitioners, i.e. capable of formal verification. Such a description cannot be provided by refinement.

Figure 3 shows a second-order linear system describing the control of a simple industrial process. A mass m is attached to a wall by a spring k_1/dashpot k_2 damper assembly. It is constrained to move horizontally, in one dimension only. The mass is also subject to external disturbance $d(t)$ and an operator input $x_i(t)$, representing the desired reference position of the mass. The actual position (of the centre of gravity of the mass) is the system output $x_o(t)$. The uncontrolled acceleration of m, by elementary physics, then depends on $k_1 x_o(t)$, $k_2 \dot{x}_o(t)$, and $d(t)$. A controller force is required to counteract the disturbance and to move the mass towards the reference position; a typical such force is defined by the equation:

$$\text{controller force } = k_3(x_o(t) - x_i(t)) + k_4 \dot{x}_o(t) \tag{7}$$

The system is described by the equation:

$$m\ddot{x}_o(t) = -k_1 x_o(t) - k_2 \dot{x}_o(t) - k_3(x_o(t) - x_i(t)) - k_4 \dot{x}_o(t) + d(t) \tag{8}$$

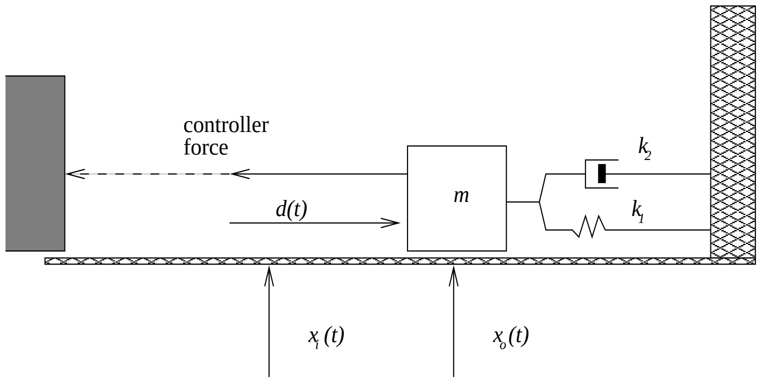

Fig. 3. A second-order linear control system

For reasons of space, only those results of the control-theoretic analysis necessary for understanding and for the retrenchment description are given; see [31] for a full analysis.

3.1 A Continuous State Space Model

We describe the case study using the state-space formulation of modern control theory. The (control-theoretic) state of a dynamic system is defined to be the vector of minimum dimensionality of variables $\mathbf{y}(t)$, such that knowledge of $\mathbf{y}(t_0)$ and all system inputs for all time after t_0, completely characterises the system behaviour.

To describe a model such as the above in state-space, we "flatten" an nth order linear differential equation into n first order equations (here n=2). Part of this process is renaming output and input variables and their derivatives to facilitate the matrix presentation below. For our example we assign state variable y_i to each derivative $x_o^{(i-1)}$. Thus y_1 represents mass position x_o and y_2 represents mass velocity $\dot{x}_o$. Inputs are renamed as components of a vector $\mathbf{u}$. Here, the following system of equations is generated:

$$\begin{aligned}
&y_1 = x_o \\
&y_2 = \dot{x}_o = \dot{y}_1 \\
&\dot{y}_2 = \ddot{x}_o = -\tfrac{1}{m}\left((k_1 + k_3)y_1 + (k_2 + k_4)y_2\right) + \tfrac{1}{m}(k_3 u_1 + u_2) \qquad (9) \\
&\quad \text{... where} \\
&u_1(t) = x_i(t) \qquad u_2(t) = d(t)
\end{aligned}$$

Thus the system can be described by the following matrix equation:

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{u} \qquad \text{... that is}$$

$$\begin{bmatrix}\dot{y}_1\\ \dot{y}_2\end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{1}{m}(k_1+k_3) & -\frac{1}{m}(k_2+k_4)\end{bmatrix}\begin{bmatrix}y_1\\ y_2\end{bmatrix} + \begin{bmatrix}0 & 0\\ \frac{k_3}{m} & \frac{1}{m}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix} \tag{10}$$

Laplace transformation, e.g. [13, 16], gives

$$\mathbf{Y}(s) = \varPhi(s)\mathbf{y}(0) + \varPhi(s)\mathbf{B}\mathbf{U}(s) \qquad \text{... where} \tag{11}$$
$$\varPhi(s) = (s\mathbf{I} - \mathbf{A})^{-1} \tag{12}$$

$\mathbf{Y}(s), \varPhi(s), \mathbf{U}(s)$ are the Laplace transforms of $\mathbf{y}(t), \varPhi(t), \mathbf{u}(t)$ respectively, where $\varPhi(t)$ is called the *transition matrix* of the system. For all $t \geq 0$, it is convenient that $\varPhi(t) = e^{\mathbf{A}t}$. $\varPhi(t)$ is calculated by inverse Laplace transformation of $\varPhi(s)$, one element at a time, by recourse to a table of Laplace transforms such as [13].

The important point here is that output $\mathbf{y}(t)$ is the sum of an initial-conditions term independent of input, and an integral term over the input. The time-domain solution for the linear system (10) is

$$\mathbf{y}(t) = \varPhi(t)\mathbf{y}(0) + \int_0^t \varPhi(t-\tau)\mathbf{B}\mathbf{u}(\tau)\,\mathrm{d}\tau \tag{13}$$

3.2 A Discrete State Space Model

The discrete model is conceptually closer to being implementable, since it assumes input $\mathbf{u}(t)$ is only available at sampling points of interval T. Information is only needed at these discrete time points: the state trace is now a sequence rather than a function of real-valued time.

The approximation lies in assuming the input to be piecewise constant, with value $\mathbf{u}(kT)$ across the sampling interval $[kT, kT+T)$. [27] shows that continuous system (10), with input piecewise constant in this way, can be described exactly as the following difference equation in state space. We call the discrete state $\mathbf{y}_d$ to distinguish it from continuous state $\mathbf{y}$. For our purposes $\mathbf{y}_d$ is only defined at sample times kT for $k \in \mathbb{N}$, although for practical purposes $\mathbf{y}_d$ is piecewise constant in continuous time by use of the ZOH chip.

$$\mathbf{y}_d(kT+T) = \mathbf{A}_d(T)\mathbf{y}_d(kT) + \mathbf{B}_d(T)\mathbf{u}(kT) \qquad \text{... where}$$
$$\mathbf{A}_d(T) = \varPhi(T) = e^{\mathbf{A}T} \qquad \mathbf{B}_d(T) = \int_0^T \varPhi(\tau)\mathbf{B}\,\mathrm{d}\tau \tag{14}$$

This picture is "local" to the k'th sampling interval. A straightforward derivation gives an analogous local expression to (14) for the continuous model, over an interval of length T, from (13). We give the continuous and discrete state space equations over interval $[kT-T, kT)$ for comparison:

$$\mathbf{y}(kT) = e^{\mathbf{A}T}\mathbf{y}(kT-T) + \int_0^T e^{\mathbf{A}\tau}\mathbf{B}\mathbf{u}(kT-\tau)\,\mathrm{d}\tau \tag{15}$$
$$\mathbf{y}_d(kT) = e^{\mathbf{A}T}\mathbf{y}_d(kT-T) + \mathbf{B}_d(T)\mathbf{u}(kT-T) \tag{16}$$

Given an input function $\mathbf{u}(t)$, then, we are interested in the approximation of continuous by discrete state, and its description as a retrenchment. That is, we seek an explicit expression for the difference function $\Delta(t) \mathrel{\widehat{=}} \mathbf{y}(t) - \mathbf{y}_d(t)$, where from (15,16) we have

$$\begin{aligned} \Delta(kT) = e^{\mathbf{A}T}\Delta(kT-T) \\ + \int_0^T e^{\mathbf{A}\tau}\mathbf{B}\mathbf{u}(kT-\tau)\,\mathrm{d}\tau - \mathbf{B}_d(T)\mathbf{u}(kT-T) \end{aligned} \tag{17}$$

3.3 Making the Approximation Explicit

Provided $\mathbf{A}$ is nonsingular (as is required by a nontrivial system (10)), then we have

$$\mathbf{B}_d(T) = \int_0^T e^{\mathbf{A}\tau}\mathbf{B}\,\mathrm{d}\tau = \mathbf{A}^{-1}e^{\mathbf{A}\tau}\mathbf{B}\Big|_0^T = \mathbf{A}^{-1}(e^{\mathbf{A}T} - \mathbf{I})\mathbf{B} \tag{18}$$

Thus the discrete contribution to error $\Delta(kT)$ from the k'th interval can be calculated exactly, giving a matrix of exponential-sinusoidal expressions in T and model parameters. The matrix $\mathbf{B}_d(t)$ (which we do not give here) is independent of k. Judicious Taylor expansion gives

$$\begin{aligned} \int_0^T e^{\mathbf{A}\tau}\mathbf{B}\mathbf{u}(kT-\tau)\,\mathrm{d}\tau - \mathbf{B}_d(T)\mathbf{u}(kT-T) \\ = \begin{bmatrix} O(T^3) \\ \frac{T^2}{2m}(k_3\dot{u}_1(kT-T) + \dot{u}_2(kT-T)) + O(T^3) \end{bmatrix} \end{aligned} \tag{19}$$

To $O(T^3)$, we see that for step k the discrete model approximates the continuous y_1, i.e. the system output x_o, exactly. The step k contribution to the error in y_2, i.e. velocity $\dot{x}_o$, is a term in T^2 and the first derivatives of the inputs. If the expansion is performed to $O(T^4)$ then the derivatives of the inputs can be seen to contribute to error in x_o.

4 Control System Design as an Evolving Retrenchment

The basis of arguing that this is a case for retrenchment is now clear: (17) and (19) provide the means, given error of approximation $\Delta(kT-T)$ at start of step k, to describe the error $\Delta(kT)$ at the end of that step.

We describe the retrenchment in the B framework of this paper, but again need to imagine the language is further enriched. As a minimum we need the reals, differentiable functions on the reals, vectors and matrices, and the associated tools of real and complex analysis implicit in this work. The syntax and semantics of B predicates and formulas needs enrichment to make statements about such types, e.g. (10) and (15).

Note that in this section, as we discuss the composition of the difference vector $\Delta(kT)$ at time kT (17), we will refer to its two summands as the *history*

Δ-vector $e^{\mathbf{A}T}\Delta(kT - T)$ and the *previous Δ-vector* (19). We will call $\Delta(kT)$ the *current Δ-vector.* Also note that from here we will use the term "state" in the sense of state variables in a retrenchment model, rather than in the control-theoretic sense.

The abstract, extended-B model is the continuous state-space model (10). State variables are given by the 2-vector $\mathbf{y}(t) = \{y_1(t), y_2(t)\}$[3] of real-valued functions, which describe system output $x_o(t)$ and its derivative. Operationally, $\mathbf{y}$ will be defined in terms of the solution (13) to the linear system (10). The retrenchment relation will represent that continuous behaviour at the discrete sample points (multiples kT of sample time T) of the concrete model[4]. The abstract model respects the granularity of the discrete in the sense that, for step k, $\mathbf{y}$ is only defined over time interval $[kT - T, kT]$. Abstract inputs for step k are given by the input vector $\mathbf{u} = \{u_1(t), u_2(t)\}$ and its derivative $\dot{\mathbf{u}} = \{\dot{u}_1(t), \dot{u}_2(t)\}$ over $[(k-1)\,T, kT]$.

Since the system behaviour is defined in terms of the solution (13) to differential equation (10), the latter is clearly appropriate as abstract invariant, conjoined with a suitable typing statement:

$$I(\mathbf{y}) \mathrel{\widehat{=}} \dot{\mathbf{y}} = \mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{u} \wedge \mathbf{y} \in [kT - T, kT] \to \mathbb{R}^2 \tag{20}$$

The abstract operation $OpA(k)$ for step k is defined in terms of the solution[5] (13). The abstract initialisation is simply $OpA(1)$:

$$OpA(k) \mathrel{\widehat{=}} \ \mathbf{y} := \lambda\, t : [kT - T, kT] \bullet e^{\mathbf{A}(t-kT+T)}\mathbf{y}(kT - T) + \int_0^{t-kT+T} e^{\mathbf{A}\tau}\mathbf{B}\mathbf{u}(t-\tau)\,\mathrm{d}\tau \tag{21}$$

$$\mathit{InitA} \mathrel{\widehat{=}} OpA(1) \tag{22}$$

Operations OpA and its concrete counterpart OpC (26) are assignments and therefore terminate universally. In retrenchments like this we can thus dispense with concern about termination.

The usual operation consistency obligation for OpA is

$$I(\mathbf{y}) \Rightarrow [OpA(k)]I(\mathbf{y}) \tag{23}$$

This is straightforwardly discharged by differentiating the OpA definition (21) with respect to t by use of the Leibnitz rule [22] for differentiating across an integral:

$$\text{... If} \qquad F(t) = \int_{A(t)}^{B(t)} f(t, \lambda)\,\mathrm{d}\lambda$$

[3] We enclose horizontally displayed vectors in parentheses $\{\}$.

[4] As in Sect. 2.2, we really mean a collection of k-indexed simple retrenchments.

[5] Note the logical close coupling that differentiation gives to invariant and behaviour; this is unusual in B models.

$$\text{... then} \qquad \frac{\mathrm{d}}{\mathrm{d}t}\int_{A(t)}^{B(t)} f(t,\lambda)\,\mathrm{d}\lambda = \int_{A}^{B} \frac{\delta f(t,\lambda)}{\delta t}\,\mathrm{d}\lambda + f(t,B)\frac{\mathrm{d}B}{\mathrm{d}t} - f(t,A)\frac{\mathrm{d}A}{\mathrm{d}t} \tag{24}$$

The rule is valid over an interval (a, b) for t provided $f, \frac{\delta f}{\delta t}$ are continuous on $[a, b]$, $A'(t), B'(t)$ are continuous on (a, b), and $A(t) \leq \lambda \leq B(t)$. $f(t, \lambda)$ is the integrand term in (21), including the input term $\mathbf{u}(t)$. This rule imposes conditions on the input, but these are no more onerous than those imposed by our use of Taylor's theorem in Sect. 3.3.

The concrete model is given by the discrete state-space model (16). We call the state vector $\mathbf{y}_d = \{y_{d1}, y_{d2}\}$. Concrete inputs are given by the input vector $\mathbf{u}$; assume for now these are identical to the abstract input values at times kT.

It is a feature of the ZOH design method that (10) is approximately satisfied by the discrete model (16). The discrete world is not rich enough to state this in the discrete invariant; indeed, to attempt to do so would breach separation of concerns in design. The retrenchment will describe this approximation. For the concrete invariant a simple bounding constraint (for some $L_1, U_1, L_2, U_2 \in \mathbb{R}$), which serves as a typing predicate, will suffice:

$$J(\mathbf{y}_d) \mathrel{\widehat{=}} y_{d1} \in [L_1, U_1] \wedge y_{d2} \in [L_2, U_2] \tag{25}$$

The concrete operation is defined in the obvious way from (16):

$$OpC(k) \mathrel{\widehat{=}} \mathbf{y}_d := e^{\mathbf{A}T}\mathbf{y}_d + \mathbf{B}_d(T)\mathbf{u}(kT - T) \tag{26}$$

Initialisation is in terms of user-defined initial conditions $\mathbf{y}_{dINIT}$ at time T, not necessarily the same as $\mathbf{y}(T)$. However, the refinement of initialisations POB (30) dictates how close $\mathbf{y}(T)$ and initial-state $\mathbf{y}_d$ should be.

$$InitC \mathrel{\widehat{=}} \mathbf{y}_d := \mathbf{y}_{dINIT} \tag{27}$$

The operation consistency obligation for OpC is established trivially from the OpC definition (26):

$$J(\mathbf{y}_d) \Rightarrow [OpC(k)]J(\mathbf{y}_d) \tag{28}$$

The retrieve relation G decomposes the current Δ-vector $\Delta(kT)$ into its two elements, and does not refer to $\Delta(kT-T)$. G bounds $\Delta(kT)$ in terms of constants $\varepsilon_k, \dot{\varepsilon}_k$[6]: the designer chooses bounds $\varepsilon_1, \dot{\varepsilon}_1$, and we will shortly see how the evolving bounds are defined inductively. For $k \in \mathbb{N}_1$:

$$G(\mathbf{y}, \mathbf{y}_d, k, \varepsilon_k, \dot{\varepsilon}_k) \mathrel{\widehat{=}} |y_1(kT) - y_{d1}| \leq \varepsilon_k \wedge |y_2(kT) - y_{d2}| \leq \dot{\varepsilon}_k \tag{29}$$

We require refinement of initialisations to state the precision of approximation at start time T:

$$[InitC]\neg\, [InitA]\neg\; G(\mathbf{y}, \mathbf{y}_d, 1, \varepsilon_1, \dot{\varepsilon}_1) \tag{30}$$

[6] We use a little lexical licence in naming the error bound on the derivative term $\dot{\varepsilon}_k$.

Next we seek the remaining syntactic components of the evolving retrenchment of $OpA(k)$ by $OpC(k)$, as defined by (2) – WITHIN clause P and CONCEDES clause C:

$$\begin{aligned} & I(\mathbf{y}) \wedge G(\mathbf{y}, \mathbf{y}_d, k-1, \varepsilon_{k-1}, \dot{\varepsilon}_{k-1}) \wedge J(\mathbf{y}_d) \wedge P(\mathbf{y}, \mathbf{y}_d, \mathbf{u}, \dot{\mathbf{u}}, k-1) \\ & \quad \Rightarrow [OpC(k)]\neg\, [OpA(k)] \\ & \qquad \neg\, (G(\mathbf{y}, \mathbf{y}_d, k-1, \varepsilon_{k-1}, \dot{\varepsilon}_{k-1}) \vee C(\mathbf{y}, \mathbf{y}_d, \mathbf{u}, \dot{\mathbf{u}}, k, \varepsilon_k, \dot{\varepsilon}_k)) \end{aligned} \tag{31}$$

We will not use the logical variable A and expect that the concession will state different error bounds $\varepsilon_k, \dot{\varepsilon}_k$ to those of $G(k-1)$. For the WITHIN clause it suffices to define:

$$P(\mathbf{y}, \mathbf{y}_d, \mathbf{u}, \dot{\mathbf{u}}, k-1) \mathrel{\widehat{=}} \mathsf{true} \tag{32}$$

Given (29), at time $kT - T$ we have for each element of the history Δ-vector

$$\begin{aligned} \left|\{e^{\mathbf{A}T}\Delta(kT-T)\}_i\right| &\le \left|e^{\mathbf{A}T}_{i1}\right| |\Delta(kT-T)_1| + \left|e^{\mathbf{A}T}_{i2}\right| |\Delta(kT-T)_2| \\ &\le \left|e^{\mathbf{A}T}_{i1}\right| \varepsilon_{k-1} + \left|e^{\mathbf{A}T}_{i2}\right| \dot{\varepsilon}_{k-1} \end{aligned} \tag{33}$$

For the previous Δ-vector we have, from (19), for some constant M_1, M_2:

$$\begin{aligned} & |\{\text{previous } \Delta\text{-vector}\}_1| \le M_1 T^3 \\ & |\{\text{previous } \Delta\text{-vector}\}_2| \le \frac{1}{2m} |k_3 \dot{u}_1(kT-T) + \dot{u}_2(kT-T)|\, T^2 + M_2 T^3 \end{aligned} \tag{34}$$

The CONCEDES clause C will bound the current difference vector $\Delta(kT)$ according to (33, 34). That is, C has effectively been derived in the (implicit, in this paper) error analysis (19) of difference function (17). Hence C will always be derivable by a prover sufficiently rich in theory and strategy to reproduce this reasoning.

C is free in all state variables and abstract input derivatives $\dot{\mathbf{u}}$:

$$\begin{aligned} & C(\mathbf{y}, \mathbf{y}_d, \dot{\mathbf{u}}, k) \mathrel{\widehat{=}} \\ & \quad |y_1(kT) - y_{d1}| \le \left|e^{\mathbf{A}T}_{11}\right| \varepsilon_{k-1} + \left|e^{\mathbf{A}T}_{12}\right| \dot{\varepsilon}_{k-1} + M_1 T^3 \\ & \quad \wedge |y_2(kT) - y_{d2}| \le \left|e^{\mathbf{A}T}_{21}\right| \varepsilon_{k-1} + \left|e^{\mathbf{A}T}_{22}\right| \dot{\varepsilon}_{k-1} \\ & \qquad\qquad + \frac{1}{2m} |k_3 \dot{u}_1(kT-T) + \dot{u}_2(kT-T)|\, T^2 + M_2 T^3 \end{aligned} \tag{35}$$

The RHS's of the two inequalities in (35) are renamed to $\varepsilon_k, \dot{\varepsilon}_k$ respectively. Any concern about defining ε_k and thus G_k in terms of input values can be addressed by reading the last line of (35) in terms of the maximum absolute values, over all time of interest, of $\dot{u}_1(t), \dot{u}_2(t)$[7]. The concession is then in the right form to be read as retrieve relation G for step k. This yields a neat description in terms of *evolving retrenchment.* Since the two bounding expressions $\varepsilon_k, \dot{\varepsilon}_k$ of (35) are functions only of $\varepsilon_{k-1}, \dot{\varepsilon}_{k-1}$ (and fixed model parameters) respectively, we can define for all $k > 1$:

$$G(\mathbf{y}, \mathbf{y}_d, k, \varepsilon_k, \dot{\varepsilon}_k) \mathrel{\widehat{=}} C(\mathbf{y}, \mathbf{y}_d, \dot{\mathbf{u}}_{\max}, k-1) \tag{36}$$

[7] The assumption of such maxima is typical control engineering practice.

and this gives the inductive definition of $\varepsilon_k, \dot{\varepsilon}_k$. Therefore the retrenchment statement (31) becomes:

$$\begin{aligned} & I(\mathbf{y}) \wedge G(\mathbf{y}, \mathbf{y}_d, k-1, \varepsilon_{k-1}, \dot{\varepsilon}_{k-1}) \wedge J(\mathbf{y}_d) \\ & \Rightarrow [OpC(k)]\neg\ [OpA(k)]\neg\ (G(\mathbf{y}, \mathbf{y}_d, k-1, \varepsilon_{k-1}, \dot{\varepsilon}_{k-1}) \vee G(\mathbf{y}, \mathbf{y}_d, k, \varepsilon_k, \dot{\varepsilon}_k)) \end{aligned} \tag{37}$$

This is precisely the type of scenario that the evolving variant of retrenchment was devised to describe. Retrieve relation $G(k-1)$ of predetermined error may be preserved by step k; the concession $G(k)$ certainly will be. Thus (37) recasts the concession of (31) as an *evolved* retrieve relation[8], which is available for use as the basis for the step $k+1$ retrenchment.

We briefly analyse the various sources of error contributed by step k, which the concession describes. Taylor expansion error is small at $O(T^3)$. The error from abstract input derivatives $\dot{u}_1, \dot{u}_2$, having a T^2-factor, is only significant for rapidly varying inputs. In the real world of the model, this is only possible for the disturbance force $d(t) = u_2(t)$. Usual engineering practice is to assume disturbance inputs have bounded (if large) derivatives.

The significant factors which multiply start-of-step error $G(k-1)$ are the elements of transition matrix $e^{\mathbf{A}T}$, i.e. constant expressions in the model parameters. It is thus the transition matrix, independent of external inputs, which will usually determine improvement or decay in the error evolution $\Delta(kT)$. A lower bound on sample time T is determined by the target hardware, but the designer has some freedom in fixing the other model parameters. There are standard control engineering design constraints on parameter-setting, concerned with required system response to representative input signals [25, 27]. It may be the case that sufficient design freedom exists to enable model parameters to be chosen such that every $\left|e^{\mathbf{A}T}_{ij}\right|$ is sufficiently less than 1. In this situation the approximation error of the retrenchment can be guaranteed to improve.

4.1 Another retrenchment

An appealing fact about this ZOH discrete approximation to a continuous control system is that the difference vector $\Delta(kT)$ (17, 19) depends only on derivatives of system inputs (at least to $O(T^3)$). This fact makes it easy to generalise the example to distinguish between abstract and concrete inputs $\mathbf{u}$ and $\mathbf{u}_d$. Typically, precision would be lost in moving from an analogue specification in continuous variables and inputs, to digital hardware-based input sensors with finite precision. Revisiting (19) with input approximation, we have a bigger expression in u_{d1}, u_{d2}. Defining difference vectors for the input representations $\Delta u_i(kT) \mathrel{\widehat{=}} u_i(kT) - u_{di}(kT)$ for $i \in \{1, 2\}$, we have[9]:

$$\int_0^T e^{\mathbf{A}\tau}\mathbf{B}\mathbf{u}(kT-\tau)\,\mathrm{d}\tau - \mathbf{B}_d(T)\mathbf{u}(kT-T)$$

[8] This retrenchment suggests a definition of *evolving refinement.*

[9] ρ, ω are derived parameters from the model (8)

$$= \begin{bmatrix} \frac{T^2}{2m}(k_3 \Delta u_1(kT-T) + \Delta u_2(kT-T)) + O(T^3) \\ \begin{pmatrix} \frac{(T-\rho\omega T^2)}{m}(k_3 \Delta u_1(kT-T) + \Delta u_2(kT-T)) \\ +\frac{T^2}{2m}(k_3 \dot{u}_1(kT-T) + \dot{u}_2(kT-T)) + O(T^3) \end{pmatrix} \end{bmatrix} \tag{38}$$

The extra error introduced by input approximation is a linear combination of the $\Delta u_i(kT - T)$ terms, with factors in powers of T. We can therefore choose WITHIN bounds to input error $\Delta u_i(kT - T)$ that make the overall extra error contributed insignificant. For example, to make

$$|\{\text{previous } \Delta\text{-vector}\}_1| \leq \frac{\varepsilon_k}{10} + \frac{\dot{\varepsilon}_k}{10} + M_1 T^3 \tag{39}$$

$$\wedge\ |\{\text{previous } \Delta\text{-vector}\}_2| \leq \frac{\varepsilon_k}{10} + \frac{\dot{\varepsilon}_k}{10} \tag{40}$$
$$+ \frac{1}{2m}|k_3 \dot{u}_1(kT-T) + \dot{u}_2(kT-T)|\, T^2 + M_2 T^3$$

bound the input error to reduce the new error terms in (38) as follows:

$$|\Delta u_1(kT-T)| \leq \min\left\{ \frac{\varepsilon_k m}{5k_3 T^2}, \frac{\varepsilon_k m}{5k_3(T-\rho\omega T^2)} \right\}$$
$$\wedge\ |\Delta u_2(kT-T)| \leq \min\left\{ \frac{\dot{\varepsilon}_k m}{5T^2}, \frac{\dot{\varepsilon}_k m}{5(T-\rho\omega T^2)} \right\} \tag{41}$$

The retrenchment description of this approximated-input scenario is that already given, with P strengthened to specify the input error bounds (41), and C weakened in its error bounds by $\frac{\varepsilon_k}{10} + \frac{\dot{\varepsilon}_k}{10}$ as per (39, 40).

5 Conclusion

Retrenchment was devised as a method to support the formal description of design transformations too rich for refinement. In this paper we introduce a generalisation, evolving retrenchment, to enable description of simulation-like behaviour through an evolving representation relation.

The retrenchment approach, like refinement, takes the abstract system model as given. The case study retrenchment in this paper has been developed on that assumption. The suggestion following the retrenchment (37) suggested another, more intricate and ambitious use for retrenchment. We indicated the possibility of exploiting design freedom (i.e. freedom of choice of model parameters) in the abstract description in order to "improve" the retrenchment. Improvement in the case study constituted a decreasing (as opposed to increasing) sequence of error bounds, i.e. an increasingly precise retrieve relation $G(k)$ over the k-indexed evolving retrenchment. Of course, the notion of improvement of retrenchment must be application-specific, and in any event about getting logically closer to some ideal refinement.

A more ambitious use of retrenchment in formal development, then, is one that exploits underspecification or nondeterminism in the abstract specification, in order to reduce the logical distance from pure refinement. Such a usage would contribute to two obvious methodological questions about retrenchment:

> "How closely does the program approximate the requirement? If you can't refine, how close can you get to refinement?".

These questions relate to nontrivial developments involving the composition of retrenchment and refinement steps, from abstract model to compilable code.

The first question is answered by the transitivity property of retrenchment [5, 29]: the composite concession of two composed retrenchments can be calculated mechanically. A more sophisticated answer is given by *maximally abstract retrenchment* [4] to the related question "Given a retrenchment from abstract model M_A to concrete model M_C, what is the corresponding abstract model M_U which is refinable to M_C?". In a theoretical sense, this gives requirements validation by constructing the abstract counterpart to the implemented system, for comparison with original requirements.

The second question is addressed, but not answered, by two pieces of work. The notion of *weakest retrenchment* [31] derives from the intuition that a weaker WITHIN clause is preferred where possible. The dual notion of *strongest retrenchment*, in terms of logical proximity of the CONCESSION and RETRIEVES clauses has been proposed [31], and is more relevant to the second question.

The focus of the paper is the description of the design step from continuous to ZOH-discrete linear system as an evolving retrenchment. This is set in a version of B much enriched with appropriate types, axioms and analysis. For practical work supported by theorem provers, it is straightforward to describe the models and their retrenchment in a language such as PVS [12] or HOL [20]. These provide some real analysis support: typically, prover support outside the usual discrete types and logics is scarce. However, current interest in the integration of theorem prover and computer algebra technologies such as [21, 17, 33, 28], therefore holds great potential for support of applications such as ours.

References

[1] K.E. Atkinson. *An Introduction to Numerical Analysis.* Wiley, 1989.

[2] R. J. R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction.* Springer, 1998.

[3] R.J.R. Back and M. Butler. Fusion and simultaneous execution in the refinement calculus. *Acta Informatica*, 35:921–949, 1998.

[4] R. Banach. Maximally abstract retrenchments. In *Proc. IEEE ICFEM2000*, pages 133–142, York, August 2000. IEEE Computer Society Press.

[5] R. Banach and M. Poppleton. Retrenchment: An engineering variation on refinement. In D. Bert, editor, *2nd International B Conference*, volume 1393 of *LNCS*, pages 129–147, Montpellier, France, April 1998. Springer.

[6] R. Banach and M. Poppleton. Retrenchment: An engineering variation on refinement. Technical Report Report UMCS-99-3-2, University of Manchester Computer Science Department, 1999.

[7] R. Banach and M. Poppleton. Retrenchment and punctured simulation. In K. Araki, A. Galloway, and K. Taguchi, editors, *Proc. IFM'99:Integrated Formal Methods 1999*, pages 457–476, University of York, June 1999. Springer.

[8] R. Banach and M. Poppleton. Sharp retrenchment, modulated refinement and simulation. *Formal Aspects of Computing*, 11:498–540, 1999.

[9] R. Banach and M. Poppleton. Retrenchment, refinement and simulation. In J. Bowen, S. King, S. Dunne, and A. Galloway, editors, *Proc. ZB2000*, volume 1878 of *LNCS*, York, September 2000. Springer.

[10] A. Blikle. The clean termination of iterative programs. *Acta Informatica*, 16:199–217, 1981.

[11] D. Coleman and J.W. Hughes. The clean termination of pascal programs. *Acta Informatica*, 11:195–210, 1979.

[12] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A tutorial introduction to PVS. In R. France, S. Gerhart, and M. Larrondo-Petrie, editors, *WIFT'95: Workshop on Industrial-Strength Formal Specification Techniques*, Boca Raton, Florida, April 1995. IEEE Computer Society Press.

[13] J.J. D'Azzo and C.H. Houpis. *Linear Control System Analysis and Design.* McGraw-Hill, 4 edition, 1995.

[14] W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison.* Cambridge University Press, 1998.

[15] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[16] R.C. Dorf and R.H. Bishop. *Modern Control Systems.* Addison-Wesley, 1998.

[17] M. Dunstan, T. Kelsey, U. Martin, and S. Linton. Lightweight formal methods for computer algebra systems. In *ISSAC'98: Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, Rostock, 1998. ACM Press.

[18] G.F. Franklin, J.D. Powell, and M.L. Workman. *Digital Control of Dynamic Systems.* Addison-Wesley Longman, 3rd edition, 1998.

[19] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 1991.

[20] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[21] John Harrison and Laurent Théry. A skeptic's approach to combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.

[22] F.B. Hildebrand. *Advanced Calculus for Applications.* Prentice-Hall, 1962.

[23] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, October 1969.

[24] D.S. Neilson. *From Z to C: Illustration of a Rigorous Development Method.* PhD thesis, Oxford University Programming Research Group, 1990. Technical Monograph PRG-101.

[25] K. Ogata. *Modern Control Engineering.* Prentice-Hall, 1997.

[26] O. Owe. Partial logics reconsidered: A conservative approach. *Formal Aspects of Computing*, 3:1–16, 1993.

[27] P.N. Paraskevopoulos. *Digital Control Systems.* Prentice-Hall, 1996.

[28] E. Poll and S. Thompson. Adding the axioms to Axiom: Towards a system of automated reasoning in Aldor. Technical Report 6-98, Computing Laboratory, University of Kent, May 1998.

[29] M. Poppleton and R. Banach. Retrenchment: extending the reach of refinement. In *ASE'99: 14th IEEE International Conference on Automated Software Engineering*, pages 158–165, Cocoa Beach, Florida, October 1999. IEEE Computer Society Press.

[30] M. Poppleton and R. Banach. Retrenchment: Extending refinement for continuous and control systems. In *Proc. IWFM'00*, Springer Electronic Workshop in Computer Science Series http://ewic.org.uk/ewic, NUI Maynooth, July 2000. Springer.

[31] M.R. Poppleton. *Formal Methods for Continuous Systems: Liberalising Refinement in B.* PhD thesis, Department of Computer Science, University of Manchester, 2001.

[32] S. Stepney, D. Cooper, and J. Woodcock. More powerful Z data refinement: Pushing the state of the art in industrial refinement. In J.P. Bowen, A. Fett, and M.G. Hinchey, editors, *11th International Conference of Z Users*, volume 1493 of *LNCS*, pages 284–307, Berlin, Germany, September 1998. Springer.

[33] S. Thompson. Integrating computer algebra and reasoning through the type system of Aldor. In H. Kirchner and C. Ringeissen, editors, *Frontiers of Combining Systems: Frocos 2000*, volume 1794 of *LNCS*, pages 136–150. Springer, March 2000.

[34] J. Woodcock and J. Davies. *Using Z: Specification, Refinement and Proof.* Prentice-Hall, 1996.