

**SOFTWARE VERIFICATION RESEARCH CENTRE
DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY OF QUEENSLAND**

**Queensland 4072
Australia**

TECHNICAL REPORT

No. 97-36

**Applying the Cogito Program
Development Environment to
Real-Time System Design**

**C. J. Fidge P. Kearney
A. P. Martin**

**Phone: +61 7 3365 1003
Fax: +61 7 3365 1533**

To appear in *Proceedings of the 21st Australasian Computer Science Conference*,
Perth, February 1998.

Note: Most SVRC technical reports are available
via anonymous ftp, from `ftp.cs.uq.edu.au` in the
directory `/pub/SVRC/techreports`.

Applying the Cogito Program Development Environment to Real-Time System Design

C. J. Fidge P. Kearney A. P. Martin

Abstract. We show how a formal program development environment, previously used for sequential, non-real-time applications only, can be exploited for parallel, real-time system design. A pragmatic approach is adopted, making best use of existing technologies, in order to quickly achieve useful results.

1 Introduction

After an intense period of research, formal program development environments are now maturing. Tools to support specification, refinement, verification and analysis are becoming powerful enough for industrial applications [Hart et al., 1996], although much work remains.

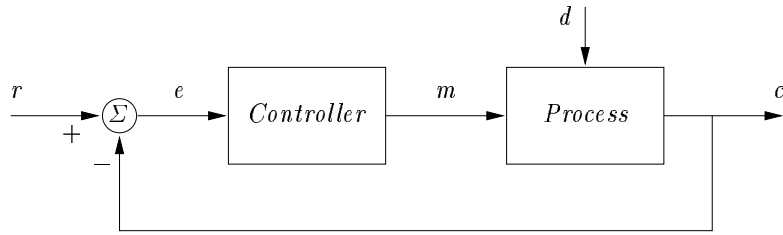
Not surprisingly, contemporary environments and tools emphasise well-established concepts, such as sequential state machines [Abrial, 1996; Cant et al., 1996]. Nevertheless, there is an increasingly urgent demand for practical techniques that can be used in more challenging application domains, especially real-time systems. Unfortunately, this is an area where there is still considerable disagreement about fundamentals such as specification notations, let alone established methods and tools.

To respond to this need, we show that practical results can be achieved immediately, by integrating and making best use of existing technologies, without waiting for the arrival of entirely new real-time methods and tools. We express the latest real-time formalisms in an existing program development environment, Cogito [Bloesch et al., 1995]. The result is then tested on an application from the domain of control theory, a field far removed from previous Cogito case studies.

2 Enabling Technologies

2.1 The Cogito Formal Development Environment

Cogito consists of methods and tools for the formal development of program code from specifications [Bloesch et al., 1995]. Cogito's formal development language, *Sum* [Kazmierczak et al., 1995], is based on the popular specification language *Z*, with extensions for modularisation, defining and combining state machines, stating preconditions, and expressing implementations. A theorem



r : Reference input variable e : Measured error (between r and c)
 c : Controlled output variable d : External disturbance (to c)
 m : Manipulated control variable

Fig. 1. Typical process control system.

prover, *Ergo* [Bloesch et al., 1996], is available for proving expected properties of specifications and showing the correctness of formal development (refinement) steps. Other Cogito tools support type-checking and translation of specifications, large-scale project management, and animation of state-machine specifications.

Cogito has been used in a number of case studies, including a significant industrial trial [Hart et al., 1996]. However, the method and toolset are heavily optimised towards the development of sequential program code, starting from a state-machine specification, rather than parallel, real-time systems.

2.2 Process Control Theory

Control theory is the means by which engineers design systems that react to changes in their environment in a timely way. Figure 1 illustrates a typical *closed-loop process control* system [Leigh, 1992, p. 13]. Its major components are the environmental *process* to be controlled, and the computerised *controller* that attempts to influence the behaviour of the process. Input to the system consists of a *reference* (or *set*) point variable r . The observable *controlled output* variable c is produced by the process.

The controller's goal is to manipulate the process in such a way that output c equals reference point r [Bollinger and Duffie, 1988, §2.3]. To do this, it uses an *error-driven* feedback loop [Leigh, 1992, p. 13]. The controller monitors the error e between the actual system output c and the desired value r . It uses this knowledge to construct a *manipulated control* variable m , which alters the behaviour of the process, in order to make c and r coincide. This task is often made harder by the presence of an unpredictable external *disturbance* (or *system load*) d , which also influences c [Leigh, 1992, §2.4] [Bollinger and Duffie, 1988, p. 15]. Another challenge is that physical delays may mean that the controller is working with stale inputs, and its attempts to manipulate the system output may take time to have any effect. Thus it is usually possible for c to only approximate r , approaching it slowly, and perhaps oscillating around the desired

value [Holzbock, 1958, ch. 1] [Leigh, 1992, p. 25].

3 A Real-Time Model for Cogito

As noted above, Cogito specifications and refinements are structured as state machines, following the conventions traditionally adopted by Z users [Wordsworth, 1992]. Although satisfactory for sequential systems, this approach is not always the best choice for real-time systems. Such systems typically contain parallel components that interact repeatedly, with each interaction required to occur within a strict window of time. Such behaviour is most succinctly described via *traces* of activity. It is for this reason that engineers and physicists model dynamic systems as functions from times to values.

A number of computing formalisms have emerged that build on this established viewpoint [Mahony, 1992; Olderog et al., 1996; Scholefield et al., 1994]. We decided to adopt the simplest approach compatible with Cogito's existing features for manipulating and reasoning about predicates. We were especially influenced by the notion of making (careful!) use of logic operators to structure specifications and undertake refinements [Olderog et al., 1996].

Generally, the description of a real-time system *component* consists of three distinct parts [Mahony, 1992, pp. 6–7] [Scholefield et al., 1994, p. 224] [Olderog et al., 1996, p. 118]:

1. a *frame* listing the variables \mathbf{v} *controlled by* this component,
2. an *assumption* predicate A describing the anticipated (trace) behaviour of *input* variables \mathbf{u} from the environment, and
3. a *commitment* predicate C describing the required (trace) behaviour of controlled variables \mathbf{v} , typically in terms of the inputs \mathbf{u} .

The last two items are the timed-trace analogues of the familiar pre and post-conditions used for defining state-machine operations. In Cogito each assumption A and commitment C predicate can be represented by a Sum schema. A system component is then defined using schema implication, $A \Rightarrow C$, stating that the commitment must be met *if* the environmental assumption is true [Olderog et al., 1996, p. 118].

The first item is necessary to avoid ambiguities in specifications involving parallel components that refer to a common variable. Conflicting requirements on the variable can be resolved if we make clear which component *controls* the variable, and enforce the reasonable restriction that each variable is controlled by one component only [Olderog et al., 1996, p. 119] [Mahony, 1992, p. 14] [Scholefield et al., 1994, p. 222]. The 'frame' makes the 'controlled' variables explicit; other variables referred to are assumed to be inputs. In the absence of an explicit frame we adopt a syntactic approach to distinguish controlled variables from inputs [Olderog et al., 1996, p. 120]. As shown in Section 4 we write commitment predicates as difference equations of the form ' $\forall t \bullet v(t) = f(u)$ ', or similar, and adopt the convention that the variable v on the left is the controlled one.

A complete specification is the parallel composition of one or more such components. We can use Sum schema conjunction, i.e., $(A_1 \Rightarrow C_1) \wedge \dots \wedge (A_n \Rightarrow C_n)$, for parallel composition [Olderog et al., 1996, p. 118], provided that the controlled variables of each component are distinct [Mahony, 1992, p. 14].

System development, or *refinement*, then proceeds by finding a *design* predicate D , that links inputs and controlled variables in such a way that the original commitment is satisfied, as long as the assumption holds. The proof obligation is then a Sum predicate $D \Rightarrow (A \Rightarrow C)$ [Olderog et al., 1996, p. 118].

Real-time formalisms also introduce special ‘interval’ operators for expressing trace requirements [Olderog et al., 1996, §5.2.3] [Millerchip et al., 1993, §1.1] but, in the small discrete-time case study described in Section 4, simple difference equations are a sufficient modelling basis [Bollinger and Duffie, 1988, §2.4].

4 A Real-Time Case Study

4.1 Specification

Firstly we specify the desired system properties in Sum. We adopt the usual model that computer-controlled systems change their state at discrete times only [Leigh, 1992, ch. 11]. The time domain can thus be represented by the natural numbers, with each unit denoting one *sampling interval* [Bollinger and Duffie, 1988, p. 8].

$$\mathbb{T} == \mathbb{N} \quad \text{[Discrete time domain]}$$

We assume that system variables and constants are all of type real (in order to avoid distracting concerns of boundary conditions and integer arithmetic).

$$\mathbb{V} == \mathbb{R} \quad \text{[Range of variable values]}$$

Ideally, we want output c to equal the reference input r as soon as possible. However, recognising that it will take some time to adjust the output accordingly, we require only that c approaches r at some minimum (non-negative) *speed* s [Holzbock, 1958, pp. 6–7]. Smaller values of s specify faster responsiveness.

$$\left. \begin{array}{l} s : \mathbb{R}_+ \\ \hline s \leq 1 \end{array} \right\} \text{[Required speed of corrective action]}$$

The shaded area in the example illustrated in Figure 2 shows bounds for acceptable values of c . Here speed $s = \frac{3}{4}$ and, since the acceptable error at time 2 may be as great as 19, we thus require the error at time 3 to be no more than $19 * \frac{3}{4} \approx 14$, and so on. This accounts for the characteristic ‘cone’ shape.

In practice, though, there is often a limit to how small the residual error can be made. An external disturbance d may continually deflect c from the desired value, causing a permanent deviation from r that cannot be corrected. (At least not by the ‘proportional’ type of controller we introduce in Section 4.2.) The worst-case acceptable deviation is the *offset* f [Holzbock, 1958, pp. 46–8].

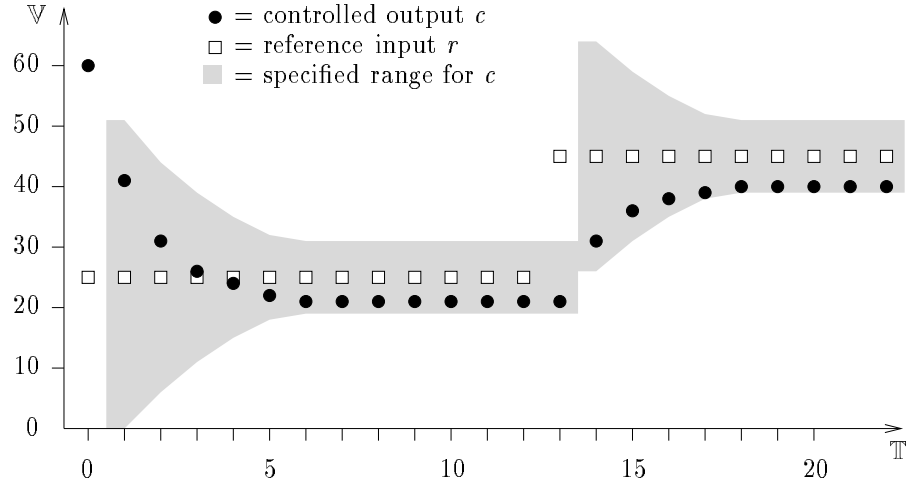


Fig. 2. Specified versus actual system behaviour for reference input $r = 25$ until time 12, followed by $r = 45$; initial (arbitrary) output $c = 60$; and constants $s = \frac{3}{4}$, $f = 6$, $K_c = 2$, $K_p = \frac{1}{4}$ and $D = -2$.

| $f : \mathbb{R}_+$ [Offset, i.e., acceptable deviation from desired value]

Thus, once output c is within f units of r , we merely require c to stay within this range. In Figure 2 the acceptable error at time 5 is 7, so at time 6 we would normally be expected to reduce the error to $7 * \frac{3}{4} \approx 5$. However, since this is less than the acceptable offset $f = 6$, the requirement from time 6 onwards is merely that output c may move no further than f units from r .

Formally, the overall requirement is specified in Cogito by the following Sum schema. (For some real number x and non-negative real y , let $x \pm y == \{z : \mathbb{R} \mid x - y \leq z \leq x + y\}$.)

<p><i>System</i></p> <p>$r, c : \mathbb{T} \rightarrow \mathbb{V}$</p> <hr/> <p>$\forall t : \mathbb{T} \bullet$ let $e(t) == r(t) - c(t) \bullet$ $c(t + 1) \in r(t) \pm \max(f, e(t) * s)$</p>

It states that the system has two externally-observable, time-dependent variables, r and c , both declared as total functions from times to values. The predicate part expresses a property that must hold for all times t . It defines the error e at time t to be the difference between r and c at this time. It then defines the acceptable range of values for output c at the *next* moment of time $t + 1$. This range is centred on the value of the reference point r at time t , and extends

as far as either the offset f , or the magnitude of the error e multiplied by the required speed of response s , whichever is greater.

The specification allows for a delay of one sampling interval before the system is required to respond to changes in the reference point. In Figure 2 the initial value of c is arbitrarily assumed to be 60. No output value can be specified at time 0, but at time 1 c is required to approach r at a rate proportional to s and the initial error $25 - 60 = -35$. It is this initial error value that determines the size of the first ‘cone’. The reference input changes from 25 to 45 at time 13. (Discrete models assume that inputs are constant between sampling times [Bollinger and Duffie, 1988, p. 24].) The smaller *worst-case* difference between c and the new reference point, $45 - 19 = 26$, determines the second cone’s size. Again, however, the allowable delay in responsiveness means that c is not required to start approaching the new reference point until time 14.

4.2 Proposed System Design

We now design a control system intended to satisfy the above specification, following the principles described in Section 2.2. For the physical process, a *process gain* constant K_p determines how much impact a unit change of the manipulated control variable m will have on c .

$$\left| \begin{array}{l} K_p : \mathbb{R}_+ \end{array} \right. \quad \text{[Process gain]}$$

The value of the output c at any time is then determined by the previous output value, plus the process gain K_p multiplied by the previous controller input m , plus the previous disturbance d .

$$\begin{array}{|l} \hline \textit{Process} \\ \hline m, c, d : \mathbb{T} \rightarrow \mathbb{V} \\ \hline \forall t : \mathbb{T} \bullet c(t+1) = c(t) + K_p * m(t) + d(t) \\ \hline \end{array}$$

This process thus has a time lag of one sampling interval for its inputs to have an effect on c .

Similarly for the controller, a *controller gain* constant K_c determines how much impact each unit of measured error e will have on m .

$$\left| \begin{array}{l} K_c : \mathbb{R}_+ \\ \hline \text{let } K == K_c * K_p \bullet \\ \quad K \leq 1 \end{array} \right. \quad \begin{array}{l} \text{[Controller gain]} \\ \text{[System gain]} \end{array}$$

The overall *system gain* K determines how much influence each unit of measured error has on c . Larger values of K achieve faster responsiveness. K may not exceed 1 because it would be nonsensical to make a correction *greater* than the observed error. We then define a trivial *proportional* controller [Bollinger and Duffie, 1988, p. 25], in which the value of the manipulated variable m is equal to the control gain K_c multiplied by the measured error e .

<i>Controller</i>
$r, m, c : \mathbb{T} \rightarrow \mathbb{V}$
$\forall t : \mathbb{T} \bullet$ <div style="margin-left: 20px;"> let $e(t) == r(t) - c(t) \bullet$ $m(t) = K_c * e(t)$ </div>

Schema *Controller* defines the value of m at time t using input values at the *same* time t . The *Controller* design thus appears to behave instantaneously. This is a simple *first-order response* model, used when processing time is negligible compared to the sampling interval [Bollinger and Duffie, 1988, pp. 113, 117–9] [Leigh, 1992, ch. 11]. (To further refine schema *Controller* to program code would require a more fine-grained time model, however.)

4.3 Environmental Assumption

Schema *Process* showed how disturbance d may have an unpredictable impact on output c . However, if this effect is not bounded we could not reasonably expect *any* control system to compensate for it. Let constant D be the largest possible *disturbance* at any time.

| $D : \mathbb{R}$ [Worst-case external disturbance]

Constant D is a *signed* number because in practice the disturbance may act in either direction, either helping or hindering our attempts to make c approach r .

The following Sum schema then introduces an assumption that the magnitude of the disturbance at any time is bounded by the magnitude of D .

<i>MaxDisturb</i>
$d : \mathbb{T} \rightarrow \mathbb{V}$
$\forall t : \mathbb{T} \bullet d(t) \leq D $

4.4 Proof of Correctness

We now want to show that the design expressed by predicates *Process* and *Controller* is a valid refinement of the specification expressed by predicate *System*, assuming that *MaxDisturb* holds. Formally, the proof obligation is expressed by the following Sum predicate.

$$(Controller \wedge Process) \Rightarrow (MaxDisturb \Rightarrow System)$$

The Cogito tools automatically translate this into a theory that can be manipulated by the Ergo theorem prover.

Since we are concerned with worst-case scenarios, we can assume that the disturbance d is *always* the worst-case value, without weakening the proof. We

can thus replace ‘ $d(t)$ ’ with ‘ D ’ in *Process*, omit schema *MaxDisturb*, and re-express the proof requirement as follows.

$$\begin{aligned}
& \forall t : \mathbb{T} \bullet \\
& \text{let } e(t) == r(t) - c(t); K == K_c * K_p \bullet \\
& (m(t) = K_c * e(t)) \wedge (c(t+1) = c(t) + K_p * m(t) + D) \\
& \Rightarrow (c(t+1) \in r(t) \pm \max(f, |e(t)| * s))
\end{aligned} \tag{1}$$

The universally-quantified expression can then be simplified as follows.

$$\begin{aligned}
(1) & \equiv \text{‘by eliminating } m(t)\text{’} \\
& (c(t+1) = c(t) + K_c * K_p * e(t) + D) \\
& \Rightarrow (c(t+1) \in r(t) \pm \max(f, |e(t)| * s)) \\
& \equiv \text{‘by eliminating } c(t+1)\text{ and using } K == K_c * K_p\text{’} \\
& (c(t) + K * e(t) + D) \in (r(t) \pm \max(f, |e(t)| * s))
\end{aligned} \tag{2}$$

Importantly, this step eliminates references to *different* moments of time in the same expression.

The proof proceeds by splitting the obligation into six distinct cases by using properties of the \pm , \max and $|\cdot|$ operators.

$$\begin{aligned}
(2) & \equiv \text{‘by eliminating } \pm\text{’} \\
& (r(t) - \max(f, |e(t)| * s) \leq c(t) + K * e(t) + D) \\
& \wedge (c(t) + K * e(t) + D \leq r(t) + \max(f, |e(t)| * s)) \\
& \equiv \text{‘by eliminating } \max\text{’} \\
& (r(t) - f \leq c(t) + K * e(t) + D) \\
& \vee (r(t) - |e(t)| * s \leq c(t) + K * e(t) + D) \\
& \wedge (c(t) + K * e(t) + D \leq r(t) + f) \\
& \vee (c(t) + K * e(t) + D \leq r(t) + |e(t)| * s) \\
& \equiv \text{‘by eliminating } |\cdot|\text{’} \\
& (r(t) - f \leq c(t) + K * e(t) + D) \\
& \vee (r(t) + e(t) * s \leq c(t) + K * e(t) + D) \\
& \vee (r(t) - e(t) * s \leq c(t) + K * e(t) + D) \\
& \wedge (c(t) + K * e(t) + D \leq r(t) + f) \\
& \vee (c(t) + K * e(t) + D \leq r(t) + e(t) * s) \\
& \vee (c(t) + K * e(t) + D \leq r(t) - e(t) * s) \\
& \equiv \text{‘by pairing occurrences of } c(t)\text{ and } r(t)\text{ and using } e(t) == r(t) - c(t)\text{’} \\
& ((1 - K) * e(t) \leq D + f) \\
& \vee (1 - K + s) * e(t) \leq D \\
& \vee (1 - K - s) * e(t) \leq D) \\
& \wedge (D \leq (1 - K) * e(t) + f) \\
& \vee D \leq (1 - K + s) * e(t) \\
& \vee D \leq (1 - K - s) * e(t)
\end{aligned} \tag{3}$$

Our goal now is to cancel occurrences of the time-dependent variable $e(t)$, in order to give a required property involving the symbolic constants only.

Consider the first conjunct in expression (3). To prove $A \vee B$ it is sufficient to prove $\neg A \Rightarrow B$. We therefore assume that the negation of the first disjunct holds. This can be written as follows, provided $K \neq 1$.

$$\frac{f + D}{1 - K} < e(t) \quad (4)$$

We then aim to use this to prove the third disjunct in expression (3), which will allow us to conclude that the entire first conjunct is true. The third disjunct can be rewritten as follows.

$$-D \leq e(t) * (K - (1 - s)) \quad (5)$$

We then make use of the following arithmetic property, which is true as long as $W \geq 0$.

$$X < Y \wedge Z \leq X * W \Rightarrow Z \leq Y * W$$

Unifying (4) with $X < Y$, yields $X = (f + D)/(1 - K)$ and $Y = e(t)$. Then unifying (5) with $Z \leq Y * W$ gives $Z = -D$ and $W = K - (1 - s)$, with postulate $K - (1 - s) \geq 0$. The general property therefore holds if the following expression, corresponding to $Z \leq X * W$, is true. Note that it does not involve $e(t)$.

$$(K - (1 - s)) * f \geq -D * s \quad (6)$$

If expression (6) is true of the design constants we may conclude that the entire first conjunct in expression (3) holds.

A similar procedure can be followed for the second conjunct. The negation of the first disjunct is assumed and can be used to establish the third disjunct as long as the following expression holds.

$$(K - (1 - s)) * f \geq D * s \quad (7)$$

Combining inequalities (6) and (7) gives us the following property (which also implies the aforementioned postulate).

$$(K_c * K_p - (1 - s)) * f \geq |D| * s \quad (8)$$

This result can be understood as follows. On the right, $|D| * s$ defines the amount of ‘disturbance’ remaining after a correction is made at the specified speed, in the worst case. On the left, the system gain $K_c * K_p$ determines the the actual amount of correction applied per unit of measured error. Term $1 - s$ is the equivalent worst-case specified correction (recall that s defines allowable error *remaining*, rather than correction made). Therefore $K_c * K_p - (1 - s)$ is a measure of the extra correction performed by the design, above that specified. Multiplying this by offset f gives us the amount of ‘extra’ correction applied when the output is as close to the reference point as it ever need get. This is the point at which the degree of correction being applied is a minimum. Thus, if the left-hand side is

at least as great as the right-hand side, the design is capable of overcoming the worst-case disturbance even when a minimal correction is being applied.

We must now consider whether equation (8) is sufficient when $K = 1$. In this case the equation simplifies as follows.

$$s * f \geq s * |D| \quad (9)$$

If $s > 0$ then we can divide by s to yield the following.

$$f \geq |D| \quad (10)$$

Together with $K = 1$, equation (10) is sufficient to satisfy equation (3) since it makes the first disjunct in each conjunct true. However, if $s = 0$ equation (9) is trivial and offers no help in attempting to establish equation (3). Therefore we must add the following constraint to allow for this case.

$$s = 0 \Rightarrow f \geq |D| \quad (11)$$

Intuitively, this tells us that when the system is required to correct the entire measured error in one time step, then the acceptable offset must be at least as great as the worst-case disturbance. This is a natural requirement, because $s = 0$ forces $K = 1$, and leaves no extra ‘capacity’ in the system gain for responding to disturbances, so a permanent error up to the size of the worst-case disturbance cannot be prevented. Thus, when equations (8) and (11) are true of the system constants, we can finally conclude that our design satisfies the specification!

The dots in Figure 2 show the actual behaviour of the design expressed by predicates *Process* and *Controller*, assuming the largest possible disturbance at all times. Starting from the arbitrary initial value, the output quickly approaches the reference point, but at time 4 the negative disturbance causes the output to pass it. However, by time 6 the system is applying a positive correction that cancels out the disturbance and equilibrium is achieved. When the reference point changes, the output again rapidly compensates until equilibrium with the negative disturbance is achieved. The actual behaviour (dots) satisfies the required behaviour (shaded area) because the system reacts to changes in the reference point more quickly than necessary, i.e., the actual system gain is better than the required speed of response, and because the offset is as great as the point at which the disturbance and correction cancel one another. Indeed, substituting the constants assumed in Figure 2 into equation (8) gives the following confirmation that the design behaves as required. (Since s is not zero, equation (11) follows trivially.)

$$(2 * \frac{1}{4} - (1 - \frac{3}{4})) * 6 \geq |-2| * \frac{3}{4} \equiv \frac{3}{2} \geq \frac{3}{2}$$

5 Directions for Future Work

Despite the success of this case study, much more work is required to determine how non-trivial real-time applications can be handled in Cogito. For instance, we

have not yet introduced the ‘interval’ operators needed for efficiently expressing properties of real-time systems [Olderog et al., 1996, §5.2.3] [Millerchip et al., 1993, §1.1]. However, it is expected that such operators *can* be introduced to Cogito because Sum encompasses the Z specification language, and the interval notations have Z-based representations.

Although a simple approach, using logic operators to compose specifications and define refinements, as was done in Section 3, places the burden of ensuring that the system is correctly structured on the programmer. Ideally, Cogito should be customised to explicitly recognise real-time specifications, so that syntactic checks can be performed automatically. For instance, introducing ‘frames’ as an explicit part of the specification language [Scholefield et al., 1994, p. 224] [Mahony, 1992, p. 7], would allow the Cogito tools to check that ‘assumption’ predicates do not refer to ‘controlled’ variables [Mahony, 1992, p. 7].

It was clear from the complexity involved in even this simple example that manipulating difference equations will be difficult for non-trivial applications. Ultimately, the effort was a combination of tool assistance and human ingenuity, with a global proof strategy already outlined before the theorem prover was used. One solution may be to follow the approach adopted for process control and implement a theory of *transfer functions* [Bollinger and Duffie, 1988, §2.4] [Leigh, 1992, ch. 4] to provide a higher-level reasoning capability.

The proof required axioms and laws not needed in the non-real-time projects previously undertaken in Cogito and thus revealed areas where Ergo’s theory base needs further population. It quickly became obvious that the theorem prover is currently underpopulated for reasoning about arithmetic and inequalities. Similarly, Ergo’s ability to reason about functions will need to be improved for more complex ‘timed trace’ examples. (We overcame this above by taking advantage of this example’s single time-unit delay, which allowed us to quickly eliminate consideration of variable values at different times from the proof.)

The exercise also suggested other tools that could be profitably added to the Cogito toolkit. For instance, the example was formulated in such a way that it could equally well have been solved using constraint analysis techniques. We also made considerable use of a simple simulator [Bollinger and Duffie, 1988, p. 118] early in the case study—the trace in Figure 2 was generated by this program. Although no substitute for a formal proof, this simple tool was invaluable in gaining an intuitive understanding of the system’s dynamic behaviour. (Cogito’s existing animation tool could not be used for the real-time case study because it works with state-machine specifications only.)

6 Conclusion

Although optimised for state-machine specifications, we have shown that the Cogito environment can also be applied to ‘timed-trace’ system design. This is a testament to the generality and flexibility of Cogito, and the Sum language and Ergo theorem prover in particular. An especially pleasing outcome was the

discovery of the relationships between the various design constants, expressed by inequalities (8) and (11), which were revealed only during the formal proof.

Acknowledgements We wish to thank Peter Robinson for help with the proof, and Ian Hayes, Axel Wabenhorst and the anonymous referees for correcting errors in the paper. This work was funded in part by the Information Technology Division of the Defence Science and Technology Organisation.

References

- Abrial, J.-R. (1996). *The B-Book: Assigning Programs to Meanings*. Cambridge University Press.
- Bloesch, A., Kazmierczak, E., Kearney, P., Staples, J., Traynor, O., and Utting, M. (1996). A formal reasoning environment for Sum—a Z based specification language. *Australian Computer Science Communications*, 18(1):149–158.
- Bloesch, A., Kazmierczak, E., Kearney, P., and Traynor, O. (1995). Cogito: A methodology and system for formal software development. *International Journal of Software Engineering and Knowledge Engineering*, 5(4):599–617.
- Bollinger, J. G. and Duffie, N. A. (1988). *Computer Control of Machines and Processes*. Addison-Wesley.
- Cant, A., Eastaughffe, K., and Ozols, M. (1996). A tool for practical reasoning about state machine designs. In *Proc. 1996 Australian Software Engineering Conference*, pages 16–26, Melbourne.
- Hart, T., Linn, F., Morello, R., Royle, G., Kearney, P., Lindsay, P., Ross, K., and Traynor, O. (1996). Formal methods in an industrial pilot project. In *Asia-Pacific Software Engineering Conference*, Seoul, Korea.
- Holzbock, W. G. (1958). *Automatic Control: Principles and Practice*. Reinhold.
- Kazmierczak, E., Kearney, P., Traynor, O., and Wang, Li (1995). A modular extension to Z for specification, reasoning and refinement. Technical Report 95-15, Software Verification Research Centre, University of Queensland.
- Leigh, J. R. (1992). *Control Theory: A Guided Tour*, volume 45 of *IEE Control Engineering Series*. Peter Peregrinus Ltd.
- Mahony, B. P. (1992). The refinement calculus and data-flow processes. In *Proc. Second Australasian Refinement Workshop*, pages 1–28, Brisbane.
- Millerchip, C., Mahony, B., and Hayes, I. J. (1993). The generic problem competition: A whole system specification of the boiler system. Software Verification Research Centre, University of Queensland.
- Olderog, E.-R., Ravn, A. P., and Skakkebæk, J. U. (1996). Refining system requirements to program specifications. In Heitmeyer, C. and Mandrioli, D., editors, *Formal Methods for Real-Time Computing*, volume 5 of *Trends in Software*, chapter 5, pages 107–134. Wiley.
- Scholefield, D., Zedan, H., and He Jifeng (1994). A specification-oriented semantics for the refinement of real-time systems. *Theoretical Computer Science*, 131:219–241.
- Wordsworth, J. B. (1992). *Software Development with Z*. Addison-Wesley.

This article was processed using the \LaTeX macro package with LLNCS style