Algebraic Models of Correctness for Microprocessors

Gennady Shmonin
scorn@wjpserver.cs.uni-sb.de

Abstract

We describe an algebraic approach to the verification of microprocessors according to its description in [1] by Anthony C. J. Fox. Recently, this work was continued with the verification of the complete microprocessor ARM6 [2]. Microprocessors are modelled using *iterated maps*, defined by equations which evolve a system from an initial state by the iterative application of a *next-state* function. Levels of temporal abstraction are related by temporal abstraction maps called *retimings*. We state the basic *correctness condition* for microprocessors. We introduce *one-step theorems* that permit verification of correctness conditions to be considerably simplified under well-defined conditions.

Introduction

This report is based on the paper [1] of Anthony C. J. Fox and concerns the verification of microprocessors. We consider an algebraic approach for the verification. First, we describe how the specification and implementation of a microprocessors can be specified as many-sorted algebras which include a clock, a state-space, and an iterated map, i.e., a state function. We consider only deterministic state evolutions, thus the iterated map and the initial state of the system completely determine full sequence of system's states.

Then we establish the relation between these algebras, which is described by data abstraction and temporal abstraction. The data abstraction turns out to be temporal invariant, the temporal abstraction is represented by the map called retiming.

These notions allow us to formally define the implementation correctness. The general definition is difficult to be checked, therefore we treat some restrictions. Defining the notions of uniformity of retimings and time-consistency of iterated maps lead to the simplified way for verification. This approach is formalised in so-called one-step theorems. We should note that these additional requirements (uniformity and time-consistency) seem to be very practical and are satisfied for most real models.

1 Equational Specification of Computing Systems

Computer systems are modelled as a sequence of states evolving in time, that is a state transforming systems, for example,

$$s_0 \to s_1 \to s_2 \to \dots$$

where s_i represents the i-th state of the architecture.

We consider only deterministic state evolutions. This means that any given state will always produce the same next state. Under this restriction, if the state recurs then the system will loop forever, that is we get a cyclic state evolution. Obviously, any system with the finite number of states produce a cyclic state evolution in the infinite time.

We are interested in two different levels of abstraction for computer systems.

- The *programmer's model (PM)* is the functional specification of a microprocessor's architecture (instruction-level). The PM specifies the semantics of the machine's operations with respect to the memory and registers visible to a programmer.
- The *abstract circuit model (AC)* is the functional specification of a microprocessor's organisation (micro-architecture). The AC specifies the functionality of high-level components (for example, a decode unit and the I/O interface of a microprocessor).

Computer systems can be modelled in an algebraic framework using primitive recursive functions. Primitive recursive functions are defined, with equations, using the constructions: substitution, bounded minimalisation and quantification, definition by cases and primitive recursion [1]. Time is modelled using a *clock algebra* and state transforming systems are modelled using *state algebras*. This specification framework is flexible enough to specify both the PM and the AC levels of abstraction.

Generally, a *many-sorted algebra* consists of *carrier sets* and *functions* ranging over the carrier sets:

$$(A_1, A_2, \ldots, A_l \mid f_1, f_2, \ldots, f_m)$$

where

- $A_1, A_2, \ldots, A_l$ are carrier sets, and
- $f_1, f_2, \ldots, f_m$ are functions of the form

  $$f_i : A_{s_1} \times A_{s_2} \times \ldots \times A_{s_n} \to A_s$$

  with $1 \leq i \leq m$ and $1 \leq s, s_j \leq l$ for $1 \leq j \leq n$. The value n is then called the *arity* of function f_i. If $n = 0$ then f_i if called the *constant*.

Now we formally define clocks and iterated maps.

Definition 1
A **clock** *is an algebra* $(T \mid 0, +1)$ *where*

- T *is a set of* **clock cycles**,
- $0 \in T$ *is the* **initial clock cycle**,
- $+1 : T \to T$ *is the next (or successor) clock cycle function.*

Definition 2
Let T *be a clock and* S *be any non-empty set. An* **iterated map** $F : T \times S \to S$ *is a primitive recursive function defined for* $s \in S$ *and* $t \in T$ *by the equations*

$$\begin{aligned} F(0, s) &= s, \\ F(t+1, s) &= f(F(t, s)) \end{aligned}$$

where $f : S \to S$ is a function called the **next-state function** *of the iterated map F. The set S is called the $state - space$ of F.*

A system starts at time $0 \in T$ in an initial state $s_0 \in S$; subsequent states are enumerated over the clock T and evaluated by an iterated map state function $F : T \times S \to S$.

The iterated map can be represented using the next-state function that is formalised in the following fact.

Fact 1
If $F : T \times S \to S$ is an iterated map with next-state function $f : S \to S$ then for all $s \in S$ and $t \in T$

$$F(t, s) = f^t(s).$$

We also consider initialised state sequences. The sequence of states is initialised with an application of the function $h : S \to S$. The function h constricts the number of possible state sequences, and this enables a more flexible definition of correctness. It is feasible for a system to be correct provided it starts in a valid state.

Definition 3
An **initialised iterated map** *$F : T \times S \to S$ is a primitive recursive function defined for $s \in S$ and $t \in T$ by the equations*

$$\begin{aligned} F(0, t) &= h(t), \\ F(t+1, s) &= f(F(t, s)). \end{aligned}$$

where $h : S \to S$ is a function called the **initialisation function** *of the iterated map F, and $f : S \to S$ is the* **next-state function** *of F.*

The representation through the next-state function is also possible.

Fact 2
If $F : T \times S \to S$ is an initialised iterated map with initialisation function $h : S \to S$ and next-state function $f : S \to S$, then for all $s \in S$ and $t \in T$

$$F(t, s) = (h \bullet f^t)(s) = f^t(h(s)).$$

Note that we define the composition $f \bullet g$ by the equation

$$f \bullet g(s) = g(f(s)).$$

2 Abstraction Mechanisms

Data and temporal abstractions are used to relate two algebras. Given algebras

$$\begin{gathered} (T, S \mid F : T \times S \to S), \\ (T', S' \mid F' : T' \times S' \to S') \end{gathered}$$

where T and T' are clocks, F and F' are iterated maps with state-spaces S and S', respectively, a map $\Psi : T' \times S' \to T \times S$ is used to abstract the clock and state-space of F' into the clock and state-space of F. When considering PM and AC specifications, S represents the state-space of the PM and S' represents the state-space of the AC. Hence, it is expected that $\mid S \mid \leq \mid S' \mid$.

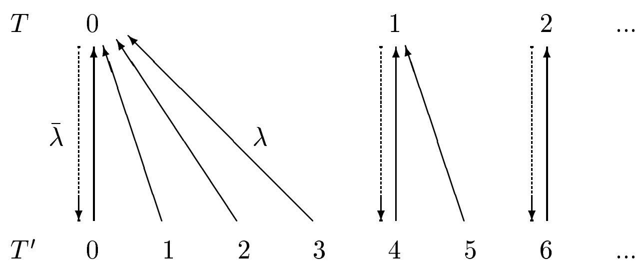

Figure 1: A retiming $\lambda \in Ret(T', T)$ with associated immersion $\bar{\lambda} \in Imm(T', T)$. The clock carrier sets T' and T are assumed to be the natural numbers.

The map Ψ is split into temporal and data mappings

$$\lambda : T' \times S' \rightarrow T,$$
$$\psi : T' \times S' \rightarrow S$$

respectively. That is to say, a vector function Ψ is defined by $\Psi = (\lambda, \psi)$.

In most cases data abstraction is temporally invariant. Therefore we regard **data abstraction** as a surjective mapping between state-spaces:

$$\psi : S' \rightarrow S$$

To represent temporal abstraction, we define two functions: *retiming* and *immersion*. Two clocks are related using a temporal abstraction map, or retiming.

Definition 4
A **retiming** $\lambda : T' \rightarrow T$ *is a surjective and monotonic map between clocks* T' *and* T *for which* $\lambda(0) = 0$. *The set of all such retimings is denoted by*

$$Ret(T', T)$$

and this is a subset of $[T' \rightarrow T]$ *which denotes the set of all maps form* T' *to* T.

Definition 5
The **immersion** $\bar{\lambda} : T \rightarrow T'$ *of a retiming* $\lambda \in Ret(T', T)$ *is defined by*

$$\bar{\lambda}(t) = \min\{t' \in T' : \lambda(t') = t\}.$$

The set of all immersions of retimings in $Ret(T', T)$ *is denoted by*

$$Imm(T', T).$$

Figure 1 illustrates the relationship between a retiming λ and its associated immersion $\bar{\lambda}$.

The following two definitions are illustrated with examples in Figures 2 and 3.

Definition 6
The **start function** *start* : $Ret(T', T) \rightarrow [T' \rightarrow T']$ *is defined by the equation*

$$start(\lambda) = (\lambda \cdot \bar{\lambda}).$$

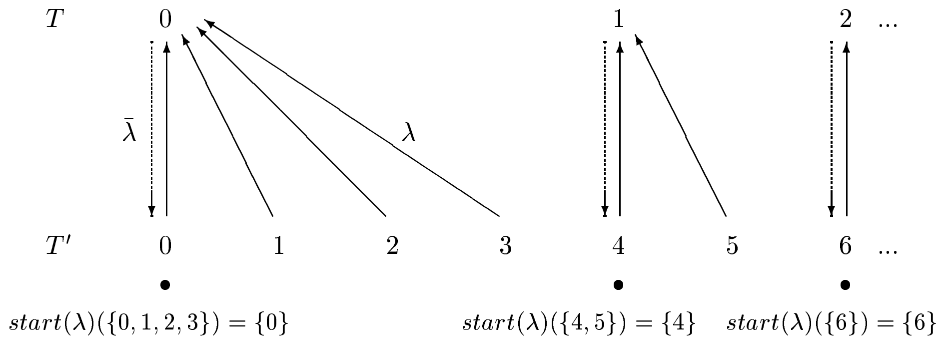

Figure 2: The function $start : Ret(T', T) \to [T' \to T]$.

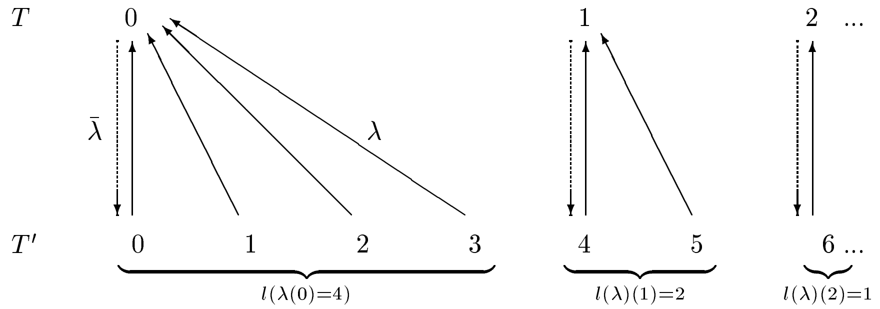

Figure 3: The function $length : Ret(T', T) \to [T \to T'^{+}]$.

Definition 7
The **length function** $l : Ret(T', T) \to [T \to T']$ *is defined by the equation*

$$l(\lambda)(t) = \bar{\lambda}(t+1) - \bar{\lambda}(t).$$

In the context of a retiming $\lambda \in Ret(T', T)$, clock T' is called *faster* than clock T (and clock T is *slower* than clock T'). A clock T' is *strictly faster* than clock T if T' is faster than T and there exists a time $t \in T$ such that $l(\lambda)(t) > 1$. Clocks T' and T are at the *same speed* if T' is faster than T but not strictly faster.

Data abstraction induces temporal abstraction. Consider the following sequence of states

$$s'_0 \to s'_1 \to s'_2 \to s'_3 \to s'_4 \to s'_5 \to \ldots$$

If the states of this sequence are abstracted using the mapping

$$s'_0, s'_5 \mapsto s_0,$$
$$s'_1, s'_2, s'_3, s'_4 \mapsto s_1$$

then the abstracted version of the original sequence is

$$s_0 \to s_1 \to s_1 \to s_1 \to s_1 \to s_0 \to \ldots$$

This sequence is observed as

$$s_0 \to s_1 \to s_0 \to \ldots$$

The abstracted state system defines a new measure of time because there are only two observable state changes. Note that in case the initial state is s'_1, we obtain the different measure of time.

Thus, retimings, and consequently immersions, should be defined relative to state transition. If state evolution is deterministic then one state is sufficient to define an entire temporal abstraction map.

Definition 8
A state-dependent retiming $\lambda : S' \to Ret(T', T)$ *is a map from a state-space* S' *to retimings in* $Ret(T', T)$*. The set of all such state-dependent retiming is denoted by*

$$Ret(S', T', T).$$

Definition 9
The state-dependent immersion $\bar{\lambda} : S' \to Imm(T', T)$ *of a state-dependent retiming* $\lambda \in Ret(S', T', T)$ *is defined by the equation*

$$\bar{\lambda}(s')(t) = \min\{t' \in T' : \lambda(s')(t') = t\}.$$

The set of all such state-dependent immersions is denoted by

$$Imm(S', T', T).$$

Consider a state system with state-space S' defined over the clock T' by the iterated map $F' : T' \times S' \to S'$. If the state system is abstracted using the data abstraction map $\psi : S' \to S$ then the natural definition for a space-dependent temporal abstraction map $\lambda : S' \to [T' \to T]$ is

$$\begin{array}{rcl} \lambda(s')(0) & = & 0, \\ \lambda(s')(t'+1) & = & \left\{ \begin{array}{ll} \lambda(s')(t'), & \text{if } \psi(F'(t', s')) = \psi(F'(t'+1, s')), \\ \lambda(s')(t') + 1, & \text{if } \psi(F'(t', s')) \neq \psi(F'(t'+1, s')). \end{array} \right. \end{array}$$

For all times greater than zero there are two cases:

- The state (after abstraction) of the system at cycle $t' + 1 \in T'$ is the same as the state (after abstraction) at cycle $t' \in T'$. In this case, the time on the abstract clock is held at $\lambda(s')(t') \in T$.
- The state of the system (after abstraction) has changed. In this case, the time of the abstract clock is incremented by one, to give $\lambda(s')(t') + 1$.

3 Implementation Correctness

Correctness is established by comparing the operational semantics of an *implementation* with the operational semantics of a *requirements specification*. For example, the state function of an AC design is compared with state function of a PM specification.

Consider two state sequences:

$$\begin{array}{l} s_0 \to s_1 \to s_2 \to s_3 \to s_4 \to \dots \\ s'_0 \to s'_1 \to s'_2 \to s'_3 \to s'_4 \to \dots \end{array}$$

where s_i represents the i-th state of the specification and s'_i represents the i-th state of the implementation. To establish correctness, the state sequences are related

$$\begin{tikzcd}
F'(\bar{\lambda}(s')(0), s') \arrow[r, "\psi"] \arrow[d, "(f')^{l(\lambda(s'))(0)}"] & F(0, \psi(s')) \arrow[d, "f"] \\
F'(\bar{\lambda}(s')(1), s') \arrow[r, "\psi"] \arrow[d, "(f')^{l(\lambda(s'))(1)}"] & F(1, \psi(s')) \arrow[d, "f"] \\
F'(\bar{\lambda}(s')(2), s') \arrow[r, "\psi"] \arrow[d, "(f')^{l(\lambda(s'))(2)}"] & F(2, \psi(s')) \arrow[d, "f"] \\
F'(\bar{\lambda}(s')(3), s') \arrow[r, "\psi"] \arrow[d, dotted] & F(3, \psi(s')) \arrow[d, dotted] \\
{} & {}
\end{tikzcd}$$

Figure 4: Implementation correctness.

using a data abstraction map ψ and a temporal abstraction map λ. The data abstraction map, which is temporally invariant, relates the two state-spaces; for correctness, the equivalence $s_0 = \psi(s'_0)$ is initially required. The retiming λ specifies when further state comparisons are made. For each time i the equivalence $s_i = \psi(s'_j)$ is required, where i is related to j by retiming. In the context of iterated map specifications, the correctness criteria outlined above is expressed in the form of a commuting diagram illustrated in Figure 4.

The number of cycles between observed implementation states is given by the length function. For example, is F' is a micro-programmed AC implementation of a PM architecture, then $l(\lambda(s'))(n)$ represents the number of cycles required for the abstract circuit to execute the $(n+1)$-th PM level instruction having already executed n PM level instructions from the initial processor state s'.

Definition 10
A state function $F' : T' \times S' \to S'$ is a **correct implementation** *of the state function $F : T \times S \to S$ with respect to state-dependent retiming $\lambda \in Ret(S', T', T)$ and data abstraction map $\psi : S' \to S$ if, and only if,*

$$F(t, \psi(s')) = \psi(F'(\bar{\lambda}(s')(t), s'))$$

for all $s' \in S'$ and $t \in T$.

In many cases, correctness criteria can be simplified to assist automated verification. The simplification process is validated by series of *one-step theorems* which conditionally reformulate correctness statements. Conditions are placed on iterated maps and retimings introducing the notions of *time-consistency*, for iterated maps, and *uniformity*, for retimings.

4 Time-Consistency

We define two types of time-consistency — one suitable for specifications and the other suitable for implementations. The time-consistency of the implementation is established with respect to the states of the implementation enumerated by the immersion of a retiming.

$$\begin{tikzcd}
s \arrow[r] \arrow[d] & s_1 = F(t_1, s) \arrow[d] \\
F(t_1 + t_2, s) \arrow[r, equal] & F(t_2, s_1)
\end{tikzcd}$$

Figure 5: Commuting diagram for time-consistent iterated maps.

$$\begin{tikzcd}
s' \arrow[r] \arrow[d] & s'_1 = F'(\bar{\lambda}(s')(t_1), s') \arrow[d] \\
F'(t'_1 + t'_2, s') \arrow[r, equal] & F'(\bar{\lambda}(s'_1)(t_2), s'_1)
\end{tikzcd}$$

Figure 6: Commuting diagram for iterated maps time-consistent with respect to a state-dependent retiming.

Definition 11
An iterated map $F : T \times S \to S$ is **time-consistent** *if, and only if, for all $s \in S$ and $t_1, t_2 \in T$*

$$F(t_1 + t_2, s) = F(t_2, F(t_2, s)).$$

Definition 12
An iterated map $F' : T' \times S' \to S'$ is **time-consistent** *with respect to a state-dependent retiming $\lambda \in Ret(S', T', T)$ if, and only if, for all $s' \in S'$ and $t_1, t_2 \in T$*

$$F'(t'_1 + t'_2, s') = F'(t'_2, s'_1),$$

where $t'_1 = \bar{\lambda}(s')(t_1)$, $s'_1 = F'(t'_1, s')$, and $t'_2 = \bar{\lambda}(s'_1)(t_2)$.

These definitions correspond to the commuting diagrams shown in Figures 5 and 6.

Lemma 1
All non-initialised iterated maps are time-consistent.

Proof. Let $F : T \times S \to S$ be a non-initialised iterated map with next-state function $f : S \to S$. Fact 1 states that $F(t, s) = f^t(s)$, therefore

$$\begin{aligned}
F(t_1 + t_2, s) &= f^{t_1 + t_2}(s) \\
&= f^{t_2}(f_1^t(s)) \\
&= f^{t_2}(F(t_1, s)) \\
&= F(t_2, F(t_1, s)).
\end{aligned}$$

□

Similarly, we can prove the following lemma.

Lemma 2
All non-initialised iterated maps are time-consistent with respect to an arbitrary state-dependent retiming λ.

Having shown that all iterated maps without initialisation function are always time-consistent it remains to examine the two cases with initialisation functions.

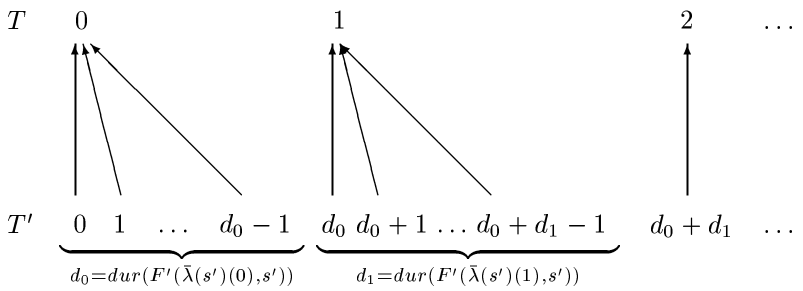

Figure 7: The construction of a uniform retiming defined with respect to the duration map *dur* and iterated map F'.

Lemma 3
Let $F : T \times S \to S$ be an iterated map with initialisation function $h : S \to S$. The map F is time-consistent if, and only if, for all $s \in S$ and $t \in T$

$$F(t, s) = h(F(t, s)).$$

Proof. Sufficiency. Fact 2 states that $F(t, s) = (h \bullet f^t)$ where $f : S \to S$ is the next-state function of the map F, therefore

$$\begin{aligned} F(t_1 + t_2, s) &= f^{t_1+t_2}(h(s)) \\ &= f^{t_2}(f^{t_1}(h(s))) \\ &= f^{t_2}(F(t_1, s)) \\ &= f^{t_2}(h(F(t_1, s))) \\ &= F(t_2, F(t_1, s)). \end{aligned}$$

Necessity. If there exist $s \in S$ and $t \in T$ such that $F(t, s) \neq h(F(t, s))$ then

$$F(0 + t, s) \neq F(0, F(t, s)).$$

□

The following lemma state the similar condition for time-consistency with respect to a state-dependent retimings for initialised iterate maps. It can be proved analogously to Lemma 3.

Lemma 4
Let $F' : T' \times S' \to S'$ be an iterated map with initialisation function $h : S' \to S'$. The map F' is time-consistent with respect to $\lambda \in Ret(S', T', T)$ if, and only if, for all $s' \in S'$ and $t \in T$

$$F'(\bar{\lambda}(s')(t), s') = h(F'(\bar{\lambda}(s')(t), s')).$$

5 Uniform Retimings

The one-step theorem requires a strong connexion between all the states generated by an iterated map F from state s and the one retiming associated with the state s. This connexion is achieved by associating a *duration* with each state of the map F. The duration for a state s represents the number of clock cycles required for the implementation F to produce the next correct state from the implementation state s'. Time-consistency ensures all observed states may be considered initial and uniformity ensures that temporal abstraction is not affected by re-initialisation because there is only ever one duration associated with a given state.

The construction of a uniform retiming is recursive, see Figure 7.

Definition 13
Let $F' : T' \times S' \to S'$ be an iterated map and $dur : S' \to T'$ be a map from states to a positive number of clock cycles. The **uniform retiming** *with respect to F' and dur is the state-dependent retiming $\lambda \in Ret(S', T', T)$ such that, for all $s \in S'$ and $t \in T$*

$$\begin{aligned} \bar{\lambda}(s')(0) &= 0, \\ \bar{\lambda}(s')(t+1) &= dur(F'(\bar{\lambda}(s')(t), s')) + \bar{\lambda}(s')(t) \end{aligned}$$

where $\bar{\lambda} \in Imm(S', T', T)$ is the immersion of λ. The singleton set containing the uniform retiming with respect to F' and dur is denoted by

$$URet^{dur}_{F'}(S', T', T).$$

Definition 14
A state dependent retiming $\lambda \in Ret(S', T', T)$ is **uniform** *with respect to F' if, and only if, there exists a function dur such that $\lambda \in URet^{dur}_{F'}(S', T', T)$. The set of all uniform retimings with respect to F' is denoted by*

$$URet_{F'}(S', T', T).$$

Lemma 5
If $\lambda \in URet_{F'}(S', T', T)$ and the iterated map $F' : T' \times S' \to S'$ is time-consistent with respect to λ, then for all $s' \in S'$ and $t \in T$

$$dur(F'(\bar{\lambda}(s')(t), s')) = \bar{\lambda}(F'(\bar{\lambda}(s')(t), s'))(1).$$

Proof. Let $h : S' \to S'$ be the initialisation function for F'. We denote $t' = \bar{\lambda}(s')(t)$ and $s'_1 = F'(t', s')$. Then, by 4

$$\begin{aligned} dur(F'(t', s')) &= dur(h(F'(t', s'))) \\ &= dur(h(s'_1)) \\ &= dur(F'(0, s'_1)). \end{aligned}$$

By definition of λ:

$$\begin{aligned} dur(F'(t', s')) &= dur(F'(0, s'_1)) \\ &= dur(F'(\bar{\lambda}(s'_1)(0), s'_1)) + \bar{\lambda}(s'_1)(0), \end{aligned}$$

whence by uniformity of λ, we get

$$dur(F'(t', s')) = \bar{\lambda}(s'_1)(1).$$

This completes the proof.

□

Combining Lemma 5 with Definition 13, we immediately get the following useful corollary.

Corollary 6
If $\lambda \in URet_{F'}(S', T', T)$ and the iterated map $F' : T' \times S' \to S'$ is time-consistent with respect to λ, then for all $s' \in S'$ and $t \in T$

$$\bar{\lambda}(s')(t+1) = \bar{\lambda}(F'(\bar{\lambda}(s')(t), s'))(1) + \bar{\lambda}(s')(t).$$

6 One-Step Theorems

The notions of time-consistency of iterated maps and uniformity of retimings allow us to develop simplified techniques for verifying the implementation correctness. The theorems presented in this section state this formally.

The first theorem demonstrates that, given the right conditions, time-consistency can be verified in one (specification) clock cycle. The second theorem states that implementation correctness can also be verified in one clock cycle.

Theorem 1
Let $F' : T' \times S' \to S'$ be an iterated map with initialisation function $h : S' \to S'$ and let $\lambda \in URet_{F'}(S', T', T)$ be a uniform retiming. For all $s' \in S'$ and $t \in T$

$$F'(\bar{\lambda}(s')(t), s') = h(F'(\bar{\lambda}(s')(t), s'))$$

if, and only if

$$F'(\bar{\lambda}(s')(0), s') = h(F'(\bar{\lambda}(s')(0), s')) \tag{1}$$

and

$$F'(\bar{\lambda}(s')(1), s') = h(F'(\bar{\lambda}(s')(1), s')) \tag{2}$$

Proof. Necessity is obvious. The proof of sufficiency is by induction over the clock T.

The claim for $t = 0$ follows immediately by the theorem's condition (Equation (1)).

Assume $F'(\bar{\lambda}(s')(t), s') = h(F'(\bar{\lambda}(s')(t), s'))$ for some $t \in T$. We have to show that this equation holds for $t + 1 \in T$, i.e, that

$$F'(\bar{\lambda}(s')(t+1), s') = h(F'(\bar{\lambda}(s')(t+1), s')).$$

Let $f : S' \to S'$ be the next-state function of F and let $dur : S' \to T'$ be a map such that $\lambda \in URet^{dur}_{F'}(S', T', T)$.

First of all, we show that the claim similar to Corollary 6 holds, i.e.,

$$\bar{\lambda}(s')(t+1) = \bar{\lambda}(F'(\bar{\lambda}(s')(t), s'))(1) + \bar{\lambda}(s')(t). \tag{3}$$

We denote $t' = \bar{\lambda}(s')(t)$. By induction hypothesis,

$$dur(F'(t', s')) = dur(h(F'(t', s'))).$$

Thus, we can proceed as in the proof of Lemma 5:

$$\begin{aligned} dur(F'(t', s')) &= dur(h(F'(t', s'))) \\ &= dur(F'(0, F'(t', s'))) \\ &= dur(F'(\bar{\lambda}(F'(t', s'))(0), F'(t', s'))) + \bar{\lambda}(F'(t', s'))(0) \\ &= \bar{\lambda}(F'(t', s'))(1). \end{aligned}$$

Since λ is a uniform retiming, Equation (3) holds.

Further, we denote $s'_1 = F'(\bar{\lambda}(s')(t), s') = F'(t', s')$. Using Equation (3), we write:

$$F'(\bar{\lambda}(s')(t+1), s') = F'(\bar{\lambda}(s'_1)(1) + \bar{\lambda}(s')(t), s').$$

By induction hypothesis and Fact 2:

$$\begin{aligned} F'(\bar{\lambda}(s')(t+1), s') &= F'(\bar{\lambda}(s'_1)(1) + \bar{\lambda}(s')(t), s') \\ &= f^{\bar{\lambda}(s'_1)(1) + \bar{\lambda}(s')(t)}(h(s')) \\ &= f^{\bar{\lambda}(s'_1)(1)}(f^{\bar{\lambda}(s')(t)}(h(s'))) \\ &= f^{\bar{\lambda}(s'_1)(1)}(F'(\bar{\lambda}(s')(t), s')) \\ &= f^{\bar{\lambda}(s'_1)(1)}(h(F'(\bar{\lambda}(s')(t), s'))), \end{aligned}$$

whence

$$F'(\bar{\lambda}(s')(t+1), s') = F'(\bar{\lambda}(s'_1)(1), s'_1). \quad (4)$$

For the last expression, Equation (2) can be applied, that results in

$$\begin{aligned} F'(\bar{\lambda}(s')(t+1), s') &= F'(\bar{\lambda}(s'_1)(1), s'_1) \\ &= h(F'(\bar{\lambda}(s'_1)(1), s'_1)) \\ &= h(F'(\bar{\lambda}(s')(t+1), s')). \end{aligned}$$

□

Corollary 7
Let $F : T \times S \to S$ be an iterated map with initialisation function $h : S \to S$. For all $s \in S$ and $t \in T$

$$F(t, s) = h(F(t, s))$$

if, and only if

$$F(0, s) = h(F(0, s))$$

and

$$F(1, s) = h(F(1, s)).$$

Proof. The claim follows from Theorem 1 with $\bar{\lambda}(s)(t) = t$ for all $s \in S$.

□

The main theorem of this report is following.

Theorem 2
Let $F : T \times S \to S$ and $F' : T' \times S' \to S'$ be iterated maps. Let $\lambda \in URet_{F'}(S', T', T)$ be a uniform retiming and let $\psi : S' \to S$ be a data abstraction map. If

1. *F is time-consistent, and*
2. *F' is time-consistent with respect to λ*

then for all $s' \in S'$ and $t \in T$

$$F(t, \psi(s')) = \psi(F'(\bar{\lambda}(s')(t), s'))$$

if, and only if

$$F(0, \psi(s')) = \psi(F'(\bar{\lambda}(s')(0), s')) \quad (5)$$

and

$$F(1, \psi(s')) = \psi(F'(\bar{\lambda}(s')(1), s')). \quad (6)$$

Proof. Necessity is obvious. The proof of sufficiency is by induction over clock T.

Time $t = 0$ is given by Equation (5).

Assume $F(t, \psi(s')) = \psi(F'(\bar{\lambda}(s')(t), s'))$ for some time $t \in T$. We have to prove this equality for $t \in T$, namely, we need to show:

$$F(t+1, \psi(s')) = \psi(F'(\bar{\lambda}(s')(t+1), s')).$$

Since F is time-consistent, we have

$$F(t+1, \psi(s')) = F(1, F(t, \psi(s'))),$$

that can be rewritten using induction hypothesis:

$$F(t+1, \psi(s')) = F(1, \psi(F'(\bar{\lambda}(s')(t), s'))).$$

For simplicity, we denote as usual $t' = \bar{\lambda}(s')(t)$ and $s'_1 = F'(t', s')$. We can apply Equation (6) for the latter equation, that gives

$$\begin{aligned} F(t+1, \psi(s')) &= F(1, \psi(s'_1)) \\ &= \psi(F'(\bar{\lambda}(s'_1)(1), s'_1)). \end{aligned}$$

According to time-consistency of F':

$$\begin{aligned} F(t+1, \psi(s')) &= \psi(F'(\bar{\lambda}(s'_1)(1), s'_1)) \\ &= \psi(F'(\bar{\lambda}(s'_1)(1) + t', s')) \\ &= \psi(F'(\bar{\lambda}(F'(t', s'))(1) + t', s')). \end{aligned}$$

By Corollary 6, we have

$$\bar{\lambda}(s')(t+1) - \bar{\lambda}(s')(t) = \bar{\lambda}(F'(t', s'))(1),$$

and therefore,

$$F(t, \psi(s')) = \psi(F'(\bar{\lambda}(s')(t), s')).$$

□

7 Conclusion

In this report, we introduce an algebraic framework through which the correctness of computer organisations can be modelled. Computer systems are modelled at two levels, the programmer's model level PM and the abstract circuit model AC. specifications are presented in the form of a state algebra $(T, S \mid F)$ where T is a clock, S is a state-space and $F : T \times S \to S$ is a state function. An explicit model of time is used; the initial state of a system is defined using an initialisation function, and a measure of time is defined with respect to the iteration of a next-state function.

Computer systems are verified using one-step theorems. These theorems require satisfying some additional conditions: retimings representing the temporal abstraction must be uniform and the state function must be time-consistent.

The paper [2] shows how this framework can be applied for the verification of the concrete microprocessor.

References

[1] A. C. J. Fox. Algebraic Models for Advanced Microprocessors. PhD Thesis, University of Wales, Swansea, 1998

[2] A. C. J. Fox. Formal Verification of the ARM6 micro-architecture. Technical Report No. 548, University of Cambridge, Computer Laboratory, November, 2002