PROJECT DESCRIPTION

1 System design with behavior tables

This project investigates *behavior tables* as a representation for interactive system design. Behavior tables are a consolidation of notations used traditionally in engineering, such as register transfer tables and decision tables. Boolean function tables, look-up tables, and even fuse maps, might be considered simple instances. Thus, the tables themselves are hardly a new discovery, although we are introducing novel syntactic features. What is new is the level of supporting automation now available through graphics interface packages. With this advancing technology, tabular representations are being recognized as perspicuous in applications ranging from software requirements specification to hardware description languages (HDLs).

Tables are a natural intermediary between linear text and schematic diagrams. With better graphic support, we are convinced that behavior tables and their variants will assume a dominant role in system specification, verification, and synthesis. This project seeks to promote their adoption in practice by building a flexible graphics facility, to be integrated with design and verification tools.

We will build on results in *design derivation*, an algebraic formalism for high-level design based on transformation and refinement. Section 1.1 is a survey of these results. In case studies, we began using tables to visualize the transformation process, especially in stages where control and architecture are contemplated at the same time. We found the tables to be superior to the underlying formal syntax, even though we intimately understand the formalism. Similar experiences are reported elsewhere, as described in Section 1.2. In particular, a tool for decision tables developed by Hoover, Chen, and others, inspired this project. We could not express the motivation better than they do [16]:

> "...we must recognize that many of the methods used in industry are semi-formal or formal. For these methods, we must develop mathematical theory and software to help analyze specifications, and offer advice about good methodology. We expect that such efforts will give impressive near term results, stretch the imagination of formal methodists, make formal methods accessible and beneficial to system designers, and help clarify the long-term aims of formal methods research."

As a continuation of formal methods research, this project attacks the widely acknowledged problem that transfer to practice is impeded by the unpalatable notation of formal logic. The graphical user interface we develop will be integrated with our design derivation system to create a tool for high-assurance design. Provisions for animating analyses will be included, with the goal of visualizing verification and synthesis processes. Finally, we will formalize and implement more powerful transformations for higher levels of design (See Section 1.5).

However, the contributions of this project are broader than just advancing formal methods in system design. Behavior tables are better than conventional HDLs for specifying many kinds of systems, whether or not they are supported by a reasoning formalism. To promote their use in practice, we will integrate the graphic front end with existing simulation and synthesis tools and explore applications to codesign. Behavior tables offer a common ground for timely synergy among formal methods research, CAD research and development, and design practice. This central goal of this project is to enable that synergy.

1.1 Results from prior NSF support

NSF support for this line of research began in 1987 with the grant *Digital Design Derivation* (MIP87-07067, 1987–89, $205,972). Within the last five years, the work has continued under grants entitled *Algebra for Digital Design Derivation* (MIP89-21842, 1990–1992, $203,000) and *Decomposing digital-system specifications into interacting sequential processes* (MIP92-08745, 1993–1996, $142,000). This summary is restricted to accomplishments since 1991. Cited publications are included in the list of references on page 16. Some of them can be accessed through the World Wide Web at URL `http://www.cs.indiana.edu/hmg/hmg.html`.

Research contributions. We built a formal foundation for transformational design by adapting existing software oriented methodology to problems in digital system design. Theoretical contributions include representations of behavioral and structural specification in functional algebra [5], adaptation of algebraic techniques to architectural refinement [42, 43, 41, 45], design decomposition based on interface specification [34, 32, 33, 31], and formalization of stream semantics [28]. We have done elementary studies comparing how different formal systems address the same design [24].

We implemented our formalism in a tool called *DDD* [1, 2], which is used interactively to transform higher descriptions of behavior to synthesizable hardware descriptions. We did several case studies to demonstrate the approach, and, in particular, to explore and illustrate the thesis that heterogeneous reasoning is is more effective than a comprehensive logic in design-critical applications [2, 28, 39, 29, 6, 5, 4, 3].

The subject of this proposal, behavior tables, emerged as a useful representation for interactive design derivation. We began using these tables informally to describe DDD [23] and later for the presentation of case studies. This prompted us to consider them more seriously as a formal design notation, and we began exploring features that are appropriate in system design applications [40, 37, 36, 35, 30].

The work with behavior tables has germinated an interest in the fundamental role of diagrams in formal mathematics [10, 20, 11].

Other research products. The DDD system is now under commercial development, with SBIR funding through NASA, by Derivation Systems, Inc. [7]. DSI is a start-up company established by graduates of this research group.

Our case studies have resulted in several design artifacts. A formally derived version of Hunt's FM9001 microprocessor is realized in FPGA technology [2]. A language-specific computer for compiled Scheme, containing a derived CPU, garbage collector, and allocator [6], is an ongoing case study relating to contemporary programming methods research. A custom VLSI realization of a fault-tolerant clock synchronization circuit has been verified and tested [28, 29]. Miner has also constructed a PVS theory of streams, to be published as part of his doctoral thesis in early 1997.

Educational activities and human resource development. Supported participants include graduate assistants Zheng Zhu (PhD, 1992), Bhaskar Bose (PhD, 1994, also a NASA Fellow), Kamlesh Rath (PhD, 1995), Mustafa Esen Tuna (ABD), Shyamsundar Pullela (inactive), Jeanette Calvert-Coffren (inactive), Lowell Vaughn (inactive), and John Zuckerman (active pre-candidate). Unsupported participants included Kathryn Fisler (AT&T Fellow, PhD 1996) Paul Miner (on educational leave from NASA, ABD), Bob Burger (NSF Fellow, ABD), William Hunt (IU/CS member of technical staff) and Ingo Cyliax (IU/CS member of technical staff).

Graduate seminars on this research were conducted in 1992 and 1993. Many of the CAD tools first acquired for this work have migrated to instructional laboratories. The FM9001 microprocessor mentioned earlier is used in a continuing project to develop an instructional vehicle for a first undergraduate course in computer organization. Two undergraduate students, Derek Kern and Lisa Hatchett, have done independent-study projects developing this system.

1.2 Other related research

The work on decision tables by Hoover, Chen, and others [17, 16] inspired us to think more seriously about behavior tables as a natural representation for design. Their *Tablewise* specification tool was developed for avionics software development, but clearly applies to reactive systems in general. In addition to a graphical front end, there are functions for verifying exclusivity and completeness of decision tables and for performing structural analyses to aid in obtaining these properties. Planned enhancements are described for introducing behaviors and assertions.

All of this work seems to integrate with the proposed project quite well. Future topics mentioned in [17] include connections to state-machine and statechart based specification. This connection is the focus of the proposed work, and we are also interested in safety-critical applications. Even if we can only unify at the notational level, the combined functionality would enhance practical interest in this research area.

Li and Gupta introduce *timed decision tables* as an HDL [27, 26]. Their results on optimizations exploiting don't care entries are directly applicable to the proposed research. Their work is also evidence of the utility of a tabular specification language for CAD tool development. Behavior tables have also been proposed as an interchange format by Gajski, Dutt, et. al. [12, 8]. We find it very encouraging that research in high-level synthesis and formal methods finds common ground in these tabular representations; it represents a new opportunity for synergy.

Both Tablewise and TDTs have provisions for *assertions* that are not yet found in our behavior tables, but will be added. TDTs contain time measures used for automatic optimization. In Tablewise the primary intent seems to be the verification of invariant properties, but assertions could also be used to state constraints and measures.

Leveson's *Requirements State Machine Language* [25]. is based on Harel's state charts [14], but uses decision tables to specify hyper-edges. She echos Hoover's observation that decision tables are readily accepted and used by practicing engineers.

In formal methods research, interest in specialized notation is rising. The LAMBDA system uses a schematic editor to drive a proof assistant [9]. More recently, the *FORMAT* project employs a timing diagram tool for specification and verification tasks [38, 13]. Hunt's formalization of an HDL semantics in *Nqthm* [18], was related directly to the DDD algebra by Bose [3, 2]. The proposed project establishes rigorous mathematical semantics for behavior tables, laying the groundwork for a mechanized formalization. We do not plan to construct such a formalization during the proposed period, however.

1.3 Syntax of behavior tables

The technicalities presented in this section are not essential to understanding the more intuitive material in Section 1.4. The process semantics developed here is at odds with that of the languages mentioned in Section 1.2, but the differences are reconcilable. The variations are by no means peculiar to tabular notation.

We think of behavior tables as denoting persistent, communicating processes rather than procedures to be invoked. The substance of the difference is that behavior tables cannot themselves be entries in other behavior tables. Instead, they are composed by connecting their I/O ports.[1] The underlying communication model (globally synchronous, self-timed, etc.) creates still more semantic veriations.

Behavior tables are arrays of *terms* over an amalgamated abstract type, which gives ground syntax for constants and operations, and equational laws for reasoning about them. Our examples will involve commonly understood types, but a type system is intended to allow conceptual hierarchies, parameterization, and other structuring capabilities. A useful tool must have built-in reasoning for concrete types, but must also have facilities for reasoning about and between user specified type complexes.

The value of a term, t, is written $[\![t]\!]\sigma$, where σ is an *assignment* or association of variables to values. The notion of term evaluation is the standard one. A generic *don't-care* constant, written as '$\natural$', denotes an undetermined

[1] Converting invokations to interactions is central to our treatment to decomposition, as illustrated in Section 1.4 and discussed in Section 1.5

value.

A finite extension of propositional logic is assumed—Hoover calls it *finite logic* [16]. Arbitrary collections of enumerated values, or *tokens*, can be formed. These finite sets come with a polymorphic selection operation. A behavior table can be thought of as an iterated composition of selection expressions.

Behavior tables are closed expressions whose terms contain variables from three disjoint sets: I (inputs), S (data state), and C (combinational signals). Fix these sets for the remainder of this section. We will write ISC for $I \cup S \cup C$ and SC for $S \cup C$. We use the term 'register' for an element of S, but this is a euphemism that should be interpreted very abstractly. There is no assmption that these variables denote finite values. Although our example in Section 1.4 may not show it, behavior tables are intended for levels of description much higher than register transfer. The form of a behavior table is

Inputs → *Outputs*	
Conditions	*Registers and Signals*
⋮	⋮
Guard	*Computation Step*
⋮	⋮

Inputs is a list of input variables and *Outputs* is a set of terms over ISC; for simplicity, assume $O \subseteq ISC$. *Conditions* is a set P of predicates over ISC, that is, terms ranging over finite types, such as truth values, token sets, etc. A *guard* is a set of constants indexed by the condition set P: $g = \{c_p\}_{p \in P}$. We say g *holds* for an assignment σ to ISC when, for each $p \in P$, either $c_p = \natural$ or $[\![p]\!]\sigma = c_p$.

A *decision table* $\mathbf{D} = [P, G]$, consists of a condition set and a list of guards. Following [17], we say a decision table is *functional* when G describes a proper partitioning of the possible assignments to ISC. In other words, the guards are "exclusive and exhaustive." A *computation step* or *action* is a set of terms, one for each register and signal: $a = \{t_v\}_{v \in SC}$. An *action table* is an indexed set of actions.

A *behavior table for* $I \rightarrow O$ consists of a decision table, $\mathbf{D}$, with guards $G = \{g_1, \ldots g_n\}$, and an action table indexed by G $\mathbf{A} = \{t_{v,k} \mid v \in SC \text{ and } g_k \in G\}$.

Synchronous-process semantics. A behavior table $[\mathbf{D},\mathbf{A}]$ for $O \subseteq ISC$ denotes a relation between input and output *streams*, a stream being an infinite sequence of values. In our prior work we get a semantics by interpreting a table as a recursive system of stream-defining equations [21]. More directly, suppose we are given a set of initial values for the registers, $\{x_s\}_{s \in S}$ and a stream for each input variable in I. Construct a sequence of assignments, $\langle \sigma_0, \sigma_1 \ldots \rangle$ for ISC as follows:

(a) $\sigma_n(i)$ is given for all $i \in I$ and all n.

(b) For each $s \in S$, $\sigma_0(s) = x_s$.

(c) $\sigma_{n+1}(s) = [\![t_{s,k}]\!]\sigma_n$ if guard g_k holds for σ_n.

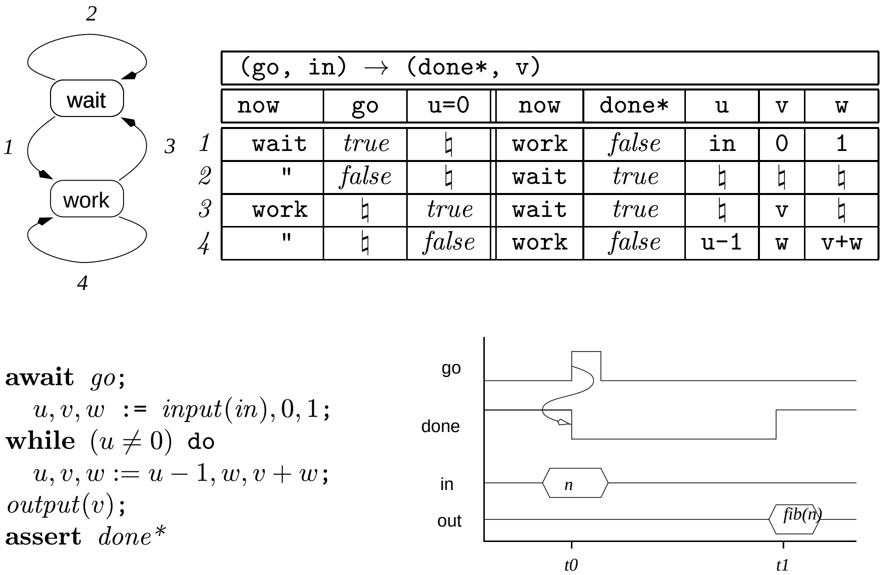
(d) For each $c \in C$, $\sigma_n(c) = [\![t_{c,j}]\!]\sigma_n$ if guard g_k holds for σ_n.

The stream associated with each $o \in O$ is $\langle \sigma_0(o), \sigma_1(o), \ldots \rangle$. This semantic relation is well defined if there are no circular dependencies among the combinational actions $\{t_{c,k} \mid c \in C,\ g_k \in G\}$. The relation is deterministic if decision table $\mathbf{D}$ is functional.

This semantics is at odds with both TDTs and Tablewise (Section 1.2), but the differences are reconcilable, and are by no means special to tabular notations. We think of behavior tables as denoting persistent, communicating processes rather than procedures to be invoked. The substance of the difference is that, under the semantics just defined, behavior tables cannot themselves be entries in other behavior tables. Instead, they are composed by connecting their I/O ports. However, coercing abstract operations to behavior tables is central to our approach to decomposition, as discussed in Section 1.5

Changing from a synchronous to an asynchronous communication model results in still another semantic variation. In fact, behavior tables will acquire a range of semantics, depending on their applications, just as HDLs and programming languages to. Their structure should help reduce the mathematical bookkeeping that often obscures semantic definitions.

Example of a behavior table. The behavior table shown in Figure 1 describes a process that computes the Fibonacci function. The inputs are control signal `go` and data input `in`; the outputs are control signal `done*` and the data signal `v`. The '`*`' is a notational convention for distinguishing signals from registers. Three other representations in Figure 1 depict various spects of the design. The labeled transition diagram is keyed to the table's rows; its labels consist of a condition under which the transition is taken, the



(go, in) → (done*, v)								
	now	go	u=0	now	done*	u	v	w
1	wait	*true*	♮	work	*false*	in	0	1
2	"	*false*	♮	wait	*true*	♮	♮	♮
3	work	♮	*true*	wait	*true*	♮	v	♮
4	"	♮	*false*	work	*false*	u-1	w	v+w

```
await $go$;
    $u, v, w$ := $input(in), 0, 1$;
while $(u \neq 0)$ do
    $u, v, w := u - 1, w, v + w$;
$output(v)$;
assert $done*$
```

Figure 1: A behavior table and related diagrams

outputs associated with the transition, and an update to the data state; the same information as a row of the table. A textual expression of the algorithm describes the computation of *fib*(n). A timing diagram shows the external handshake protocol.

The behavior table in Figure 2 (page 12) describes the garbage collector of a list processing computer [6]. It is more representative of the tables we work with in our case studies. Its level of specification is more abstract, with two of the registers of type *memory*. An implementation of this table was realized in about 5,000 ACTEL FPGA cells, of which 1,500 4-input MUX elements compute the behavior. A behavior table for the same computer's CPU is about twice as big, at a level of description where garbage collection is an abstract operation. A table closer the the register transfer level would be more than ten times larger, but even at that scale the tables are useful and would be much more so with better display automation.

1.4 Basic algebra of behavior tables

Let us explore some basic transformations, starting with the table in Figure 1. As in any derivation, the order of presentation is not necessarily the order in which the transformations were conceived. In practice, backtracking is involved as the architectural goals develop and concrete representations are introduced. The final derivation is just a proof of the design.

The `now` and `done*` columns suggest a assignment of concrete values `0` to `work` and `1` to `wait`. To reduce clutter, let us also assign `1` to `true` and `0` to `false`.

(go, in) → (done*, v)							
now	go	u=0	now	done*	u	v	w
1	1	♮	done*	¬go	in	0	1
"	0	♮	"	¬go	♮	♮	♮
0	♮	1	"	u=0	♮	v	♮
"	♮	0	"	u=0	u-1	w	v+w

With these changes, the first and second rows have become identical up to don't-cares, so we can merge them to obtain

(go, in) → (done*, v)							
now	go	u=0	now	done*	u	v	w
1	♮	♮	done*	¬go	in	0	1
0	♮	1	"	u=0	♮	v	♮
"	♮	0	"	u=0	u-1	w	v+w

The predicate `go` has become irrelevant will be removed. We note in passing that the last two rows could be merged by replacing the term for `v` with `select(u=0,v,w)`. Behavior tables seem especially useful for this kind of interplay between control and computation—all the more so with the provisions for *indirection* discussed in Section 1.5.

Next is a scheduling transformation on the third row that puts the arithmetic terms `u-1` and `v+w` into different computation steps. The goal is to assign these operations to a single arithmetic component.

go, in → done*, v						
now	u=0	now	done*	u	v	w
1	♮	done*	¬go	in	0	1
0	1	"	u=0	♮	v	♮
0	0	2	false	u-1	v	w
2	♮	done*	u=0	u	w	v+w

The newly created control token, '2', induces a type mismatch with boolean done* in the now column. This is a problem to be resolved by underlying type inference system. In addition to implicit coercions, this transformation's validity depends *both* on the fact that the sequence of two steps preserves the original computation *and* the fact that the surrounding synchronization protocol is preserved. Verifying the latter of these conditions is not automatic, in general, but is reducible to a transformation in cases where the synchronization context is known.

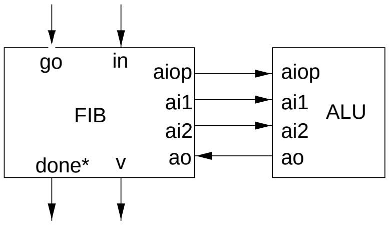
The next table is a simple example of *system factorization*, a decomposition technique that is central to the derivation formalism. As desired, terms u-1 and v+w are allocated to a single combinational arithmetic component, called ALU.

FIB: (go, in, ao) → (done*, v, ai1*, ai2*, aop*)

now	u=0	now	done*	u	v	w	aop*	ai1*	ai2*
1	♮	done*	¬go	in	0	1	♮	♮	♮
0	1	"	u=0	♮	v	♮	♮	♮	♮
0	0	2	false	ao	v	w	sub	u	1
2	♮	done*	u=0	u	w	ao	add	v	w

ALU:(aop,ai1,ai2) → ao*

aop	ao*
add	ai1+ai2
sub	ai1-ai2



A system factorization encapsulates a set of subject terms in a new table and generates residual interface signals [19]. Here, the interface signal aop generates instruction tokens, sub and add, telling ALU which operation to perform. The table manipulation tool keeps track of the connectivity.

To finish the example, we make some assignments to the don't-care entries whose ultimate effect is to isolate control. As a second example of system factorization, we decompose into a control process generating an encoded command signal, cmd, to the data path DP:

CTL: (go, u) → (done*, cmd*, aop*)

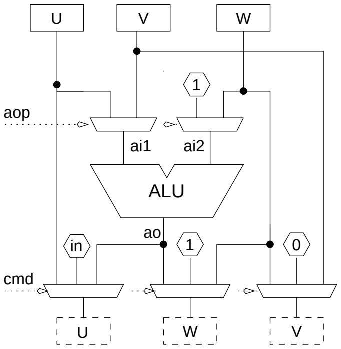
now	u=0	now	done*	cmd*	aop*
1	♮	done*	¬go	0	♮
0	1	"	u=0	1	sub
0	0	2	false	1	sub
2	♮	done*	"	2	add

DP: (cmd, in, ao)
→ (u, v, ai1*, ai2*)

cmd	u	v	w	ai1*	ai2*
0	in	0	1	♮	♮
1	ao	v	w	v	w
2	u	w	ao	u	1

ALU:(aop,ai1,ai2) → ao*

aop	ao*
add	ai1+ai2
sub	ai1-ai2



Discussion of the example. The preceeding example was carried out by hand, but all the transformations involved have already been mechanized in the DDD transformation system. Currently, DDD manipulates recursive systems of stream equations. The advantage of this approach has been that the DDD forms are directly executable expressions in a general purpose modeling language (Scheme, with extensions for streams). Thus, DDD system is using a "shallow" semantic embedding, for which the tables are simply a visual representation [22, 2, 37].

One problem with this approach is that as the visual representation becomes sophisticated, its relation to the underlying semantic expressions becomes correspondingly more difficult to sustain. It is a problem for the user, who is interacting with the syntax, as well as for the tool implementor, who is striving to support the user's interpretation. A more direct internal representation of behavior tables within DDD would be simpler and better.

In practice, system factorization is a goal-directed activity. Consider a pipelined ALU that takes operands sequentially.

	NOW	RQ	(= U A)	(pointer? H)	(bvec-h? H)	(eq? 'forward (tag d))	(tag H)	(= C 0)	(= C 1)	NOW	OLD:*mem*	NEW:*mem*	H:*cont*	D:*cont*	C:*addr*	U:*cont*	A:*addr*	AK
1	idle	1	♮	♮	♮	♮	♮	♮	♮	driver	OLD	NEW	H	D	C	(rd OLD H)	0	0
2	"	0	♮	♮	♮	♮	♮	♮	♮	idle	OLD	NEW	H	♮	♮	♮	♮	1
3	driver	♮	1	♮	♮	♮	♮	♮	♮	idle	NEW	OLD	0	D	C	U	A	1
4	"	♮	0	♮	♮	♮	♮	♮	♮	nextobj	OLD	NEW	(rd NEW U)	D	C	U	A	0
5	nextobj	♮	♮	1	♮	♮	♮	♮	♮	objtype	OLD	(wt NEW U (cell H A))	H	(rd OLD H)	C	U	A	0
6	"	♮	♮	0	1	♮	♮	♮	♮	driver	OLD	NEW	H	(rd OLD H)	C	(+ U (btow-u (pr-pt H)) (cin 1))	A	0
7	"	♮	♮	0	0	♮	♮	♮	♮	driver	OLD	NEW	H	(rd OLD H)	C	(+ U (const 0) (cin 1))	A	0
8	objtype	♮	♮	♮	♮	1	♮	♮	♮	driver	OLD	(wt NEW U (cell H d))	H	D	C	(+ U (const 0) (cin 1))	A	0
9	"	♮	♮	♮	♮	0	fixed	♮	♮	copy	(wt OLD H (cell forward A))	NEW	(cell H (add1-ptr (pr-pt H)))	D	(add1-2 (fixed-size H))	U	A	0
10	"	♮	♮	♮	♮	0	vec	♮	♮	vec	(wt OLD H (cell forward A))	NEW	(cell H (add1-ptr (pr-pt H)))	D	(btow-c (pr-pt d))	U	A	0
11	"	♮	♮	♮	♮	0	bvec	♮	♮	vec	(wt OLD H (cell forward A))	NEW	(cell H (add1-ptr (pr-pt H)))	D	(btow-c (pr-pt d))	U	A	0
12	vec	♮	♮	♮	♮	♮	♮	1	♮	driver	OLD	(wt NEW A d)	(cell H (add1-ptr (pr-pt H)))	D	C	(+ U (const 0) (cin 1))	(add1-a A)	0
13	"	♮	♮	♮	♮	♮	♮	1	♮	copy	OLD	(wt NEW A d)	(cell H (add1-ptr (pr-pt H)))	(rd OLD H)	C	U	(add1-a A)	0
14	copy	♮	♮	♮	♮	♮	♮	♮	1	driver	OLD	(wt NEW A d)	H	D	C	(+ U (const 0) (cin 1))	(add1-a A)	0
15	"	♮	♮	♮	♮	♮	♮	♮	0	copy	OLD	(wt NEW A d)	(cell H (add1-ptr (pr-pt H)))	(rd OLD H)	(sub1 C)	U	(add1-a A)	0

Figure 2. Behavior Table for a Garbage Collector

`ALU:(op,in) → (phase, out*)`

phase	op	phase	hold	out*
1	♮	2	in	♮
2	add	1	in	hold + in
2	sub	1	in	hold - in

A still more general notion of factorization is needed to instantiate this sequential behavior in `FIB`. This topic is discussed further in the next section.

Generally, we are interested in levels of design at which human interaction is used. In numerous case studies, we have found behavior tables to be a valuable aid in forming tactical design goals and managing the abstractions of higher levels of specification. However, we *do not* promote them as the sole visual representation needed in interactive design. We used several graphic representations of control, architecture, and interface in presenting of our example. Behavior tables are a fusion of control and structural aspects, not ideal for describing either, but highly useful in when the designer must address both.

1.5 Research topics and project goals

The design derivation tools we have already developed are cost-effective in cases requiring a high assurance of correctness. Our goal is to expand the range of applicability by creating better reasoning tools, and in particular, a better user interface . This project achieves that goal by supporting a more concise notation and developing more robust transformations.

Even if formal reasoning is not involved, behavior tables are highly effective for describing reactive processes. While we do have specific research objectives in formal methods, this project is designed to contribute to more widespread adoption of tabular notation in hardware design, embedded design, codesign, and even software.

A behavior-table widget. We will implement a prototype graphical object for manipulating behavior tables. Display software is needed to organize the text arrays; perform windowing, abbreviation, and navigation; animate analysis; and generate diagrams. We hope to conduct this work cooperatively, incorporating ideas and software from existing systems like Tablewise, DTDs, and RSML [Section 1.2]. In return, we will develop display objects with more general capabilities than our local needs require.

The first year of the proposed period will be exploratory. In preliminary investigations we have looked at *Motif*, *Tk/Tcl*, *Emacs*, *Latex* and various Scheme based libraries (UT's *MrEd*, *Stk*, and, IU's *SDL*). We have found neither a consensus on graphics infrastructure, nor much satisfaction with what is presently available. Many people are pinning their hopes on Java as a possible common ground, and we intend to look at this prospect very seriously.

By the end of the first year we will have established core requirements and chosen a software infrastructure for implementing a behavior table display tool. In the second year we will finish a prototype suitable for published demonstrations of the ideas and ready for integration with other reasoning and design tools.

Design derivation with behavior tables. DDD will be re-implemented to operate directly on symbolic tabular representations. We will be able to display formal derivations with behavior tables by the half-way point in the project, and will go on to develop facilities to interact with DDD through the graphical front end.

System design decomposition. System factorization in conjunction with data abstraction is the basis of decomposition in our formalization. The notion is more general than the examples show, having evolved over several years of research [31, 33, 39, 45, 32, 44, 34, 19].

The `ALU` factorization in Section 1.4 is trivial factorization because it doesn't involve state. The `CTL` factorization did involve state but was still trivial because the time granularity was fixed. Another step in complexity was suggested by the pipelined ALU, shown in the discussion.

The behavior table on page 12 has registers `OLD` and `NEW` of type *memory*. A simple factorization would decompose it into tables representing the data path and two synchronous memory components [19].

However, suppose we intend to implement the memory abstraction with a conventional DRAM, and so introduce sequential *read*, *write*, and *refresh* protocols. The coordination may be subsumed in the timing discipline (e.g. mutually interacting clocks, or self timed communication), involve a physical interface (e.g. multiplexing the address and data signals), or be explicitly instantiated in the collector's behavior. A general notion of factorization accommodating these choices, developed [33, 32], was done with behavior tables in mind, and will be implemented during this project.

Indirection We have added syntax for *bounded indirection* which often significantly reduces the size of behavior tables [35]. We believe this notation to be novel for hardware description languages. If r is a signal or register, then #r denotes a token referring to r. If register s contains such a token, then @s denotes the entity to which s refers; that is,

$$\texttt{@}s \;\equiv\; \mathsf{case}\ s\ \mathsf{of}\ \ldots \#r\!: r \ldots$$

In [40] we show how a behavior table describing a bus reduces to one line when indirection is used to specify sources and destinations. In [39] we explore control indirection. These features will be incorporated in both the graphic table object and the underlying transformation system.

Abstraction. Providing a type system to support behavior table manipulation is an important subproject. Unfortunately, we have yet to find everything we need in a single existing type system. In addition to finite logic, we want provisions for first-order inference with transparent coercions, a logical notion of implementation support factorization, and an embedded theory of streams. We have investigated some of these issues using PVS [28], but integrating the graphics with this theorem prover would be too difficult. A more likely path is to refine a locally developed type inference system for Scheme [15].

In summary, this project will result in two disseminable software artifacts. The first is an autonomous graphic front end for manipulating behavior tables, with provisions for animating analysis and synthesis algorithms obtained from contemporary work.

Second, a descendent of the DDD transformation system will be integrated with the GUI, and its type system enhanced to better support system factorization.

As we have always done in the past, we will integrate these products with CAD tools so that the realistic examples of the approch can be demonstrated and meaningful case studies can be conducted.

D References cited

[1] Bhaskar Bose. DDD - a transformation system for digital design derivation. Technical Report 331, Indiana University, Computer Science Department, May 1991.

[2] Bhaskar Bose. *DDD-FM9001: Derivation of a Verified Microprocessor.* PhD thesis, Computer Science Department, Indiana University, USA, 1994. Technical Report No. 456, 155 pages.

[3] Bhaskar Bose and Steven D. Johnson. DDD-FM9001: Derivation of a verified microprocessor. In G. Milne and L. Pierre, editors, *Proceedings of IFIP Conference on Correct Hardware Design and Verification Methods*, volume 683 of *LNCS*, pages 191–202. Springer, 1993.

[4] Bhaskar Bose, Steven D. Johnson, and Shyam Pullela. Integrating boolean verification with formal derivation. In D. Agnew, L. Claesen, and R. Camposano, editors, *Proceedings of IFIP Conference on Hardware Description Languages and their Applications*, pages 127–134. Elsevier, April 1993.

[5] Bhaskar Bose, M. Esen Tuna, and Steven D. Johnson. System factorization in codesign: A case study of the use of formal gechniques to achieve hardware-software decomposition. In *Proceedings of the International Conference on Computer Design*, pages 458–461. IEEE, October 1993.

[6] Robert G. Burger. The scheme machine. Technical Report 413, Indiana University, Computer Science Department, August 1994. 59 pages.

[7] Derivation Systems, Inc., Carlsbad, California. *DRS: Derivational Reasoning System*, 1.2.1 edition, December 1995. Contact `drs@derivation.com`.

[8] Nikil D. Dutt and Daniel D. Gajski. Exel: A language for interactive behavioral synthesis. In John A. Darringer and Franz J. Rammig, editors, *Computer Hardware Description Languages and their Applications*, pages 3–18, 1989.

[9] Simon Finn, Michael P. Fourman, Micheal Francis, and Robert Harris. Formal system design: interactive synthesis based on computer-assisted formal reasoning. In Claesen, editor, *Formal VLSI Specification and Synthesis: VLSI Design Methodds–I.* North-Holland, 1989.

[10] Kathryn Fisler. *A Unified Approach to Hardware Verification Through a Heterogenious Logic of Design Diagrams.* PhD thesis, Computer Science Department, Indiana University, USA, 1996.

[11] Kathryn Fisler and Steven D. Johnson. Integrating design and verification envrionments through a logic supprting hardware diagrams. In *ACV'95*, pages 669–674, 1995. Proceedings of the ASP-DAC'95, CHDL'95, VLSI'95 International Conference, Chiba, Japan; IEEE Cat. No. 95TH8102.

[12] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[13] Werner Grass, Cristian Grobe, Sefan Lenk, Wolf-Dieter Tiedemann, Carlos Delgado Kloos, Andrés Marín, and Tomaás Robles. Transformation of timing diagram specifications into VHDL code. In *ACV'95*, pages 643–650, 1995. Proceedings of the ASP-DAC'95, CHDL'95, VLSI'95 International Conference, Chiba, Japan; IEEE Cat. No. 95TH8102.

[14] D. Harel. Statecharts: a visual formalism for complex systems. *The Science of Computer Programming*, 8:231–274, 1987.

[15] Christopher T. Haynes. Infer: A statically-typed dialect of Scheme. Technical Report 367, Indiana University Computer Science Department, March 1993.

[16] D. N. Hoover and Zewei Chen. Tbell: A mathematical tool for analyzing decision tables. Contractor Report 195027, National Aeronautics and Space Administration, Hampton VA 23681-0001, November 1994. Authors' affiliation: Odyssey Research Associates, Inc., Ithaca NY.

[17] D. N. Hoover, David Guaspari, and Polar Humenn. Applications of formal methods to specification and safety of avionics software. Contractor Report 4723, National Aeronautics and Space Administration Langley Research Center (NASA/LRC), Hampton VA 23681-0001, November 1994. Authors affiliation: Odyssey Research Associates, Inc., Ithaca NY. Printed copies available from NASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights MD 21090-2934.

[18] Warren A Hunt Jr. and Bishop C. Brock. The DUAL-EVAL hardware description language and its use in the formal specification and verification of the FM9001 microprocessor. *Formal Methods in System Design*, 1997. To appear. Invited presentation at *ACV'95*, Chiba, Japan. Contact `hunt@cli.com` for a draft copy.

[19] Steven D. Johnson. Manipulating logical organization with system factorizations. In Leeser and Brown, editors, *Hardware Specification, Verification and Synthesis: Mathematical Aspects*, volume 408 of *LNCS*, pages 260–281. Springer, 1989. Proceedings of Mathematical Sciences Institute Workshop, Cornell University, 1989.

[20] Steven D. Johnson, Gerard Allwein, and K. Jon Barwise. Toward the rigorous use of diagrams in reasoning about hardware. In Gerard Allwein and Jon Barwise, editors, *Logical Reasoning with Diagrams.* Oxford University Press, 1996.

[21] Steven D. Johnson and B. Bose. A system for digital design derivation. Technical Report 289, Indiana University, Computer Science Department, Indiana University, August 1989. Summary presented at the 1989 IEEE High Level Synthesis Workshop, Kennebunkport, Maine.

[22] Steven D. Johnson, B. Bose, and C.D. Boyer. A tactical framework for digital design. In Birtwistle and Subramanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 349–383. Kluwer, Boston, 1988.

[23] Steven D. Johnson and Bhaskar Bose. A system for mechanized digital design derivation. In Subramanyam, editor, *ACM/SIGDA International Workshop on Formal Methods in VLSI Design*, 1991. Meeting held in Miama, Florida in 1990; proceedings remain unpublished. A copy can be obtained through URL `http://www.cs.indiana.edu/hmg/hmg.html`.

[24] Steven D. Johnson, Paul S. Miner, and Albert Camilleri. Studies of the single-pulser in various reasoning systems. In Ramayya Kumar and Thomas Kropf, editors, *Theorem Provers in Circuit Design*, volume 901 of *LNCS*, pages 209–227. Springer, 1995. Proceedings of the Second International Conference, TPCD'94.

[25] Nancy G. Leveson, Mats Per Erik Heimdahl, Holly Hildreth, and Jon Damon Reese. Requirements specifiation for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684–707, September 1994.

[26] Jian Li. Timed decision tables: A behavioral model for embedded system specification and optimization. Technical Report UIUCDCS-R-96-1971, Univeristy of Illinois Department of Computer Science, 1304 West Springfield Ave, Urbana IL 61801, 1996. Ph. D. dissertation.

[27] Jian Li and Rejash K. Gupta. HDL optimization using timed decision tables. In *33rd ACM/IEEE Design Automation Conference*, 1996.

[28] Paul S. Miner and Steven D. Johnson. Verification of an optimized fault-tolerant clock synchronization circuit: A case study exploring the boundary between formal reasoning systems. In Satnam Singh, Mary Sheeran, and Geraint Jones, editors, *Third Workshop on Designing Correct Circuits.* Springer, 1996.

[29] Paul S. Miner, Shyamsundar Pullela, and Steven D. Johnson. Interaction of formal design systems in the development of a fault-tolerant clock synchronization circuit. In *13th Symposium on Reliable Distributed Systems*, pages 128–137, 1994. Proceedings of SRDS 94 held at Dana Point, California, October 1994.

[30] K. Rath, I. Celis, and R. M. Wehrmeister. RTBA: A generic bit-sliced bus architecture for datapath synthesis. Technical Report 321, Department of Computer Science, Indiana University, December 1990.

[31] Kamlesh Rath. *Sequential System Decomposition*. PhD thesis, Computer Science Department, Indiana University, USA, 1995. Technical Report No. 457, 90 pages.

[32] Kamlesh Rath, Bhaskar Bose, and Steven D. Johnson. Derivation of a DRAM memory interface by sequential decomposition. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 438–441. IEEE, October 1993.

[33] Kamlesh Rath, Venkatesh Choppella, and Steven D. Johnson. Decomposition of sequential behavior using interface specification and complementation. *VLSI Design Journal*, 3(3-4):347–358, 1995.

[34] Kamlesh Rath and Steven D. Johnson. Toward a basis for protocol specification and process decomposition. In D. Agnew, L. Claesen, and R. Camposano, editors, *Proceedings of IFIP Conference on Hardware Description Languages and their Applications*, pages 157–174. Elsevier, April 1993.

[35] Kamlesh Rath, M. Esen Tuna, and Steven D. Johnson. Behavior tables: A basis for system representation and transformational system synthesis. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 736–740. IEEE, November 1993.

[36] Kamlesh Rath, M. Esen Tuna, and Steven D. Johnson. An introduction to behavior tables. Technical Report 392, Indiana University Computer Science Department, December 1993.

[37] Kamlesh Rath, M. Esen Tuna, and Steven D. Johnson. Specification and synthesis of bounded indirection. Technical Report 398, Indiana University, Computer Science Department, February 1994.

[38] Rainer Schlör. A prover for VHDL-based hardware design. In *ACV'95*, pages 643–650, 1995. Proceedings of the ASP-DAC'95, CHDL'95, VLSI'95 International Conference, Chiba, Japan; IEEE Cat. No. 95TH8102.

[39] M. Esen Tuna, Steven D. Johnson, and Bob Burger. Continuations in hardware-software codesign. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 264–269. IEEE, October 1994.

[40] M. Esen Tuna, Kamlesh Rath, and Steven D. Johnson. Specification and synthesis of bounded indirection. In *Proceedings of the Fifth Great Lakes Symposium on VLSI*, pages 86–89. IEEE, March 1995.

[41] Zheng Zhu. *Structured Hardware Design Transformations*. PhD thesis, Computer Science Department, Indiana University, USA, 1992.

[42] Zheng Zhu and Steven D. Johnson. An algebraic framework for data abstraction in hardware description. In Jones and Sheeran, editors, *Proceedings of The Oxford Workshop on Designing Correct Circuits*. Springer, 1990.

[43] Zheng Zhu and Steven D. Johnson. An example of interactive hardware transformation. In Subramanyam, editor, *Proceedings of ACM International Workshop on Formal Methods in VLSI Design*, January 1991. available as Techical Report 383, Computer Science Department, Indiana University.

[44] Zheng Zhu and Steven D. Johnson. Automatic synthesis of sequential synchronizations. In D. Agnew, L. Claesen, and R. Camposano, editors, *Proceedings of IFIP Conference on Hardware Description Languages and their Applications*, pages 285–301. Elsevier, April 1993.

[45] Zheng Zhu and Steven D. Johnson. Capturing synchronization specifications for sequential compositions. In *Proceedings of the 1994 IEEE International Conference on Computer Design (ICCD 94)*, pages 117–121. IEEE, October 1994.