Alex Tsow: An Overview of Starfish: A Table-Centric Tool for Interactive Synthesis, Proceedings of The Sixth NASA Langley Formal Methods Workshop, p.76–78

An Overview of *Starfish*: A Table-Centric Tool for Interactive Synthesis[⋆]

Alex Tsow

The MITRE Corporation[⋆⋆]*, Mclean, VA 22102, USA*
atsow@mitre.org

Extended Abstract

Engineering is an interactive process that requires intelligent interaction at many levels. My thesis [1] advances an engineering discipline for high-level synthesis and architectural decomposition that integrates perspicuous representation, designer interaction, and mathematical rigor. *Starfish*, the software prototype for the design method, implements a table-centric transformation system for reorganizing control-dominated system expressions into high-level architectures. Based on the *digital design derivation (DDD)* system—a designer-guided synthesis technique that applies correctness preserving transformations to synchronous data flow specifications expressed as co-recursive stream equations—Starfish enhances user interaction and extends the reachable design space by incorporating four innovations: *behavior tables*, *serialization tables*, *data refinement*, and *operator retiming.*

Behavior tables express systems of co-recursive stream equations as a table of guarded signal updates. Developers and users of the DDD system used manually constructed behavior tables to help them decide which transformations to apply and how to specify them. These design exercises produced several formally constructed hardware implementations: the FM9001 microprocessor, an SECD machine for evaluating LISP, and the *SchemEngine*, garbage collected machine for interpreting a byte-code representation of compiled Scheme programs. Bose and Tuna, two of DDD's developers, have subsequently commercialized the design derivation methodology at *Derivation Systems, Inc. (DSI)*. DSI has formally derived and validated PCI bus interfaces and a Java byte-code processor; they further executed a contract to prototype SPIDER—NASA's ultra-reliable communications bus.

To date, most derivations from DDD and DRS have targeted hardware due to its synchronous design paradigm. However, Starfish expressions are independent of the synchronization mechanism; there is no commitment to hardware or globally broadcast clocks. Though software back-ends for design derivation are limited to the DDD stream-interpreter, targeting synchronous or real-time software is not substantively different from targeting hardware.

The separation of concerns—e.g., architecture, behavior, data representation, and interface coordination—is standard engineering doctrine. In particular, it is futile to expose all aspects

⋆ Many thanks to the NASA Langley Research Center's generous sponsorship of this work through their Graduate Student Researcher's Program, NGT-1-010009. This extended abstract is a revised excerpt from the author's doctoral dissertation accepted by the Indiana University Computer Science Department.[1]

⋆⋆ The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.

equally well with a single language. Behavior tables represent a compromise between behavior and architecture: its rows roughly characterize a specification's control oriented aspects, while the columns represent its architectural, or structural, aspects.

The behavior table transformations—among other things—allow designers to trade between these two axes, thereby balancing between the two aspects. It is no surprise, then, that behavior tables are well suited for deriving architectural components from behaviorally oriented expressions.

Type Management and Data Refinement: Behavior tables operate on arbitrarily abstract data-types, not just bit-vectors and bounded integers. In this respect, they are far more expressive than standard hardware description languages. Starfish implements an explicit type system and a framework for data refinement to support high-level specification with abstract data types.

Demand for explicit typing arose from several areas: the need to limit decision expressions to finitely branching guards, the need to prevent incompatible signal merging opportunities among unused slots in table columns, and the desire to increase feedback by disallowing unsound transformations at earlier stages. The type system, which is based on multisorted structures, takes on a second responsibility: it forms a database of term-level identities. One of the core transformations applies algebraic identities (e.g., operator commutativity) to terms. While many term rewrites in DDD are combinator expansions, each algebraic term rewrite requires external validation. Starfish leverages the type system's identity database to confirm algebraic rewrites—only the identity pattern needs external verification.

Since the type system declares function symbols, signatures and identities, it provides a foundation for *data refinement*. At the simplest level, a system of identities can express one-to-one homomorphisms between types. While such an identity system transforms abstract *terms* into representation terms, the architectural algebra preceding Starfish could not transform abstract *signals* into representation signals in a general way. The first attempts to impose signal-level refinement were *ad hoc*, but the current refinement process follows from retiming and recursive identity expansion. In addition to refinement by one-to-one homomorphism, Starfish supports *one-to-many refinements*, where there are multiple representations for each abstract type, and *stateful refinements*, which represent multiple signals with references to a shared store.

While behavior tables are not useful for defining data refinements, they are useful for exploiting and managing their consequences. Data refinements lead to more detailed specifications and consequently a wider transformation space. System decompositions, the problem for which behavior tables were developed, may "cut across" a representation that implements an abstract type with a collection of signals. For instance, suppose a refinement simulates abstract stacks with a pointer and array; subsequent architectural organization may separate the array from the pointer. In another case, a stateful refinement may impose serial access on the previously unconstrained concurrency of abstract operations. Behavior tables and their scheduling aid, serialization tables, provide an interactive method for integrating the serial requirements into a system's control and architecture by scheduling access before and after stateful refinements.

Scheduling and serialization: Starfish introduces *serialization tables* for scheduling the evaluation of complex action terms over several steps. Like behavior tables, columns represent signals and rows represent simultaneous actions which update the signals. Serialization tables are an organizational aid that helps designers solve the NP-hard problems involved in high-level synthesis; e.g., how to fit an evaluation sequence within a specified number of registers and execution units. Serialization tables help specify evaluation order and intermediate resource usage for compound actions in a behavior table. As the schedule develops, the serialization tables display partial symbolic-evaluations of the intermediate actions. This feedback mechanism helps designers specify actions in the subsequent steps. Starfish validates correctness before integrating the actions into behavior table expressions.

The DDD algebra views serialization as primarily a behavioral problem. Yet, the goal of scheduling is often architecturally dictated. One must use a limited set of resources. Register allocation, functional allocation, and timing are not fully exposed in DDD's behavioral representations. Serialization tables display these aspects more clearly than DDD's co-recursive stream equations, making them a better suited medium for the schedule specification process.

Retiming: Starfish supports retiming in two ways. One is with serialization tables and local re-serialization, or adjustment of schedules. The other is with a transformation that converts combinational signals to sequential signals and vice versa. In schematic terms, the transformation pushes a functional unit from one side of a register to the other. Although retiming is the critical step in transforming abstract signals to representation signals, the motivating example in Starfish was a stack-calculator derivation. The original specification used a combinational `top` accessor for the output signal. Any plausible implementation would store the top value in a register. The exercise of hand-specifying a stack-calculator with a registered `top` signal was enough to see a generalizable pattern. Indeed, equivalent transformations have been used in formal synthesis and microarchitecture algebra.

This talk surveys Starfish's incorporation of behavior tables, data refinement, serialization, and retiming into design derivation. Please see my thesis [1] for an in-depth presentation of these techniques; full text is freely available on the Web.

References

[1] Alex Tsow. *Starfish: A Table Centric Tool for Design Derivation.* PhD thesis, Indiana University Computer Science Department, Bloomington, IN, July 2007. Technical Report 650, 272 pages, `http://www.cs.indiana.edu/pub/techreports/TR650.pdf`.