Thesis/Dissertation Sheet

Surname/Family Name	:	**Robinson**
Given Name/s	:	**Amos Stephen**
Abbreviation for degree as in the University calendar	:	**PhD**
Faculty	:	**Faculty of Engineering**
School	:	**School of Computer Science and Engineering**
Thesis Title	:	**The stuff that streams are made of**

Abstract 350 words maximum: (PLEASE TYPE)

To learn from a large dataset, we generally want to perform lots of queries. If we perform each query separately, we may spend more time reading and re-reading the same dataset than we spend computing the answer. Instead of performing each query separately, we would like to amortise the cost of reading the data by performing multiple queries at the same time.

Two streaming models for executing multiple queries concurrently are push streams and Kahn process networks.
Push streams can be used to execute multiple queries concurrently, but push streams can be unwieldy to use as queries must be constructed ``back-to-front''. We introduce a query language called Icicle, which allows programmers to write and reason about queries using a more familiar array-based semantics, while retaining the execution strategy of push streams. The type system of Icicle guarantees that well-typed query programs have the same semantics whether they are executed as array programs or as stream programs, and that all queries over the same input data can be executed together.

However, push streams do not support computations with multiple inputs except for non-deterministically merging two streams. Kahn process networks support multiple inputs and multiple queries, but require dynamic scheduling and inter-process communication, which can introduce significant overhead. We introduce a method for taking multiple processes in a Kahn process network and fusing them together into a single process. The fused process communicates through local variables rather than costly communication channels. This fusion method generalises previous work on stream fusion and demonstrates the connection between fusion and the synchronised product operator, which is generally used in the context of verification and model checking, rather than as an optimisation.

Some queries require multiple passes, as they need to read the input data multiple times, or may produce intermediate outputs which are then read back in. There are usually many different ways to schedule the work among the separate passes. Prior work demonstrated how integer linear programming (ILP) can be used to find optimal schedules for imperative array programs. However, these approaches only handle operations that preserve the size of the array, missing some optimisation opportunities. We introduce a clustering algorithm for scheduling queries written using array combinators, and extend prior work to support size-changing operations.

FOR OFFICE USE ONLY Date of completion of requirements for Award: