Undecidability

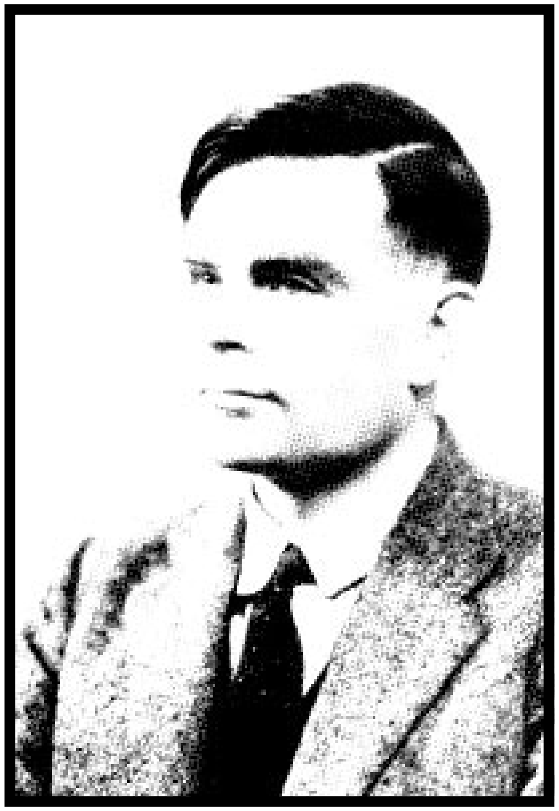
The Halting Problem

It is undecidable whether a Turing machine terminates.

The Finite Tape Problem

It is undecidable whether a Turing machine can execute with finite memory.

"Undecidable" means that no algorithm can decide *in finite time* for all instances.



One Solution — Limit the Model of Computation

- **Static dataflow [Dennis]**
- **Well-behaved dataflow [Gao]**
- **K-bounded loops [Culler]**
- **Computation graphs [Karp & Miller]**
- **Synchronous dataflow [Lee & Messerschmitt]**
- **Synchronous languages [Lustre]**

Turing Complete

- **Boolean dataflow [Buck]**
- **Dynamic dataflow [Dennis, Arvind, ...]**
- **Kahn process networks [Kahn]**

Another Solution — Solve the Undecidable Problem

The Boolean Dataflow Solution

Quasi-Static scheduler: A finite list of *annotated* firings is executed forever. The annotations are Boolean conditions under which the firing should occur.

The finite schedule must return the graph to its original state for all outcomes of the Booleans.

- **It is undecidable whether such a schedule exists.**
- **Heuristics search for such a schedule, and fall back on dynamic scheduling if they fail to find one.**

Dynamic scheduling

A run-time algorithm determines which actor fires next.

The problem:

Does there exist an algorithm that satisfies the bounded memory criterion and deadlock constraint that can make *n* firing decisions in O(*n*) time for any *n*?

Notes:

- **An infinite execution takes infinite time.**
- **An infinite execution must make steady progress.**
- **If an execution stops, there must be no enabled actors.**
- **If an execution consumes unbounded memory, there must be no execution with bounded memory.**

Policies that Fail

1. **Fairness**

   - **All actors should execute infinitely often, if possible.**
   - **All data produced on an arc should be consumed.**

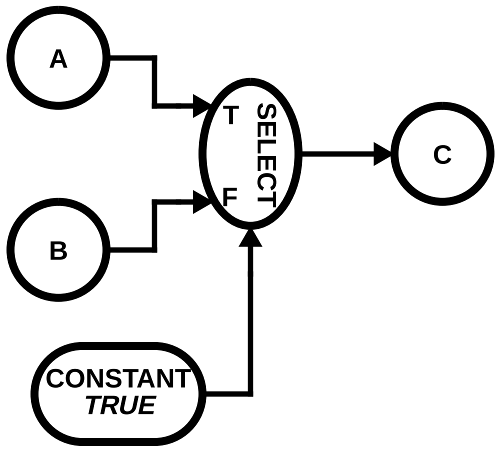
2. **Data-driven**

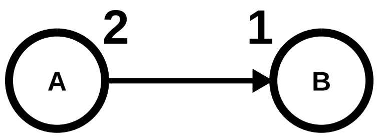
   - **Fire actors enabled by data at their inputs.**

3. **Demand-driven**

   - **Fire actors whose outputs are needed.**

Fairness

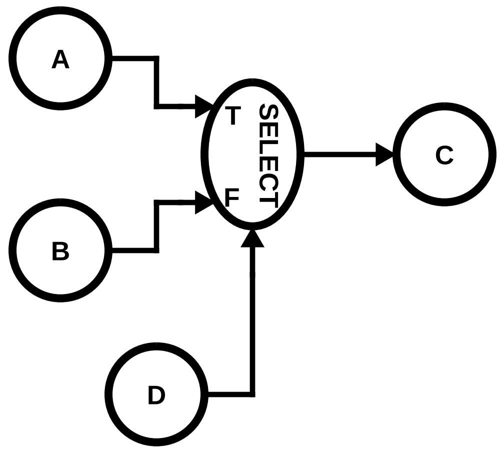




This fails under the policy that all actors should execute infinitely often, if possible, because A and B are always enabled.

This could fail under the policy that all data produced on an arc should be consumed.

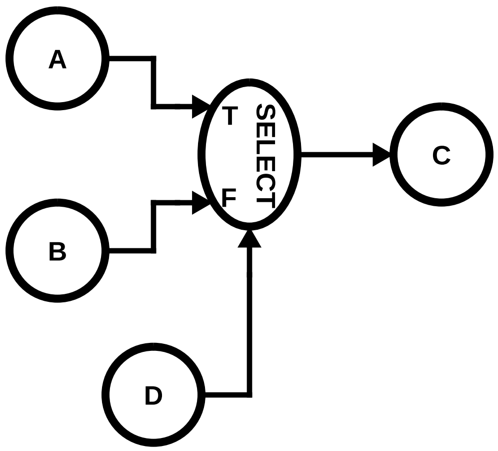
Data-Driven



A data-driven policy gives no clue about how often to fire A and B.

Note that a common solution to this is to say that A and B represent inputs from the outside, and the outside should determine how often they fire. ***This is not satisfactory.***

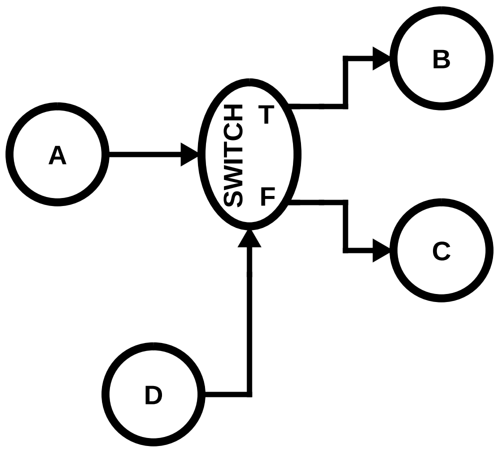
The least fixed point may not be the desired solution



The least fixed point solution will (probably) have infinite sequences on all three source outputs.

This suggests a demand driven approach (which also does not necesarily find the least fixed point solution), but ...

Demand-Driven



A demand-driven policy gives no clue about how often to issue demands from B and C.

Note that a common solution to this is to say that B and C represent outputs to the outside, and the outside should determine how often they issue they demands. *This is not satisfactory.*

Balanced Data/Demand Driven Execution

- Eazyflow [Jagannathan and Ashcroft]
- Anticipation coefficients [Kahn and McQueen]
- Ptolemy DDF scheduler (before version 0.6) [Ha]

All of these fail because they require fixed thresholds to determine whether streams are eager or lazy. The thresholds cannot be fixed [Parks].

We solve the problem first for Kahn process networks (with blocking reads), then specialize to dataflow graphs.

Bounded Memory Lemma

Bounded memory means either:

(a) the maximum number of tokens in existence at any time is bounded, or

(b) the maximum capacity of each arc is bounded.

Lemma

> **A fixed, finite-size Kahn process network can be executed in bounded memory under (a) if and only if it can be executed in bounded memory under (b).**

The Growing Bounded Buffer Policy

Definition: a process in *input-blocked* if it is blocked trying to read an input.

> **Policy: Start with an arbitrary bound *K*. Execute non-input-blocked processes as long as they do not cause the number of tokens on their output arcs to exceed *K* (in which case they become *output-blocked*). If this never terminates, we have found a live, bounded-memory execution. If it terminates, there are two possibilities. First, all processes are input-blocked, so the graph is not live, and we can stop. If some process is output-blocked, then we should increase *K* and continue.**

Theorem [Parks ‘95]:

> **This policy satisfies the bounded memory criterion and deadlock requirement.**

Adapting this to Dataflow

Policy:

> **Start with an arbitrary bound *K*. Execute enabled actors that will not cause the number of tokens on their output arcs to exceed *K*. If this never terminates, we have found a live, bounded-memory execution. If it terminates, there are two possibilities. First, if there are no enabled actors, the graph is not live, and we can stop. If there are enabled actors, then we should increase *K* and continue.**