Chapter 1-3: Embedded Computing

Soo-Ik Chae

Topics

- Why models of computation?
- Structural models.
- Finite-state machines.
- Turing machines.
- Petri nets.
- Control flow graphs.
- Data flow models.
- Task graphs.
- Control flow models.

Models of computation

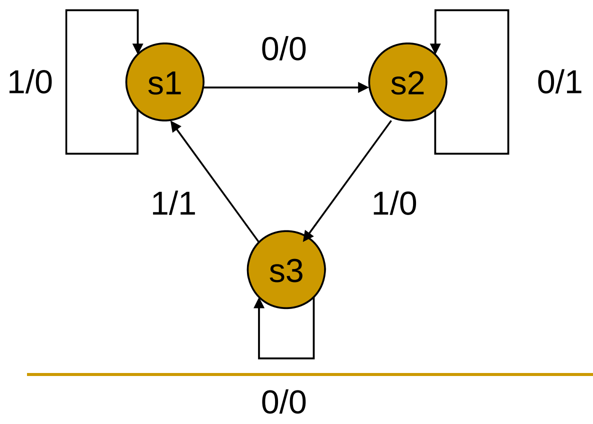
- Models of computation define the basic capabilities of an abstract computer
  - They affect programming style.
- No one model of computation is good for all algorithms.
- Large systems may require different models of computation for different parts.
  - Models must communicate compatibly.

Why is MoC useful?

- It help us to understand the expressiveness of various programming languages.
  - Capability: can (cannot) do something
  - Implication of programming style:
- Language styles
  - Finite versus infinite state
  - Control versus data
  - Sequential versus parallel
- Expressiveness
  - Heterogeneously programmed
  - Interoperability

Finite state machine

- State transition graph and table are equivalent:



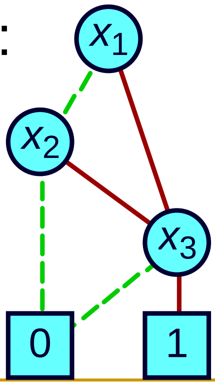
Input	current	next	output
0	s1	s2	0
1	s1	s1	0
0	s2	s2	1
1	s2	s3	0
0	s3	s3	0
1	s3	s1	1

Finite state machine

- Finite state machines
  - M = { I, O, S, Δ, T}, S: current state, Δ: states
  - Moore and Mealy machines
  - Time: integer-valued, not real-valued
  - Stream: a model of terminal behavior – sequential behavior: a totally ordered set of symbols
- Finite state machine properties
  - Finite state.
  - Nondeterministic variant.

Boolean Manipulation with OBDDs

- Ordered Binary Decision Diagrams
- Data structure for representing Boolean functions
- Efficient for many functions found in digital designs
- Canonical representation
- Example:



$(x_1 \vee x_2) \& x_3$

- **Nodes represent variable tests**
- **Branches represent variable values**

  Dashed for value 0

  Solid for value 1

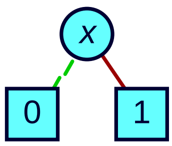
Example OBDDs

Constants

0 **Unique unsatisfiable function**

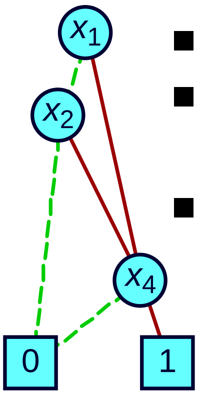
1 **Unique tautology**

Variable



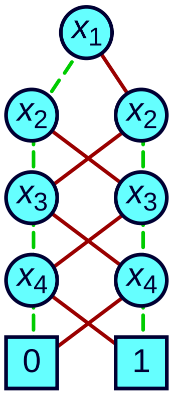
Treat variable as function

Typical Function



- $(x_1 \vee x_2) \& x_4$
- **No vertex labeled** x_3
  - independent of x_3
- **Many subgraphs shared**

Odd Parity

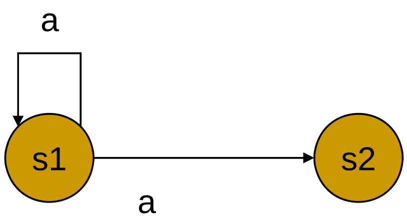


Linear representation

Ordered binary decision diagram (OBDD)

- Allow us to perform many checks that are useful tests of the correctness of practical systems
  - Building product machines which is easier to express complex functions as systems of communicating machines
  - Reachability – many bug manifest themselves as inabilities to reach certain states in the machine.

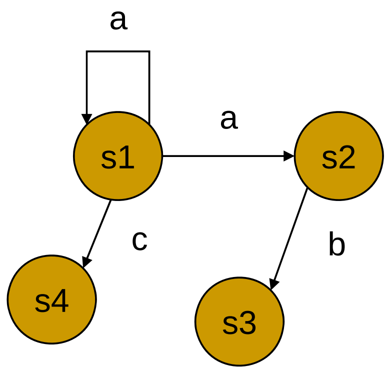
Nondeterministic FSM



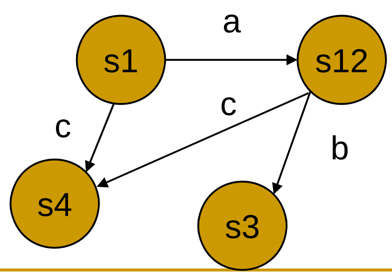
- Several transitions out of a state for a given input.
  - Equivalent to executing all alternatives in parallel.
- Can allow ε moves---goes to next state without input.
- Interpretation
  - The machine nondeterministically choose a transition such that future inputs will cause the machine to be in the proper state
  - The machine follows all possible transition simultaneously until future inputs cause it prune some paths.

Deterministic FSM from nondeterministic FSM

- Add states for the various combinations of nondeterminism.



nondeterministic



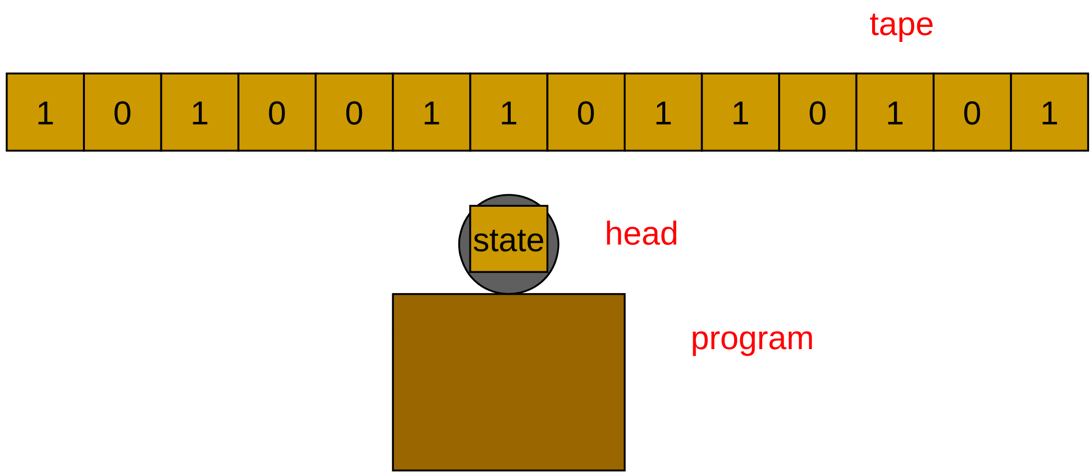
deterministic

Church-Turing Thesis

- Alan Turing invented Turing machines and defined the notion of computable function via these machines.
- Alonzo Church invented a formal system called the lambda calculus and defined the notion of computable function via this system.
- It was proved that both models are equally strong in the sense that they define the same class of computable functions.

Turing machine

- Turing machine: general model of computing:



Turing machine steps

1. Read the current square in the tape.
2. Erase the current square in the tape.
3. Consult its program to determine what to do next. Based on the current state and the symbol that was read, may write a new symbol and/or move the tape.
4. Change its state as described by the program.

Turing machine properties

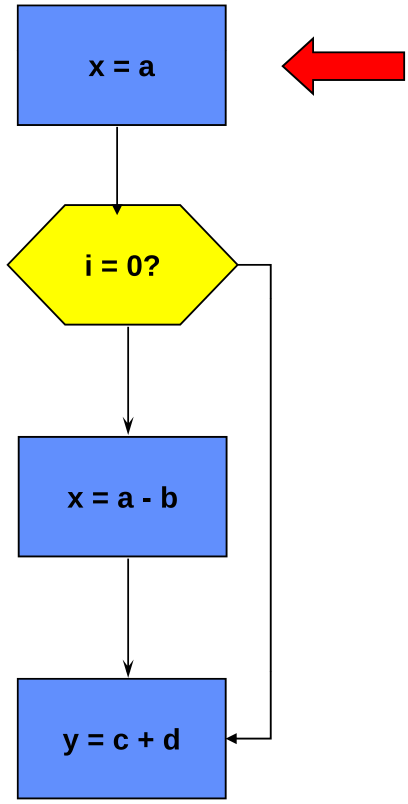
- Example program:
  - If (state = 2 and cell = 0): print 0, move left, state = 4.
  - If (state = 2 and cell = 1): print 1, move left, state = 3.
- Can be implemented on many physical devices.
- Turing machine is a general model of computability.
- Can be extended to probabilistic behavior.

Turing machine properties

- Infinite tape = infinite state machine.
- Basic model of computability.
  - Lambda calculus is an alternative model.
  - Other models of computing can be shown to be equivalent/proper subset of Turing machine.

Control flow graph

- CFG is a finite-state model of computation.
- Commonly used to model program structure.



CDFG properties

- Finite state model.
- Single thread of control.
- Additional data flow models which describe the operation of unconditional nodes
- Can handle subroutines.

DFG

- Tree structure
  - Nodes: data operations
  - Edges: data dependency
  - Sources: Input
  - Sinks: output
- Basic DFG are commonly used in compilers
- Finite state model.
  - It describes parallelism in that it defines only a partial order on the operation in the graph.
  - Any order of operation that satisfies the data dependencies is acceptable.

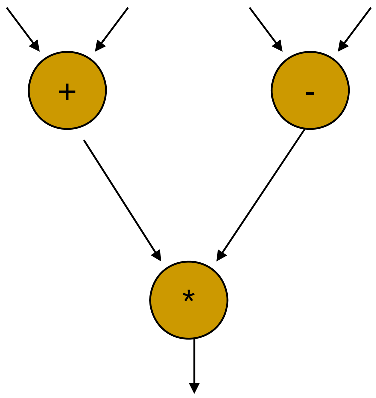
Data flow graph

- Partially-ordered computations:

+ -, *

+, -, *

-, +, *



DFG

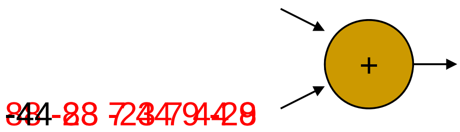
- We can use streams to model the behavior of the DFG.
- The node in the DFG use firing rules to determine their behavior.
  - Standard data flow firing rule: consume a token and generate a token
  - A conditional node with (n+1) terminals: n data input d0, d1, .. And a control input k. When k=1, d1 is consumed and transferred to the output.

DFG

- We can use streams to model the behavior of the DFG.
- The node in the DFG use firing rules to determine their behavior.
  - Standard data flow firing rule: consume a token and generate a token
  - A conditional node with (n+1) terminals: n data input d0, d1, .. And a control input k. When k=1, d1 is consumed and transferred to the output.

Data flow streams

- Captures sequence but not time.
- Totally-ordered set of values.
  - New values are appended at the end as they appear.
- May be infinite.

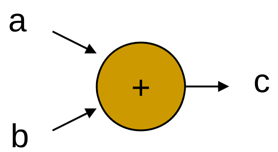


Firing rules

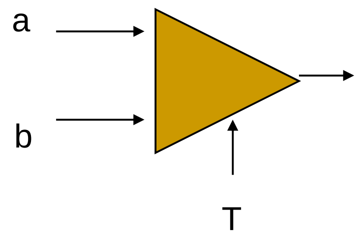
- A node may have one or more firing rules.
- Firing rules determine when tokens are consumed and produced.
  - Firing consumes a set of tokens at inputs, generates token at output.

Example firing rules

- Basic rule fires when tokens are available at all inputs:



- Conditional firing rule depends on control input:



Data flow graph properties

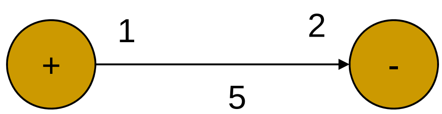
- Finite state model.
- Basic data flow graph is acyclic.
- Scheduling provides a total ordering of operations.

Synchronous data flow

- Lee/Messerschmitt: Relate data flow graph properties to schedulability.
  - Synchronous communication between data flow nodes.
  - Nodes may communicate at multiple rates.

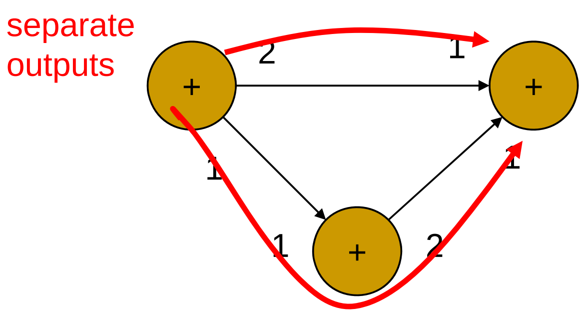
SDF notation

- Nodes may have rates at which data are produced nor consumed.
- Edges may have delays.



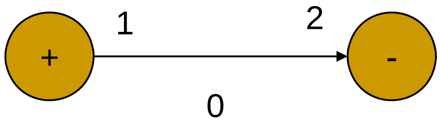
SDF example

- This graph has consistent sample rates:



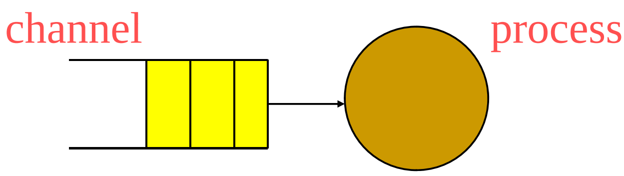
Delays in SDF graphs

- Delays do not change rates, only the amount of data stored in the system.
- Changes system start-up.



Kahn process network

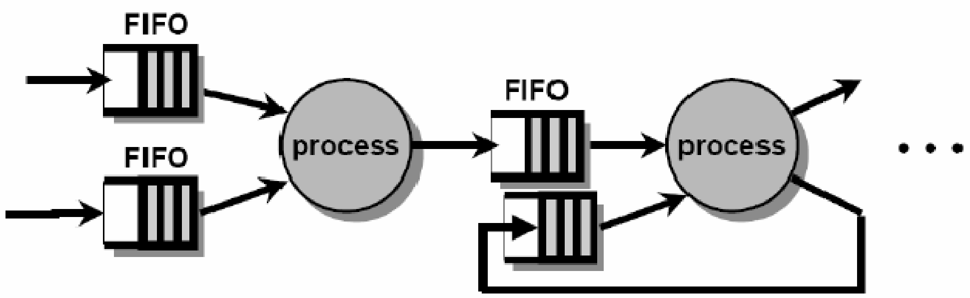
- Process has unbounded FIFO at each input:



  - Each channel carries a possibly infinite sequence or stream.
  - A process maps one or more input sequences to one or more output sequences.
    - Block read / nonblocking write

Properties of processes

p-tuple of sequences	$\mathbf{X}=(X_0, X_1, \ldots X_p) \in \mathbf{S}^p$
ordered set of seq.	$\mathbf{X} \subseteq \mathbf{X}'$ if $X_i \subseteq X_i'$ for each i
set of p-tuple of sequences	$\chi = \{\mathbf{X}_0, \mathbf{X}_1, \ldots\}$
functional process	$F : S^p \rightarrow S^q$



Continuous process $F(\text{lub } \chi) = \text{lub } F(\chi)$

Properties of processes

- Processes are usually required to be continuous: least upper boundedness can be moved across function boundary.
- Monotonicity:
  - $X \subseteq X' \Rightarrow F(X) \subseteq F(X')$

Least fixed point semantics

- Let *X* be the set of all sequences.
- A network is a mapping *F* from the sequences to the sequences (where *I* represents the input sequence):

$$X = F(X, I)$$

- The behavior of the network is defined as the unique *least fixed point* of the equation (*LFP*).
- If *F* is continuous then the least fixed point exists

$$LFP = LUB(\{ F^n(\perp, I) : n \geq 0 \})$$

Least fixed point semantics

- Start with the empty sequence.
- Apply the (monotonic) function.
- Apply the function again to the result.
- Repeat forever.

The result "converges" to the least fixed point.

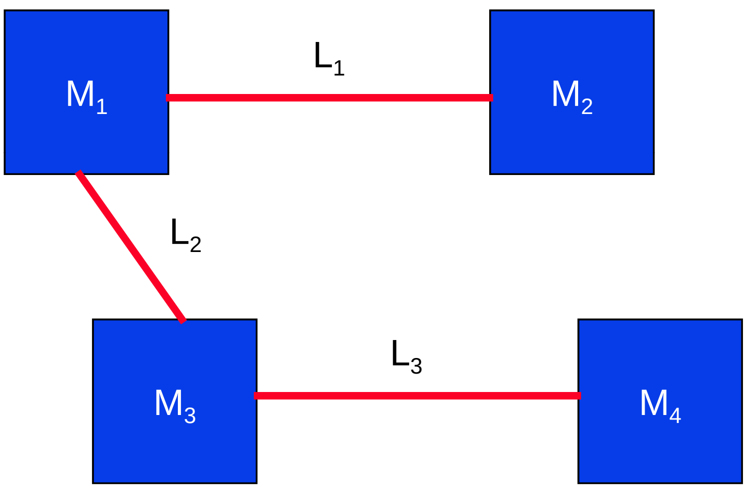
Network properties

- A network of monotonic processes is a monotonic process.
  - Even in the presence of feedback loops.
- Can add nondeterminism in several ways:
  - allow process to test for emptiness;
  - allow process to be internally nondeterminate;
  - allow more than one process to consume data from a channel;
  - etc.

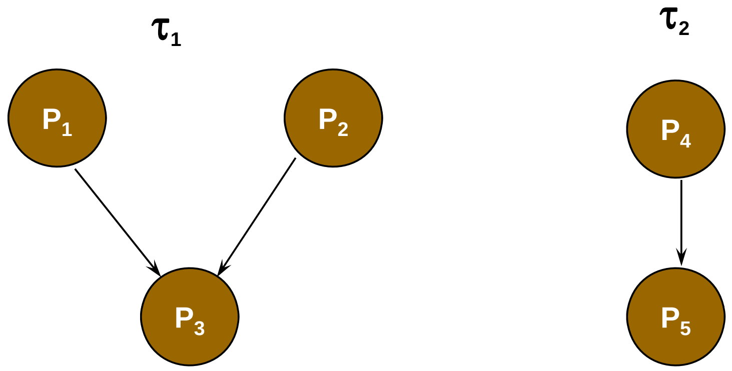
Parallelism and Communication

- Parallelism in hardware must be matched by parallelism in the programs
- Parallel algorithm describe time as partially ordered
  - As we bind operations to the architecture, the description is changed to a totally ordered description.
  - Some operations may be left partially ordered to be managed by the operating system.

Processor graph



Task graph



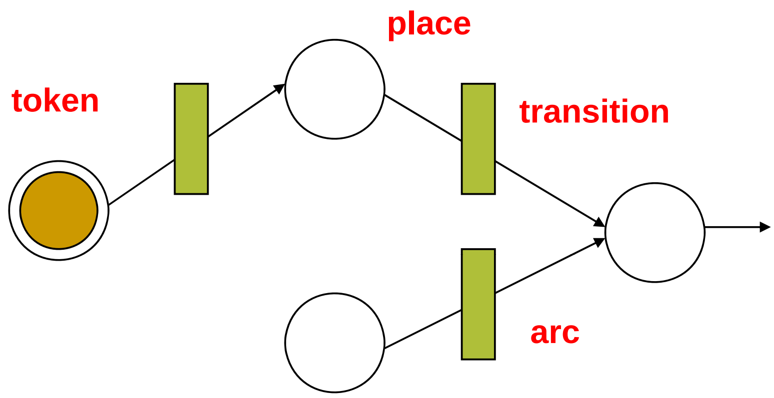
- Task graph is a simple model of parallelism
  - Nodes: processes or tasks
  - Edges: data dependencies
- Used to model multi-rate systems.

Task graph properties

- Not a Turning machine.
  - No branching behavior.
  - May be extended to provide conditionals.
- Possible models of execution time:
  - Constant.
  - Min-max bounds.
  - Statistical.
- Can model late arrivals, early departures by adding dummy processes.

Petri net

- Parallel model of computation: Equivalent with Turing machines
- It is a weighted, directed bipartite graph
  - Place
  - Transition
  - Arc
  - Token



Firing rule

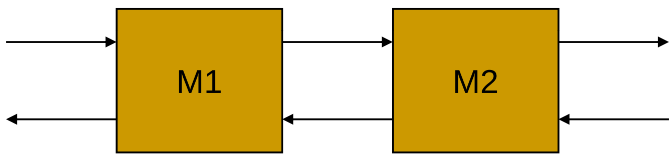
- A transition is enabled if each place at its inputs have at least one token.
  - Enabled transitions may fir but are not required to do so.
  - A transition doesn't have to fire right away.
- Firing a transition removes tokens from inputs and adds a token to each output place.
- In general, may require multiple tokens to enable, which is specified by the weight of each incoming arc

Properties of Petri nets

- Turing complete.
- Arbitrary number of tokens.
  - Nondeterministic behavior.
  - Naturally model parallelism.

Communication styles

- Useful parallelism necessarily involves communication between the parallel components of the system.
- Two types
  - Buffered
  - Unbuffered



- Two communicating FSMs
  - The first step in analyzing the behavior of such networks of FSMs is often to form the equivalent product machine.

Communication

- Synchronous vs. asynchronous
- Blocking vs. nonblocking
- In blocking communication
  - The sender blocks or waits until the receiver has the data.
- In non-blocking communication,
  - If there is no buffer and the receiver is not ready, the sender will drop the data
  - Adding a buffer allows the sender to move on even if the receiver is not ready , assuming that the buffer is not fill.
  - An infinite-size buffer allows unlimited non-blocking communication.
- Buffer sizing is important
  - data rate control with full and empty signals.

Source and Uses of Parallelism

- Parallelism can be found at many different levels of abstraction.
- Instruction-level parallelism
  - It is not visible in the source code and so cannot be manipulated by the programmer
- Data-level parallelism
  - It can be found in a basic block of a program, especially in a nest of loops
- Task-level parallelism
  - particularly important in embedded system because the system often perform several different types of computation on data streams.