Formal Specification

By

Anil Kumar Dudyala

Assistant Professor, Dept of CSE, NIT Patna

(Source: Fundamentals of Software Engineering by Dr. RAJIB Mall)

Contents

- Formal Definitions
- Algebraic specification:
  - Development technique
  - Rewrite rules
  - Example
  - Pros and cons
- State machine modelling
- Summary

Formal Definitions

- **Formal specification techniques:**
  - **model-oriented**
  - **property-oriented**

Formal Definitions

- Property-oriented techniques:
  - axiomatic specification
  - algebraic specification
- Model-oriented techniques:
  - Z, VDM, Petri net, State machine, etc.

Formal Definitions

- **Axiomatic techniques:**
  - **based on early work on program verification.**
  - **Use first-order predicate logic:**
    - **specify operations through pre and post conditions**

Axiomatic specification

- **Example**

$$F(x\colon real)\colon real$$

Pre: $\boldsymbol{x} \in \boldsymbol{R}$

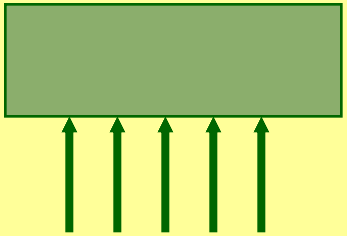
Post: $\left\{(\boldsymbol{x} \le \mathbf{100}) \cap \left(\boldsymbol{f}(\boldsymbol{x}) = \frac{\boldsymbol{x}}{\mathbf{2}}\right)\right\} \cup$
$\{(\mathbf{x} > \mathbf{100}) \cap (\boldsymbol{f}(\boldsymbol{x}) = \mathbf{2} * \boldsymbol{x})\}$

Formal Definitions

- **Algebraic technique:**
  - **data types are viewed as heterogeneous algebra**
  - **axioms are used to state properties of data type operations**

Algebraic Specification

- Using algebraic specification:



  - the meaning of a set of interface procedures is defined by using equations.

Algebraic Specification

- Algebraic specifications are usually presented in four parts:
  - types section
  - exceptions section
  - signature section
  - rewrite rules section

Types Section

- Types Section Lists:
  - sorts (or types) being specified
  - sorts being imported
  - Importing a sort:
    - makes it available in specification.

Exception Section

- Lists names of exceptional conditions used in later sections:
  - under exceptional conditions error should be indicated.

Signature Section

- **Defines signatures of interface procedures:**
  - **e.g. PUSH takes a stack and an element and returns a new stack.**
  - **push:**
    - **stack × element → stack**

Rewrite rules section

- Lists the properties of the operators:
  - In the form of a set of axioms or rewrite rules.
    - allowed to have conditional expressions

Developing Algebraic Specification

- The first step in defining an algebraic specification:
  - identify the set of required operations.
  - e.g. for string identify operations:
    - create, compare, concatenate, length, etc.

Developing Algebraic Specification

- Generally operations fall into 2 classes:
- Constructor Operations :
  - Operations which create or modify entities of the sort e.g., create, update, add, etc.

Developing Algebraic Specification

- **Inspection Operations :**
  - **Operations which evaluate attributes of the sort, e.g., eval, get, etc.**

Developing Algebraic Specification

- **A rule of thumb for writing algebraic specifications:**
  - **first establish constructor and inspection operations**

Developing Algebraic Specifications

- Next, write down axioms:
  - compose each inspection operator over each constructor operator.

Developing Algebraic Specifications

- If there are m constructors and n inspection operators:
  - we should normally have m*n axioms.
  - However, an exception to this rule exists.

Developing Algebraic Specifications

- **If a constructor operation can be defined using other constructors:**
  - **we need to define inspection operations using only primitive constructors.**

Example: Stack

- Let us specify an unbounded stack supporting:
  - push,
  - pop,
  - newstack,
  - top,
  - empty.

Example: Stack

- **Types:**
- **defines stack**
- **uses boolean, element**
- **Exception:**
- **underflow, novalue**

Example: stack

- **Syntax:**
- **push:**
  - **stack × element → stack**
- **pop:**
  - **stack → stack+{underflow}**

Example: stack

- **top:**
  - **stack $\rightarrow$ element+{novalue}**
- **empty:**
  - **stack $\rightarrow$ boolean**
- **newstack:**
  - **$\phi \rightarrow$ stack**

Equations: stack

- pop(newstack)=underflow
- pop(push(s,e))=s
- top(newstack)=novalue
- top(push(s,e))=e
- empty(newstack)=true
- empty(push(s,e))=false

Rewrite rules

- Rewrite rules let you determine:
  - the meaning of any sequence of calls on the stack functions.

Rewrite rules

- Empty(push(pop(push(newstack,e_1)),e_2)):
  - you can eliminate the call on pop by observing:
    - it is of the form pop(push(s,e)).

Rewrite rules

- **After simplification:**
  - **empty(push(newstack,e_2))**
  - **false**

Two important questions

- **Finite termination property:**
  - **Does application of rewrite rules terminate after a finite number of steps?**
  - **We might endlessly go on applying rewrite rules without coming to any conclusion?**

Two important questions

- **Unique termination property:**
  - **Can different sequence in application of the rewrite rules always give the same answer?**
  - **If we choose to simplify different terms of the expression in different experiments:**
    - **shall we always get the same answer?**

Algebraic Specification

- For arbitrary algebraic equations:
  - convergence is undecidable.
- If the r.h.s. of each rewrite rule has fewer terms than the left:
  - rewrite process must terminate.

Auxiliary Functions

- Sometimes development of a specification requires:
  - extra functions not part of the system:
    - to define the meaning of some interface procedures.

Auxiliary Functions: Example

- To specify bounded stacks:
  - need to add a depth function:
    - push returns either a stack or
    - an exception "overflow" when depth is exceeded.

Bounded stack

- In order to specify a bounded stack:
  - we need to make changes to different sections to include auxiliary functions.

Auxiliary Functions

- Syntax:
- push:
  - stack × element ==> stack
- depth:
  - stack → integer

Auxiliary Functions

- Equations:
  - depth(newstack)=0
  - depth(push(s,e))=depth(s)+1
  - push(s,e)=overflow if depth(s) >= Max

Example 2: coord

- **Types:**
  - **sort coord**
  - **imports integer, boolean**

Example: coord

- **Signature:**
  - **create(integer,integer) → coord**
  - **X(coord) → integer**
  - **Y(coord) → integer**
  - **Eq(coord,coord) → boolean**

Example: coord

- **Rewrite rules:**
  - **X(create(x,y))=x**
  - **Y(create(x,y))=y**
  - **Eq(create(x1,y1),create(x2,y2)) = ((x1=x2) and (y1=y2))**

Structured Specifications

- Writing formal specifications is time consuming.
- To reduce effort, we need to reuse specifications:
  - instantiation of generic specifications
  - incremental development of specifications

Specification Instantiation

- Take an existing specification:
  - specified with some generic parameter
  - Instantiate with some sort

Incremental Development

- **Develop specifications for simple sorts:**
  - **using these specify more complex entities.**

Pros and Cons

- **Algebraic specifications have a strong mathematical basis:**
  - **can be viewed as heterogeneous algebra.**

Pros and Cons

- An important shortcoming of algebraic specifications:
  - cannot deal with side effects
  - difficult to use with common programming languages.

Pros and Cons

- Algebraic specifications are hard to understand:
  - also changing a single property of the system
    - may require changing several equations.

Specification of timing constraints

- **Timing constraints:**
  - **expressed in terms of occurrence of certain events.**

Events

- A stimulus to the system from its environment.
- Can also be an externally observable response:
  - that the system makes to its environment
- Events can be instantaneous
  - or assumed to have a duration

Types of timing constraints

- **Performance constraints:**
  - constraints imposed on the response of the system.
- **Behavioral constraints:**
  - constraints imposed on the action and reaction time of the environment (or the user).

Specification of timing constraints

- **We will specify timing constraints:**
  - **in terms of stimuli and response**
  - **modelled as FSMs (Finite State Machines)**

State machine modelling

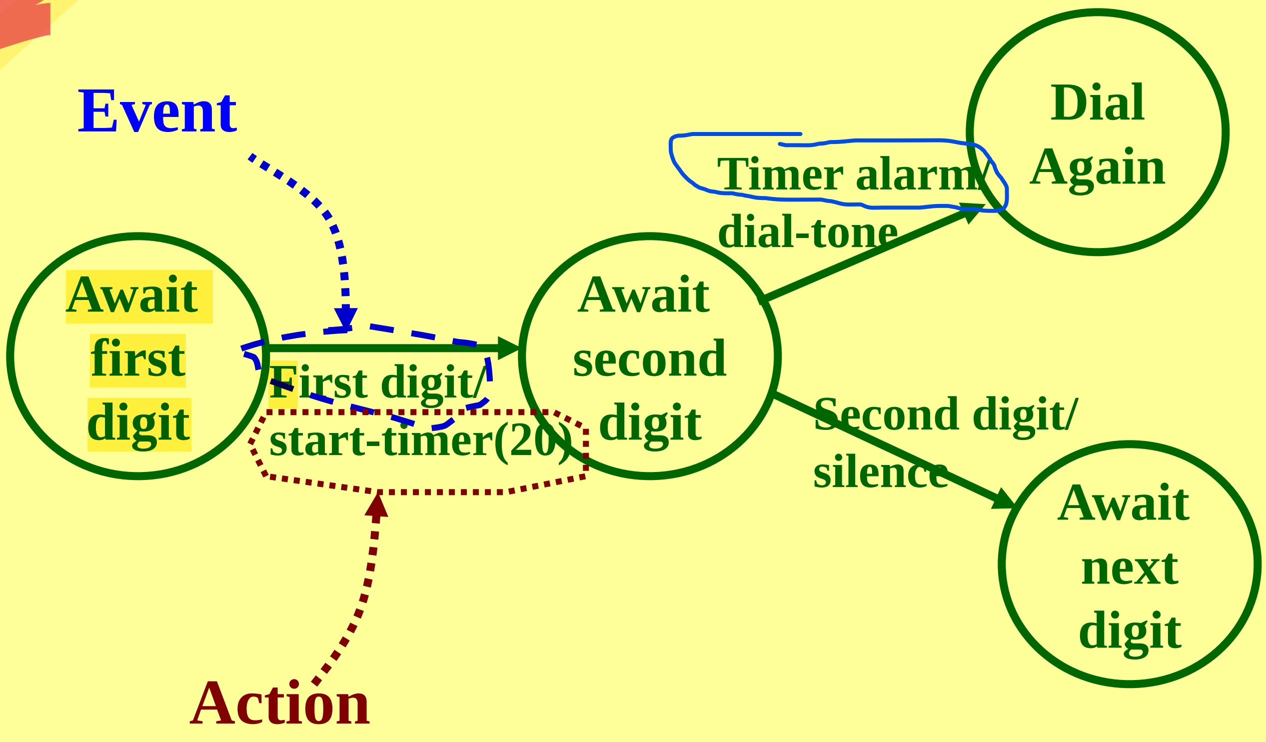
- Assumes that at any time:
  - the system is in one of a number of possible states.
- When a stimulus is received,
  - it may cause a transition to a different state.

Finite Automaton with Output

- A set of states
- final state:
  - some states designated as final states
- an alphabet of input symbols
- an alphabet of output symbols
- a transition function:
  - maps a combination of states and input symbols to states

STATE + INPUT SYMBOLS -> MAP --- STATES

Representation



Types of finite automaton

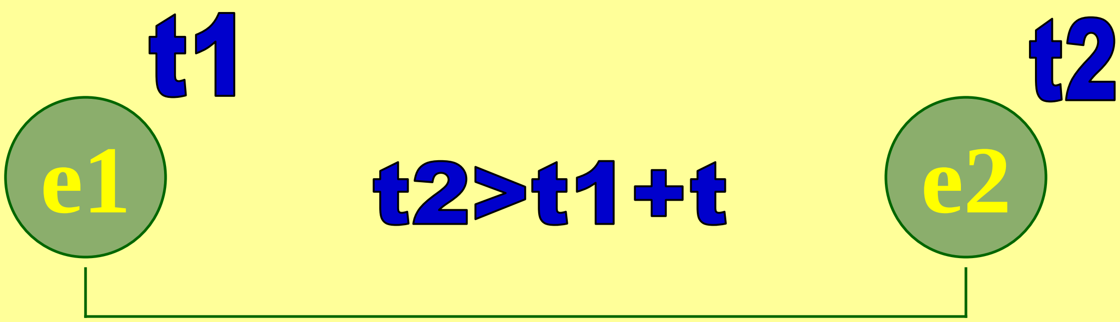
- A finite automaton with output may be organized in two ways:
  - A **Moore machine** is an automaton
    - each state is associated with an output symbol
  - A **Mealy machine** associates each transition with an output symbol

Classification:

- Three types of timing constraints:
  - Minimum
  - Maximum
  - Durational

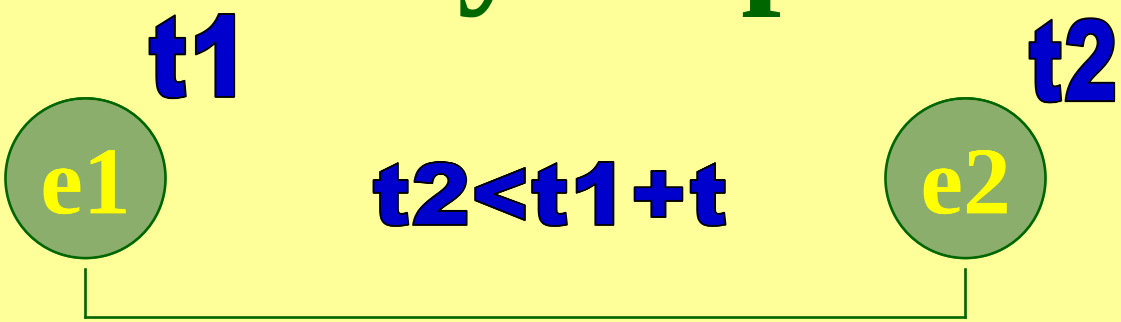
Minimum

- **Between two events:**
  - **No less than t time units may elapse**



Maximum

- **Between two events:**
  - **No more than t time units may elapse**



Durational

- **An event must occur for t units of time**

Maximum

- S-S (stimulus-stimulus)
- S-R (stimulus-response)
- R-S (response-stimulus)
- R-R (response - response)

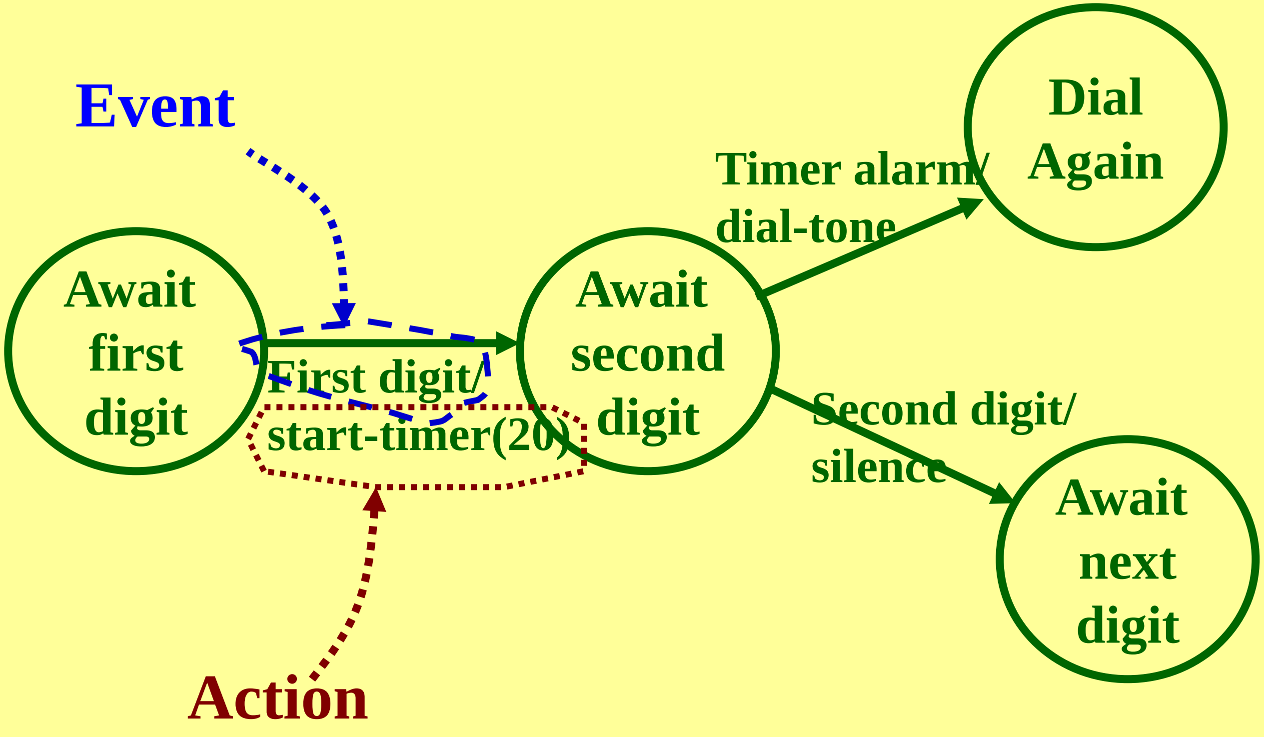
Maximum S-S

- **Maximum time between occurrence of two stimuli:**
  - **e.g. after dialling first digit,**
    - **the second digit should be dialled no more than 20 secs later.**

FSM Representation

- **To represent timing constraints in an FSM:**
  - **a timer alarm is used as an artificial stimulus.**

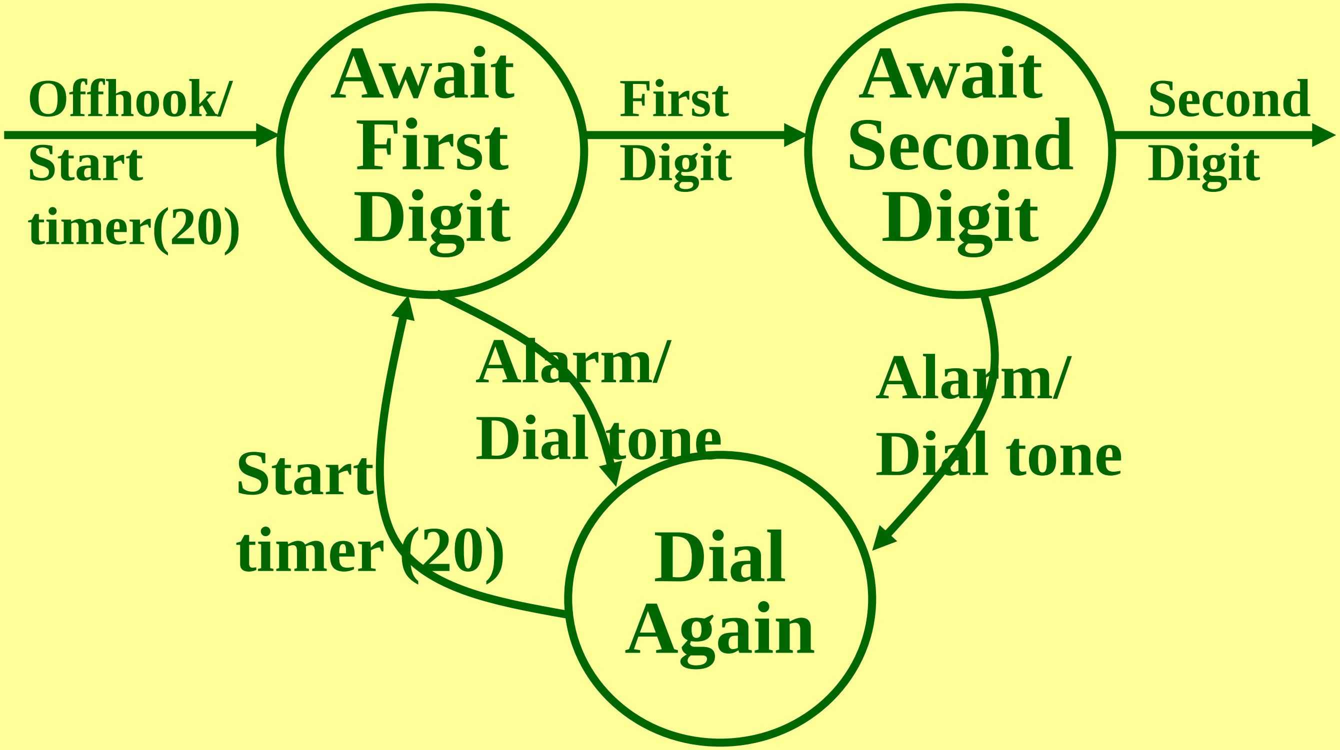
Representation



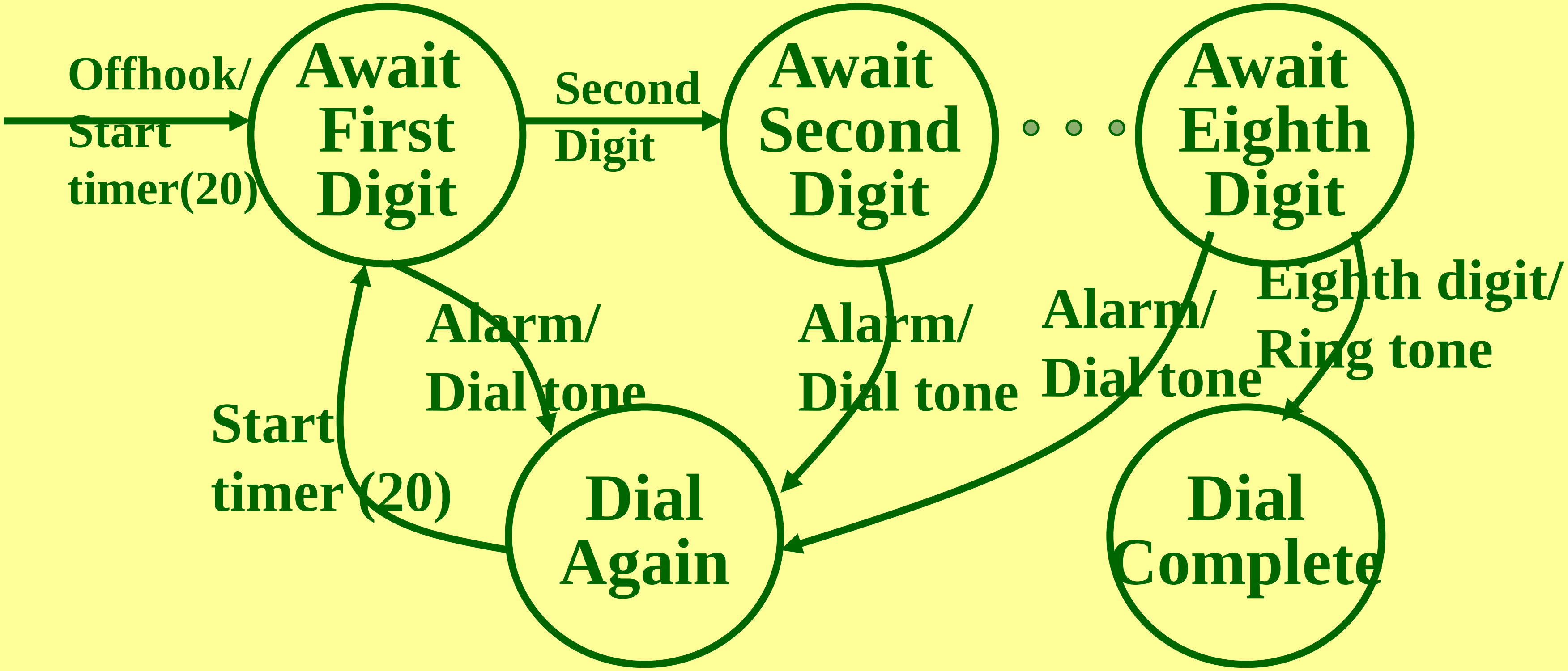
Maximum S-R

- Maximum time between stimulus and response:
  - e.g. caller shall receive dial tone no later than 20 secs after lifting the receiver.

Representation



Complete Representation



Maximum R-S

- **Maximum time between system's response and the next stimulus from the environment:**
  - **e.g after receiving the dial tone, the caller shall dial the first digit within 20 sec.**

Minimum constraints

- S-S
- R-S
- R-R

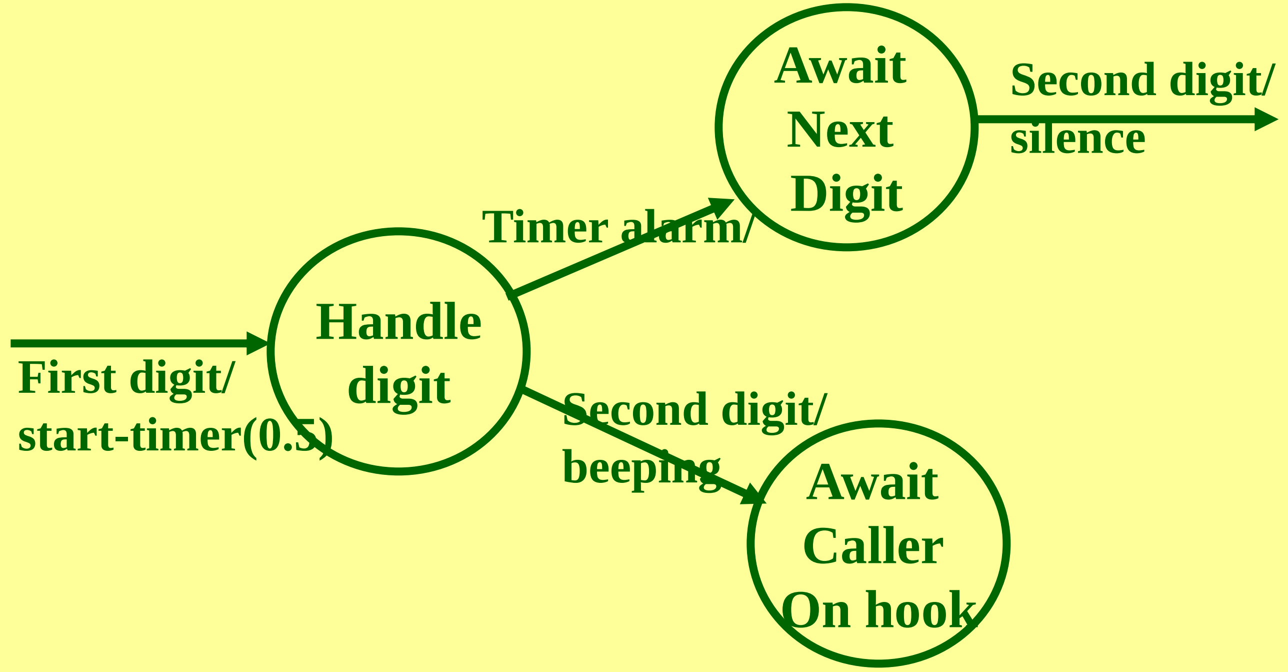
Minimum S-S

- **A minimum time is required between two stimuli:**
  - **e.g. a minimum of 0.5 sec must elapse between the dialling of one digit and the dialling of the next.**

Minimum S-S

- This is an example of behavioral constraints on system users:
  - the complete specification should include
    - response the system should make if the user responds too soon.

Representation



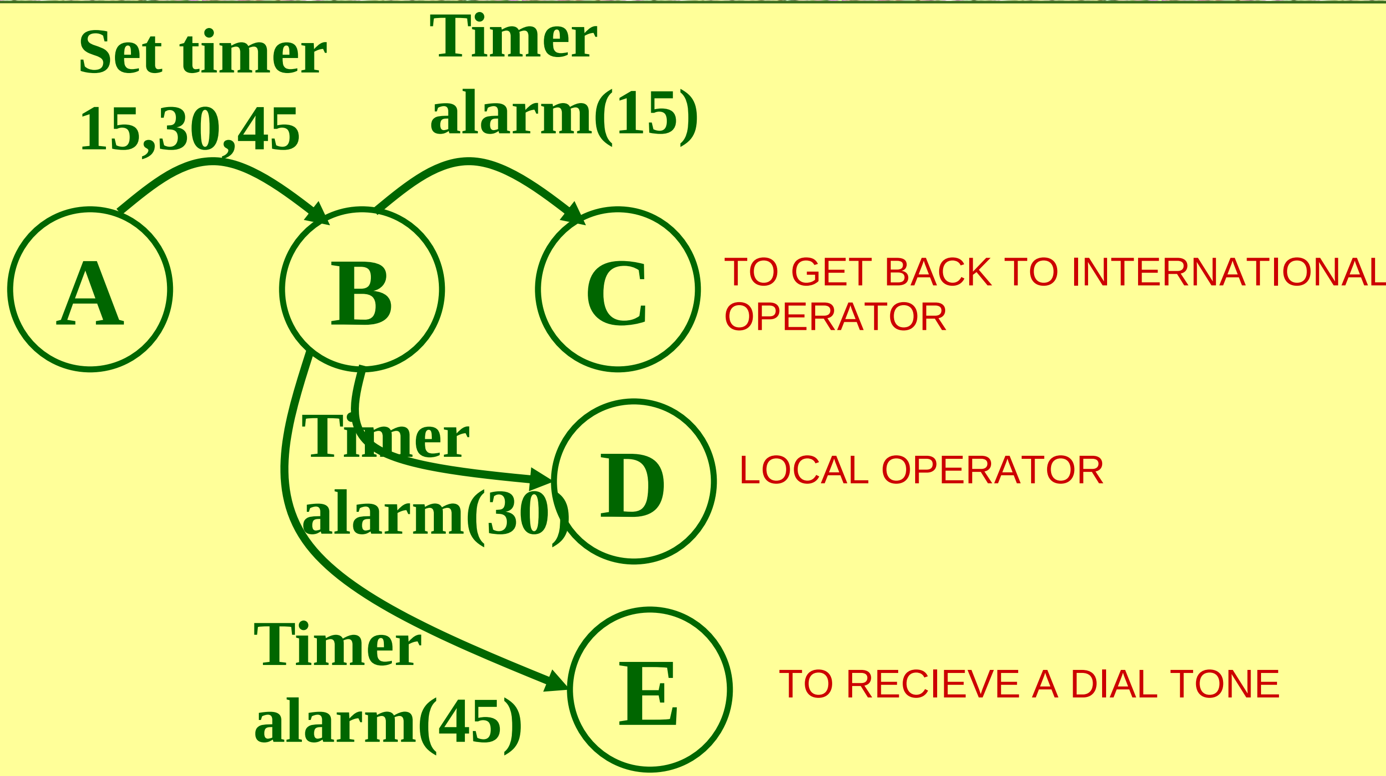
Durational timing constraints

- To go back to the international operator
  - press the button for at least 15 secs (but no more than 30 secs)

Durational timing constraints

- To get back to the local operator:
  - press the button for at least 30 secs, but no more than 45 secs
- To receive a dial tone press the button for at least 45 secs.

Representation



Reference

- Ian Somerville, Chapter Chapter 10
- R. Mall, Chapter 4
- B. Dasarathy, “Timing constraints of R-T systems,” IEEE TSE, 1985, pp. 80--86.

Summary

- **We started by discussing some general concepts in:**
  - **formal specification techniques.**

Summary

- **Formal requirements specifications:**
  - **have several positive characteristics.**
  - **But the major shortcoming is that they are hard to use.**

Summary

- It is possible that formal techniques will become more usable in future
  - with the development of suitable front-ends.

Summary

- We discussed a sample specification technique,
  - algebraic specification
  - gives us a flavour of the issues involved in formal specification.

Summary

- **We discussed specification of timing constraints:**
  - **classification**
  - **modelling using FSMs**

Next Lecture

- Real-time system design:
  - –Structured analysis of system behavior
  - –Ward and Mellor technique