Finite State Machines

Tópicos Avançados em Engenharia de Software

Prof. Dr. Rogério Eduardo Garcia

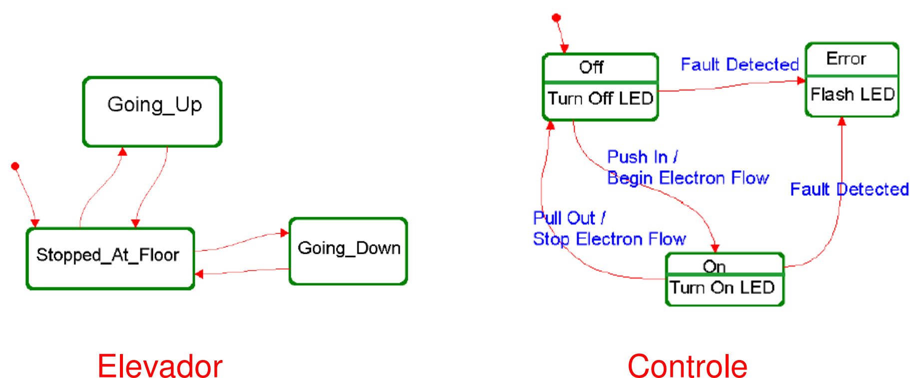
Finite State Machines

- FSM é um modelo matemático de um sistema que assume:
  - O sistema pode ser mapeado em # finito de condições chamados *estados*
  - O comportamento do sistema em um dado estado é sempre o mesmo
  - O sistema permanece em estados por período significante de tempo
  - O sistema pode mudar de estados somente em um # finito de modos, chamados *transições*
  - Transições são a resposta do sistema para *eventos externos ou internos*
  - Funções ou operações chamadas *ações* podem ser executadas quando uma transição acontece, entra em estado, ou sai de um estado
    - Implementado por operações de um objeto
  - Transições e ações levam (aproximadamente) são instantâneas
    - “síncronas”
  - Eventos não permitidos em um estado são ignorados ou resultam em um erro ou, ainda, são enfileirados

Finite State Machines

- FSM = (Inputs, Outputs, States, InitialState, NextState, Outs)
- Adequadas para controladores, protocolos, etc
- Não são completos (Turing), mas mais fáceis de analisar
- Fácil para usar em conjunto com algoritmos de síntese e verificação

FSM Exemplos



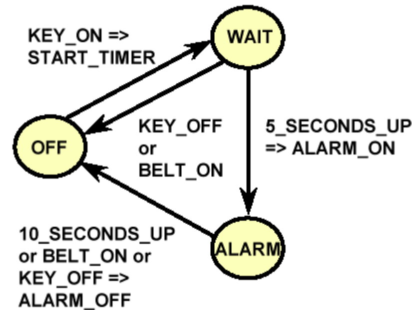
Fonte: B. P. Douglass & iLogix

FSM Exemplo

- Especificação Informal
  se um motorista liga a chave e não põe o cinto de segurança em 5 segundos, o alarme deve disparar por 5 segundos ou até que o cinto seja colocado, ou a chave seja desligada
- Representação Formal

```
Inputs = {KEY_ON, KEY_OFF, BELT_ON,
BELT_OFF, 5_SECONDS_UP, 10_SECONDS_UP}
Outputs = {START_TIMER, ALARM_ON,
ALARM_OFF}
States = {Off, Wait, Alarm}
Initial State = off
NextState: CurrentState, Inputs ->
NextState
e.g. NextState(WAIT, {KEY_OFF}) = OFF

Outs: CurrentState, Inputs -> Outputs
e.g. Outs(OFF, {KEY_ON}) = START_TIMER
```

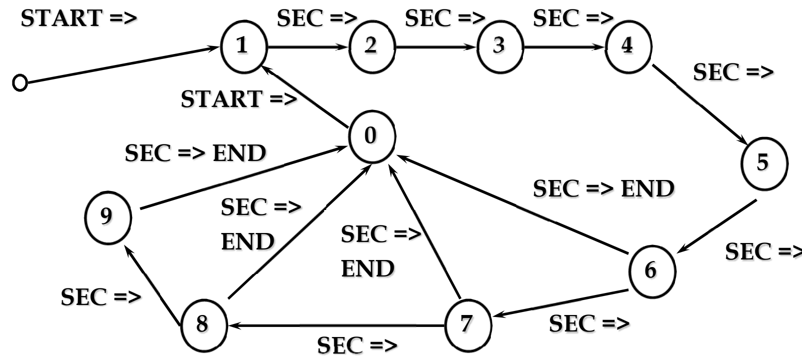


FSM Não-Determinística

- Uma FSM é dita não-determinística quando o estado seguinte (*NextState)* e as funções de saída podem ser relações (ao invés de funções)
- Não-determinismo pode ser mais fácil de modelar:
  - Comportamento não especificado
    - Especificação Incompleta
  - Comportamento não-conhecido
    - e.g., o modelo do ambiente

NDFSM: time range

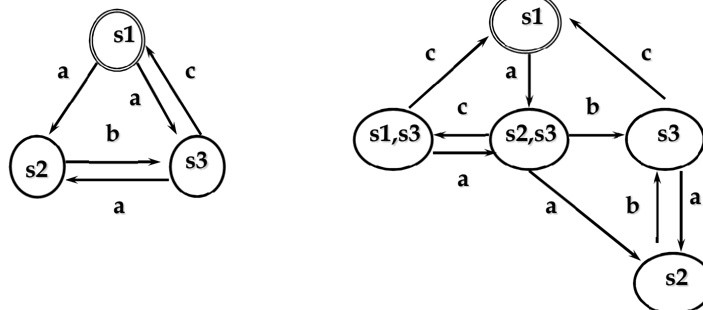
- Special case of unspecified/unknown behavior, but so common to deserve special treatment for efficiency
- E.g. delay between 6 and 10 s



Are NDFSM and FSM equivalent?

NDFSMs and FSMs

- Formally FSMs and NDFSMs are equivalent
  - (Rabin-Scott construction, Rabin '59)
- In practice, NDFSMs are often more compact
  - (exponential blowup for determinization)



Características de FSMs

- O conjunto de estados define o espaço de estados válidos
- Espaço de estados são *flat*
  - Todos os estados estão no mesmo nível de abstração
  - Todos os nomes de estados são únicos
- Modelos de Estados são *single thread*
  - Somente um único estado pode ser válido em um tempo *t*
- Todas as ações são sobre um estado de entrada
  - Não-reativa (resposta demora um ciclo)
  - Fácil de compor (sempre bem definido)
  - Bom para implementação
- Todas ações estão em transições
  - Reativa (tempo de resposta é zero)

Source: B. P. Douglass & iLogix

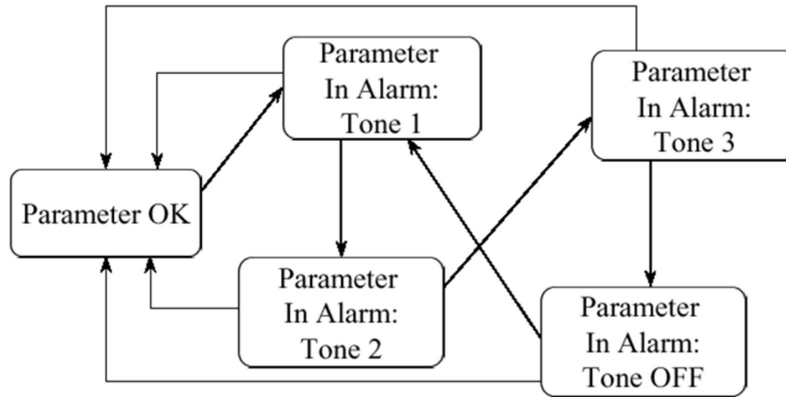
Problemas com FSM Convensional

- Superespecificado
  - Sequencia completamente especificada
- Escalabilidade – devido à falta de metáfora para decomposição
  - Número de estados pode ser difícil de gerenciar
- Não suporta concorrência
- Não suporta para conexões ortogonais

Source: B. P. Douglass & iLogix

Escalabilidade

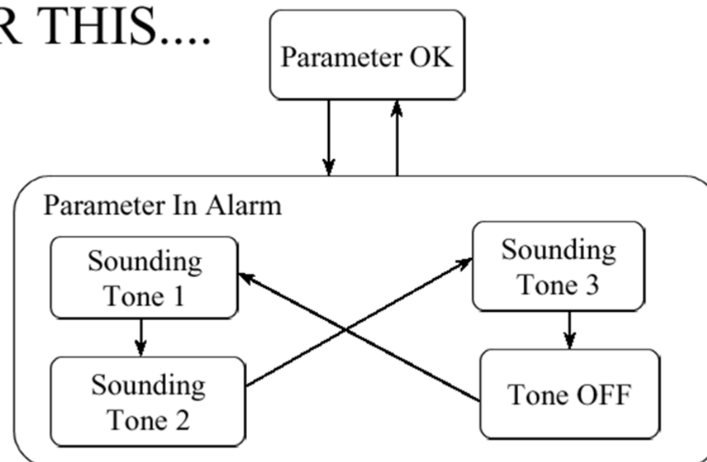
THIS....



Source: B. P. Douglass & iLogix

Escalabilidade

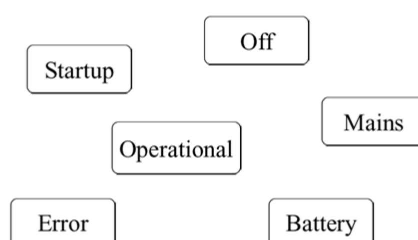
OR THIS....



Source: B. P. Douglass & iLogix

Concorrência

- Problema:
  - Um dispositivo pode estar em estados
    - Off, Starting-up, Operational, Error
  - E pode estar em execução usando
    - mains, battery
- Como organizar esses estados?



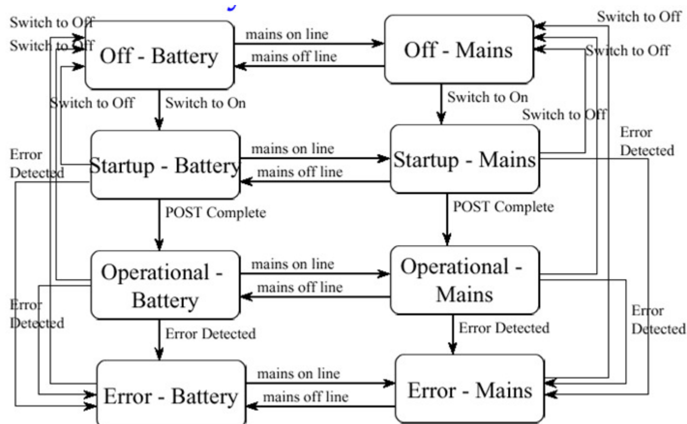
Fonte: B. P. Douglass & iLogix

Concorrência

- Estados “combinados” :
  - *Operation* com *battery*
  - *Operation* com *mains*
- Leva à explosão de estados
- Solução?
  - Permitir estados para operar concorrentemente

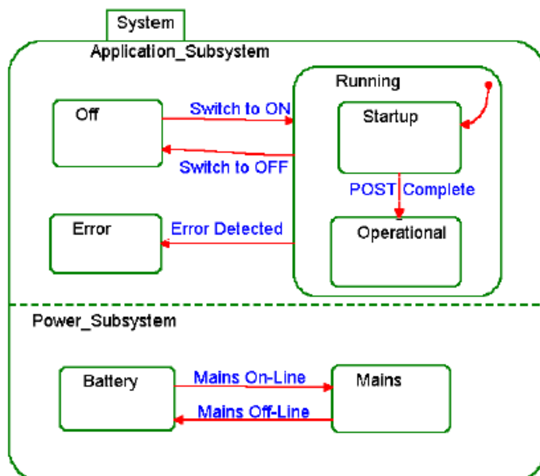
Source: B. P. Douglass & iLogix

Solução



Source: B. P. Douglass & iLogix

Solução: modelo de estados concorrentes



Source: B. P. Douglass & iLogix

Componentes Ortogonais

myInstance: myClass	
tColor	Color
boolean	ErrorStatus
tMode	Mode

enum tColor {eRed, eBlue, eGreen};

enum boolean {TRUE, FALSE}

enum tMode {eNormal, eStartup, eDemo}

Como modelar os estados desse objeto?

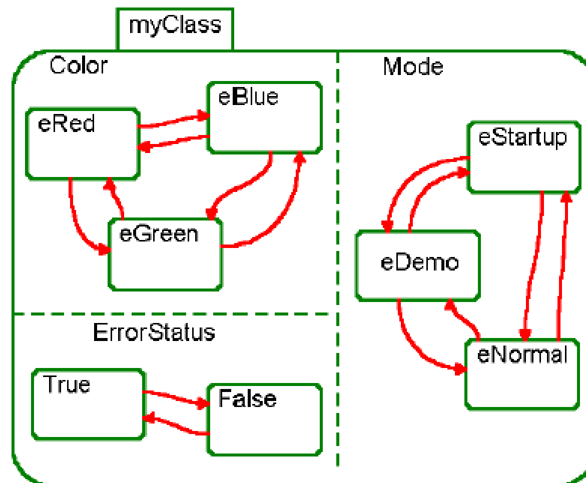
Source: B. P. Douglass & iLogix

Abordagem 1: Enumerar todos

eRed, FALSE, eDemo	eBlue, FALSE, eDemo	eGreen, FALSE, eDemo
eRed, TRUE, eDemo	eBlue, TRUE, eDemo	eGreen, TRUE, eDemo
eRed, FALSE, eNormal	eBlue, FALSE, eNormal	eGreen, FALSE, eNormal
eRed, TRUE, eNormal	eBlue, TRUE, eNormal	eGreen, TRUE, eNormal
eRed, FALSE, eStartup	eBlue, FALSE, eStartup	eGreen, FALSE, eStartup
eRed, TRUE, eStartup	eBlue, TRUE, eStartup	eGreen, TRUE, eStartup

Source: B. P. Douglass & iLogix

Abordagem 2



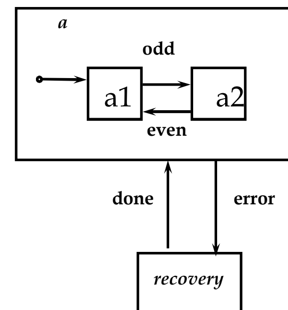
Source: B. P. Douglass & iLogix

StateCharts

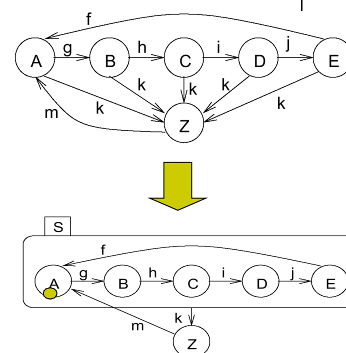
- Conventional FSMs are inappropriate for the behavioral description of complex control
  - Flat and unstructured
  - Inherently sequential in nature
  - Give rise to an exponential blow-up in # of states
    - Small system extensions cause unacceptable growth in the number of states to be considered
- StateCharts support:
  - *Repeated decomposition* of states into AND/OR sub-states
    - Nested states, concurrency, orthogonal components
  - *Actions* (may have parameters)
  - *Activities* (functions executed as long as state is active)
  - *Guards*
  - History
  - A *synchronous* (instantaneous broadcast) comm. mechanism

Hierarchical FSM models

- Problem: how to reduce the size of the representation?
- Harel's classical papers on StateCharts (language) and bounded concurrency (model): 3 *orthogonal exponential reductions*
- Hierarchy:
  - State *a* "encloses" an FSM
  - Being in *a* means FSM in *a* is active
  - States of *a* are called OR states
  - Used to model pre-emption and exceptions
- Concurrency:
  - Two or more FSMs are simultaneously active
  - States are called AND states
- Non-determinism:
  - Used to abstract behavior



Introducing hierarchy



- Classical automata not useful for complex systems (complex graphs cannot be understood by humans).
- Introduction of hierarchy
- StateCharts [Harel, 1987]

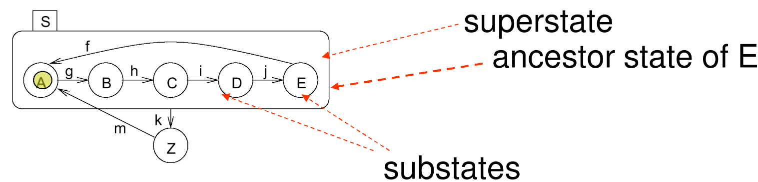
FSM will be **in** exactly one of the substates of S if S is **active**
(either in A or in B or ..)

Features of StateCharts

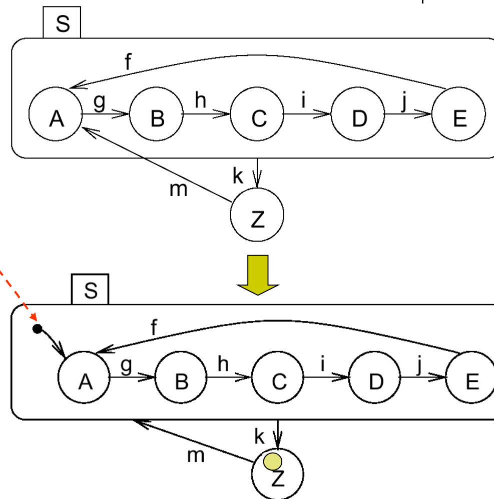
- Nested states and hierarchy
  - Improves scalability and understandability
  - helps describing preemption
- Concurrency - two or more states can be viewed as simultaneously active
- Nondeterminism - there are properties which are irrelevant

Definitions

- Current states of FSMs are also called **active** states.
- States which are not composed of other states are called **basic states.**
- States containing other states are called **super-states**.
- For each basic state s, the super-states containing s are called **ancestor states**.
- Super-states S are called **OR-super-states**, if exactly one of the sub-states of S is active whenever S is active.

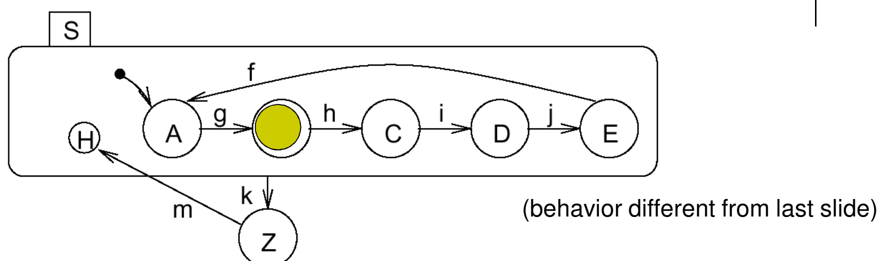


Default state mechanism



- Try to hide internal structure from outside world!
- Default state
- Filled circle indicates sub-state entered whenever super-state is entered.
- Not a state by itself!

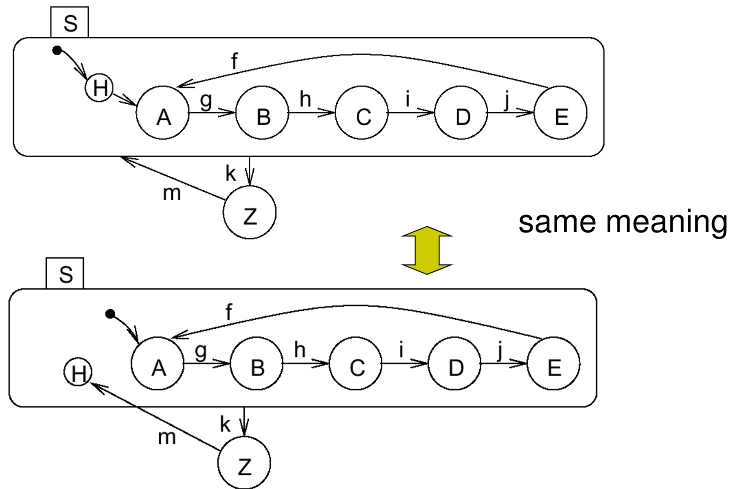
History mechanism



(behavior different from last slide)

- For input m, S enters the state it was in before S was left (can be A, B, C, D, or E). If S is entered for the very first time, the default mechanism applies.
- History and default mechanisms can be used hierarchically.

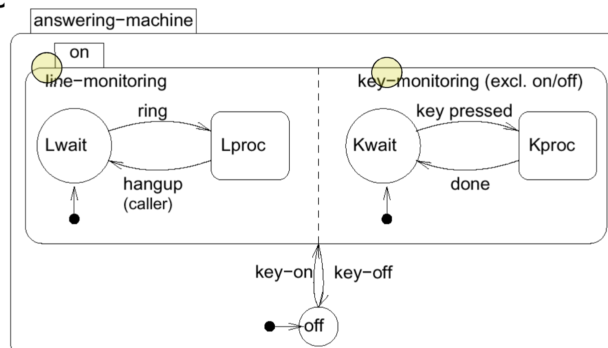
Combining history and default state mechanism



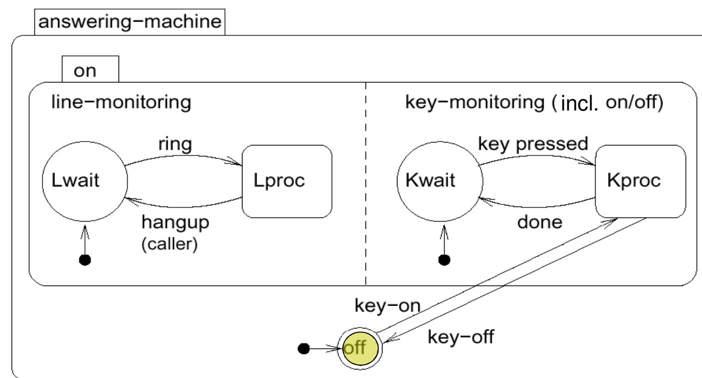
same meaning

Concurrency

- Convenient ways of describing concurrency are required.
- **AND-super-states**: FSM is in **all** (immediate) sub-states of a super-state; Example:

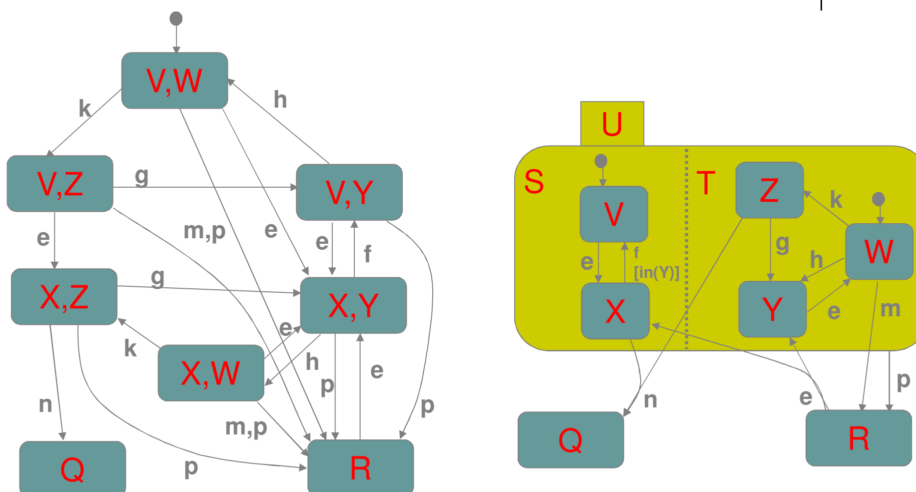


Entering and leaving AND-super-states



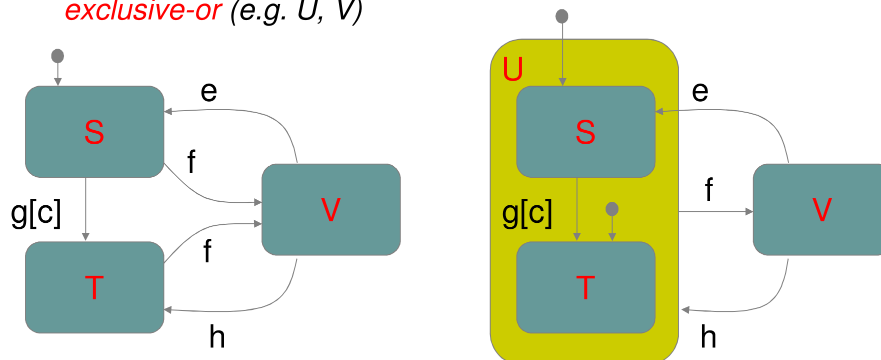
- Line-monitoring and key-monitoring are entered and left, when service switch is operated.

Benefits of AND-decomposition



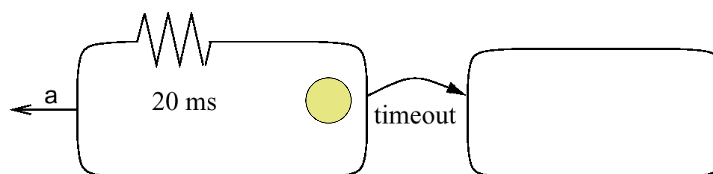
AND/OR State Comparison

- *AND-states* have *orthogonal state components*
  - AND-decomposition can be carried out on any level of states
    - more convenient than allowing only one level of communicating FSMs
- *OR-states* have sub-states that are related to each other by *exclusive-or (e.g. U, V)*



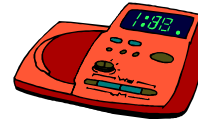
Timers

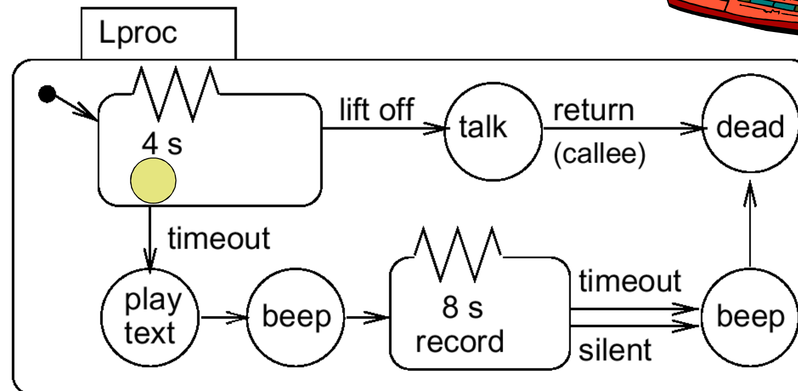
- Since time needs to be modeled in embedded systems,
- timers need to be modeled.
- In StateCharts, special edges can be used for timeouts.



If event a does not happen while the system is in the left state for 20 ms, a timeout will take place.

Using timers in answering machine

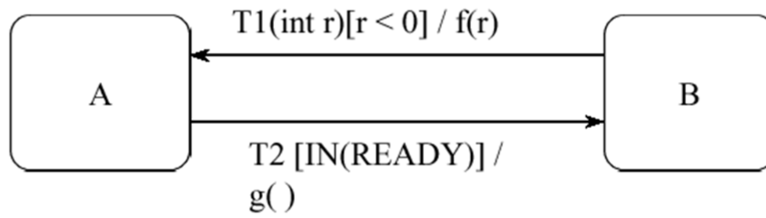




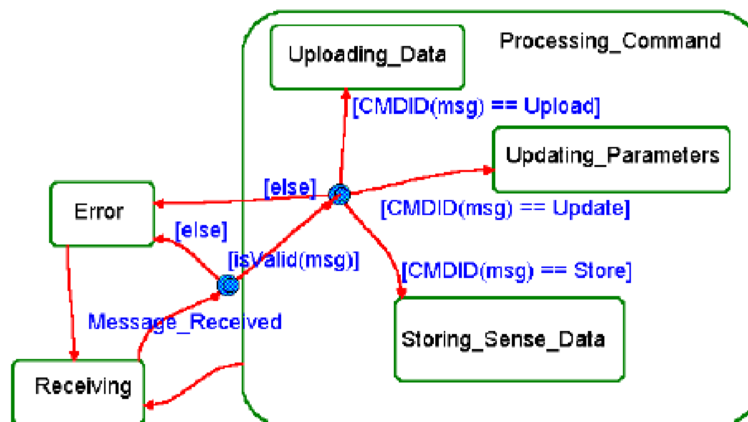
General form of edge labels

- The general syntax of an expression labeling a transition in a StateChart is *n[c]/a*, where
  - *n* is the *event* that triggers the transition
  - *c* is the *condition* that guards the transition (cannot be taken unless c is true when e occurs)
  - *a* is the *action* that is carried out if and when the transition is taken
- Alternative: *name(params)[guards]^event_list/action_list*
  - Event list, aka propagated transitions, is a list of transitions that occur in other concurrent state machines because of this transitions
- For each transition label, event condition and action are optional
  - an event can be the changing of a value
  - standard comparisons are allowed as conditions and assignment statements as actions

Transitions



Conditional Transitions



Source: B. P. Douglass & iLogix

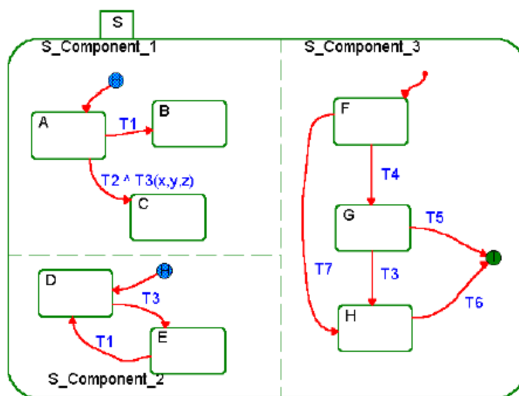
StateCharts Actions and Events

- An action *A* on the edge leaving a state may also appear as an event triggering a transition going into an orthogonal state
  - Executing the first transition will immediately cause the second transition to be taken simultaneously
- Actions and events may be associated to the execution of orthogonal components:
  - action *start(A)* causes activity A to start
  - event *stopped(B)* occurs when activity B stops
  - *entered(S), exited(S), in(S)* etc.

Communication in Concurrent FSMs

- Broadcast events
  - Events are received by more than one concurrent FSM
  - Results in transitions of the same name in different FSM
- Propagated transitions
  - Transitions which are generated as a result of transitions in other FSMs

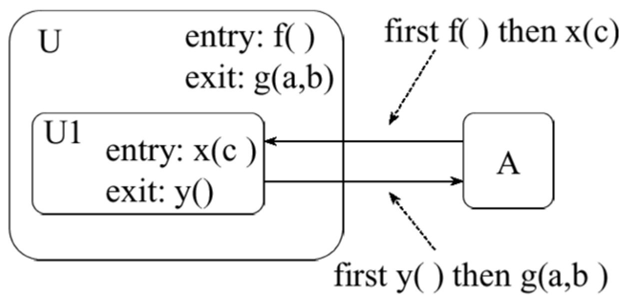
Propagations and Broadcasts



Source: B. P. Douglass & iLogix

Order of Nested Actions

- Executed from outermost – in on entry
- Executed from innermost – out on exit



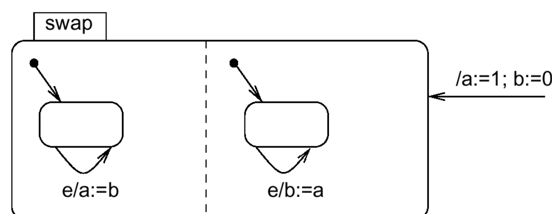
Source: B. P. Douglass & iLogix

The StateCharts simulation phases (StateMate Semantics)

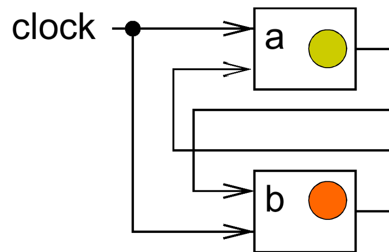
- How are edge labels evaluated?
- Three phases:
  - Effect of external changes on events and conditions is evaluated,
  - The set of transitions to be made in the current step and right hand sides of assignments are computed,
  - Transitions become effective, variables obtain new values.
- Separation into phases 2 and 3 guarantees deterministic and reproducible behavior.

Example

- In phase 2, variables a and b are assigned to temporary variables. In phase 3, these are assigned to a and b. As a result, variables a and b are swapped.
- In a single phase environment, executing the left state first would assign the old value of b (=0) to a and b. Executing the right state first would assign the old value of a (=1) to a and b. The execution would be nondeterministic.



Reflects model of clocked hardware



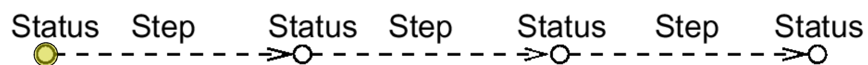
- In an actual clocked (synchronous) hardware system, both registers would be swapped as well.

Same separation into phases found in other languages as well, especially those that are intended to model hardware.

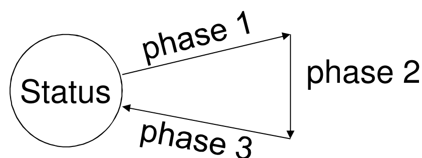
Steps

- Execution of a StateChart model consists of a

Status Step Status Step Status Step Status



Status= values of all variables + set of events + current time
Step = execution of the three phases (StateMate semantics)



Evaluation of StateCharts

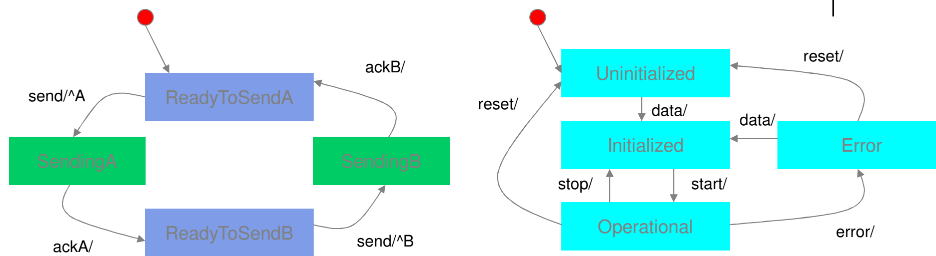
- **Pros:**
- Hierarchy allows arbitrary nesting of AND- and OR-super states.
- (StateMate-) Semantics defined in a follow-up paper to original paper.
- Large number of commercial simulation tools available
  (StateMate, StateFlow, BetterState, ...)
- Available „back-ends“ translate StateCharts into C or VHDL, thus enabling software or hardware implementations.

Evaluation of StateCharts

- **Cons:**
- Generated C programs frequently inefficient,
- Not useful for distributed applications,
- No program constructs,
- No description of non-functional behavior,
- No object-orientation,
- No description of structural hierarchy.

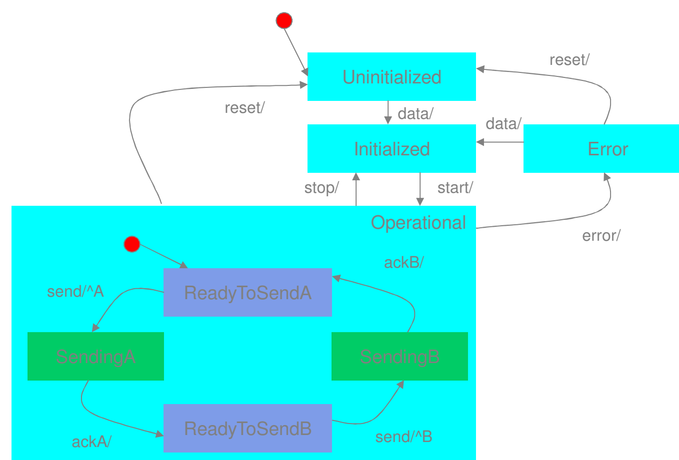
Extensions:
Module charts for description of structural hierarchy.

Example of the Power of StateChart Formalism



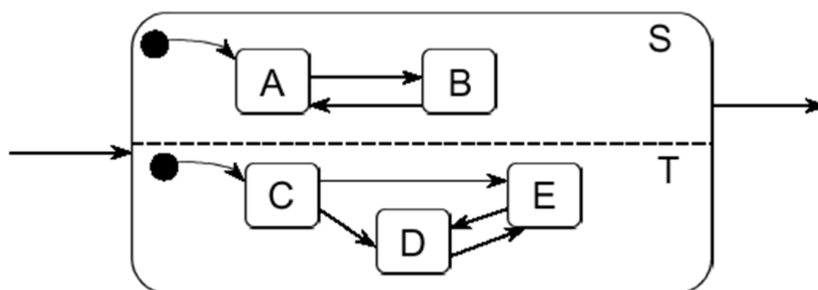
- Conflicting function & control behaviors
  - Function: primary service of the entity
  - Control: actions performed within the system context
- Solutions: single automaton, two peer concurrent states

The Combined State Machine in StateChart Formalism



Concurrent Statecharts

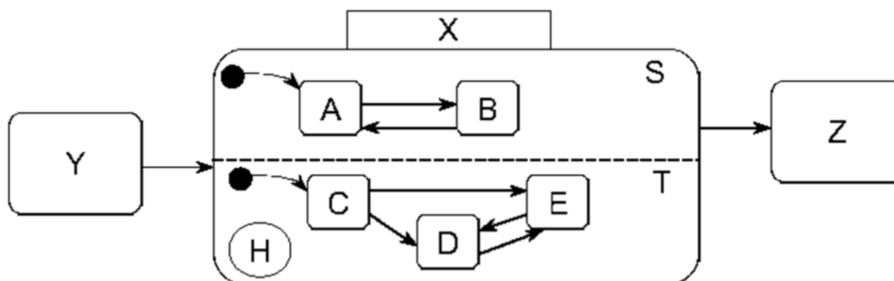
- Many embedded systems consist of multiple threads, each running an FSM
- State charts allow the modeling of these parallel threads



Source: B. P. Douglass & iLogix

Concurrent Statecharts

- States S and T are active at the same time as long as X is active
  - Either S.A or S.B must be active when S is active
  - Either T.C, T.D or T.E must be active when T is active

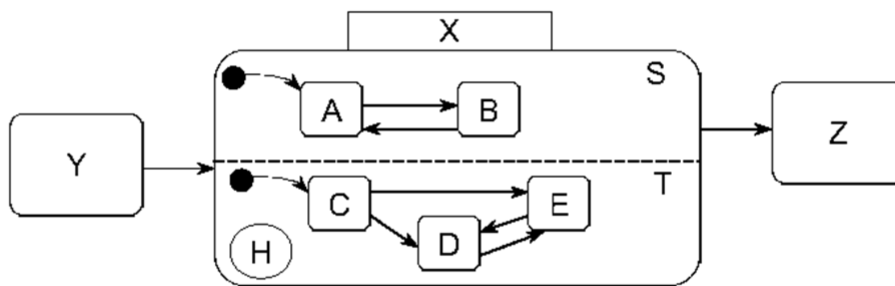


Source: B. P. Douglass & iLogix

Concurrent Statecharts

unesp

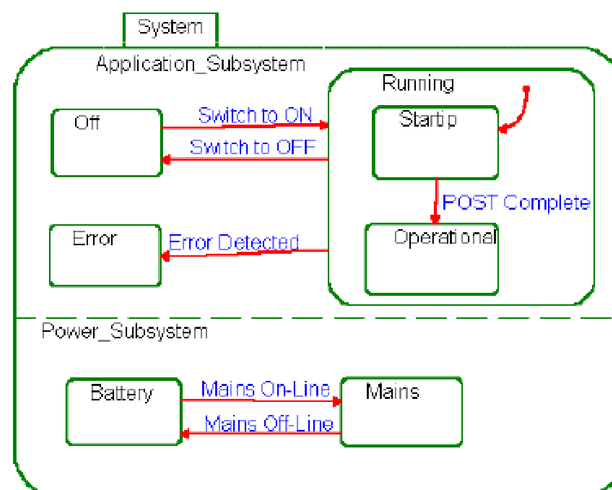
- When X exits, both S and T exit
  - If S exits first, the FSM containing X must wait until T exits
  - If the two FSMs are always independent, then they must be enclosed at the highest scope



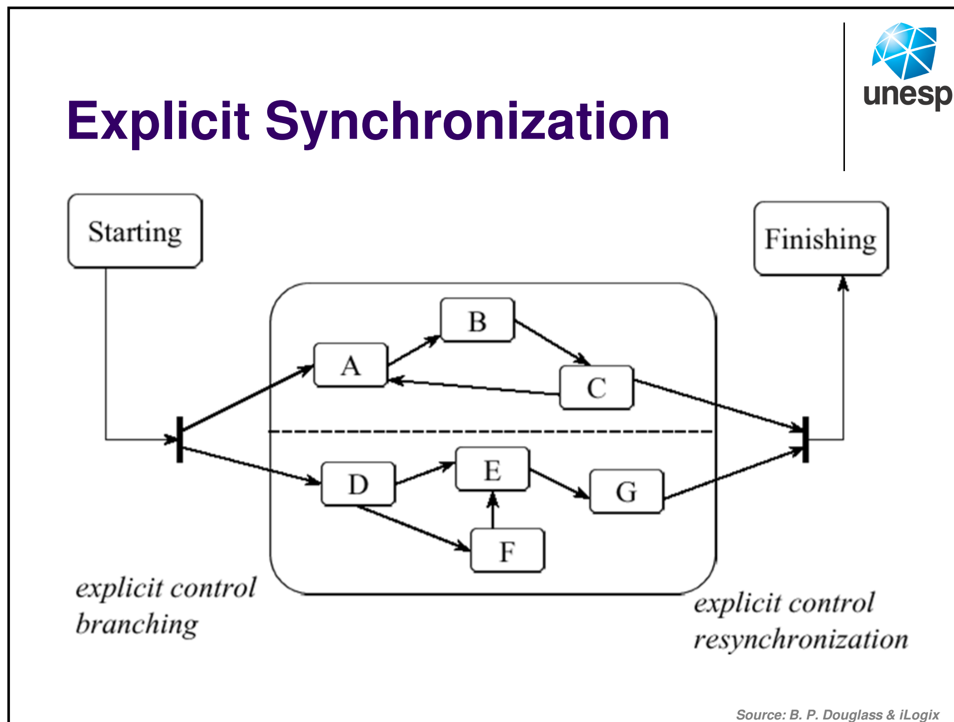
Source: B. P. Douglass & iLogix

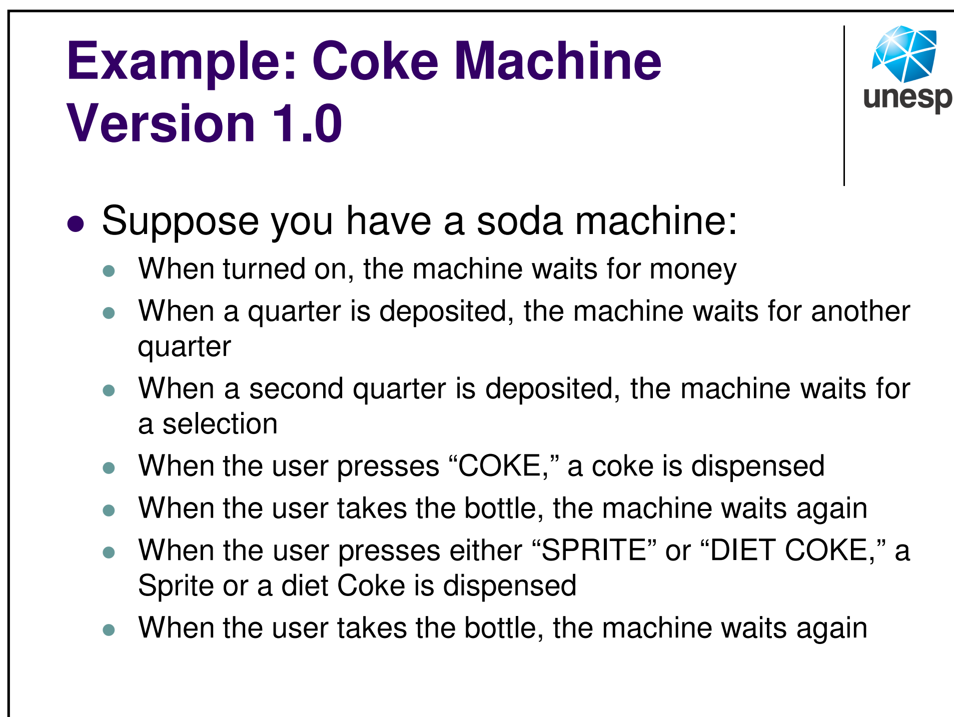
Example Concurrent FSM

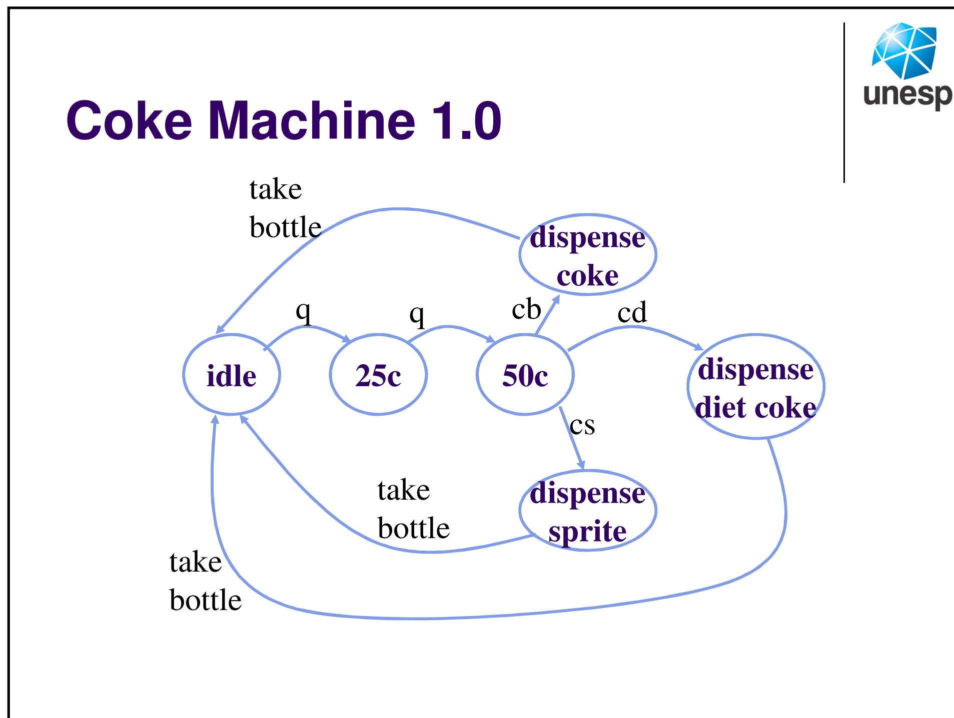
unesp



Source: B. P. Douglass & iLogix


Explicit Synchronization
unesp
Starting
Finishing
A
B
C
D
E
F
G
explicit control branching
explicit control resynchronization
Source: B. P. Douglass & iLogix


Example: Coke Machine Version 1.0
unesp
Suppose you have a soda machine:
When turned on, the machine waits for money
When a quarter is deposited, the machine waits for another quarter
When a second quarter is deposited, the machine waits for a selection
When the user presses “COKE,” a coke is dispensed
When the user takes the bottle, the machine waits again
When the user presses either “SPRITE” or “DIET COKE,” a Sprite or a diet Coke is dispensed
When the user takes the bottle, the machine waits again

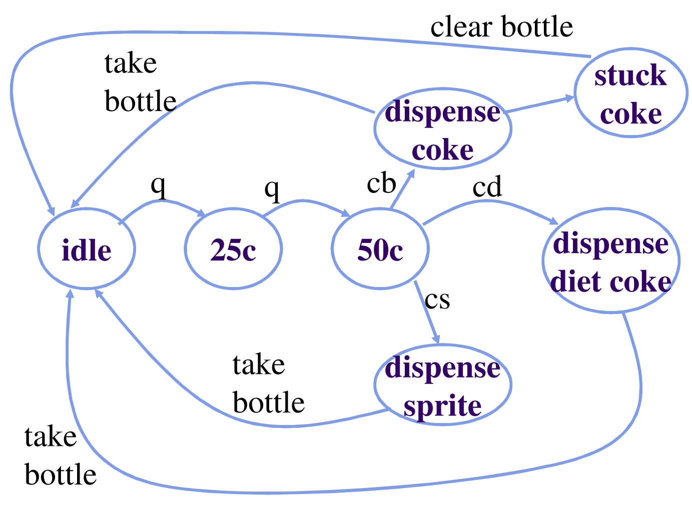


Coke Machine, Version 2.0

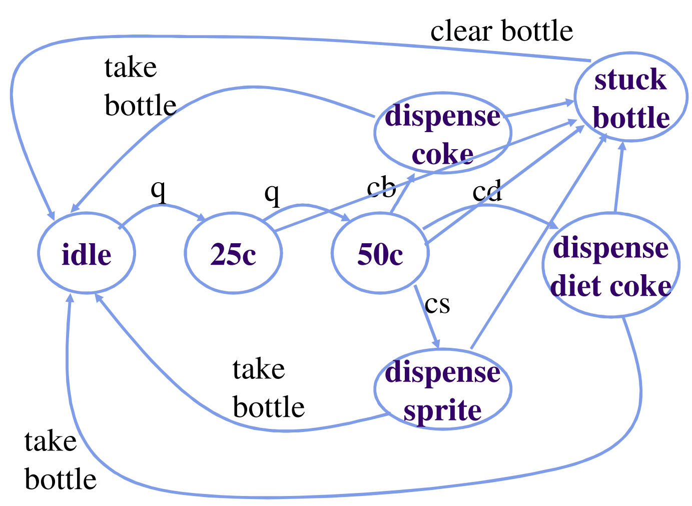
unesp

- Bottles can get stuck in the machine
  - An automatic indicator will notify the system when a bottle is stuck
  - When this occurs, the machine will not accept any money or issue any bottles until the bottle is cleared
  - When the bottle is cleared, the machine will wait for money again
- State machine changes
  - How many new states are required?
  - How many new transitions?

Coke Machine, Version 2.0



Coke Machine, Version 2.0



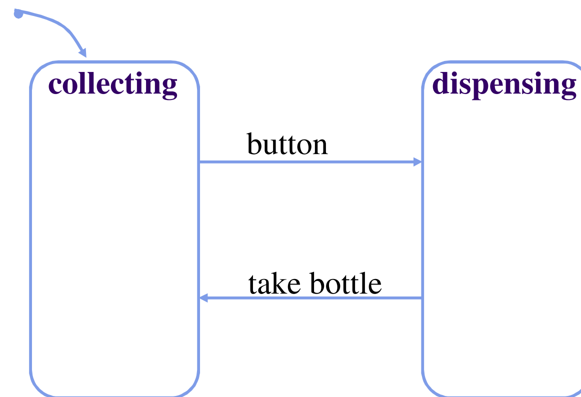
Coke Machine, Version 2.1

- Bottles sometimes shake loose
  - An additional, automatic indicator will indicate that the bottle is cleared
  - When the bottles are cleared, the machine will return to the same state it was in before the bottle got stuck
- State machine changes
  - How many new states are required?
  - How many new transitions?

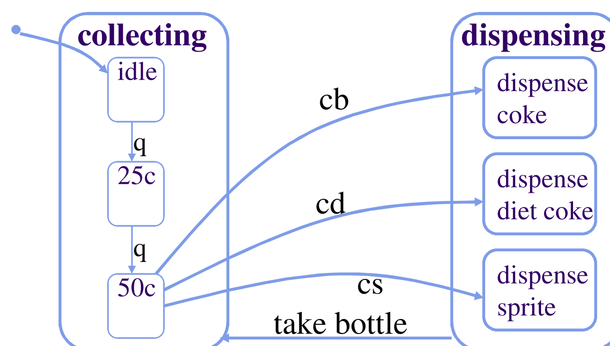
Coke Machine, Version 3.0

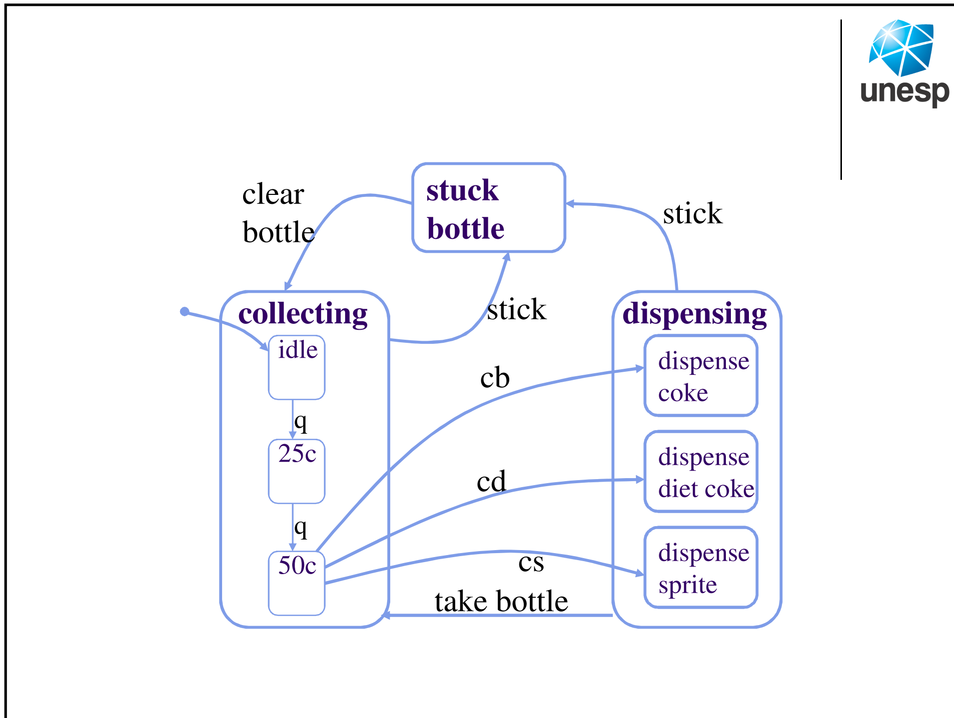
- Automatic bottle filler
  - If a button is pressed, the machine will toggle between bottle filling and dispensing modes
  - When in bottle filling mode:
    - Bottles may be inserted if the Coke machine is ready
    - When a bottle is inserted, the machine will NOT be ready to accept another bottle and will check the bottle
    - If the bottle check finds a Coke was inserted, it will signal Coke_OK and return to ready
    - If the bottle check finds a Diet Coke was inserted, the coke machine will signal Diet_OK and return to ready
    - Otherwise, the bottle will be immediately dispensed
- State machine changes
  - How many new states are required?
  - How many new transitions?

Bottle Dispenser

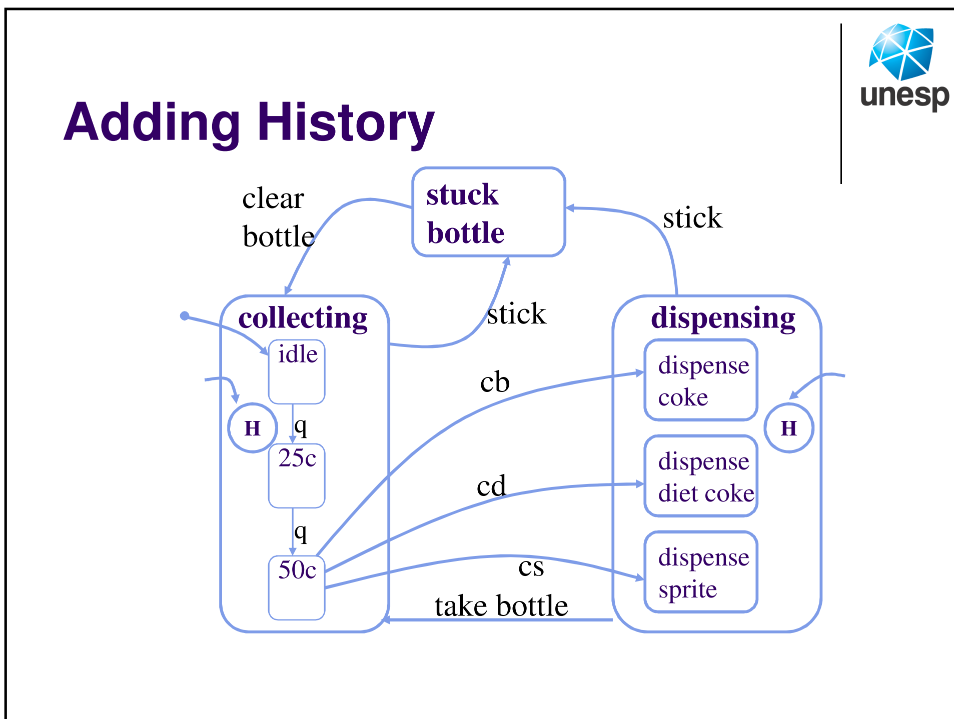


Bottle Dispenser

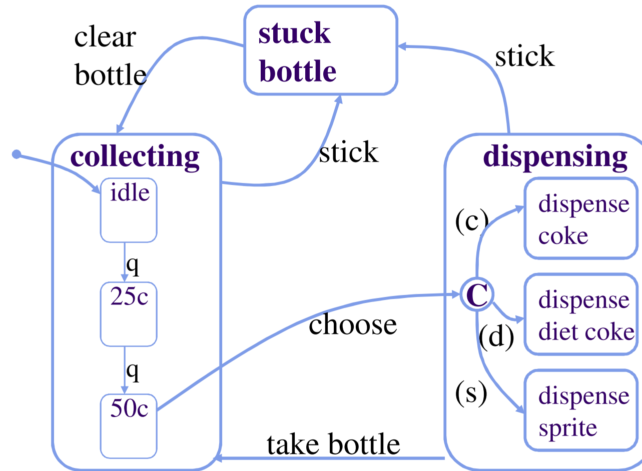




Adding History

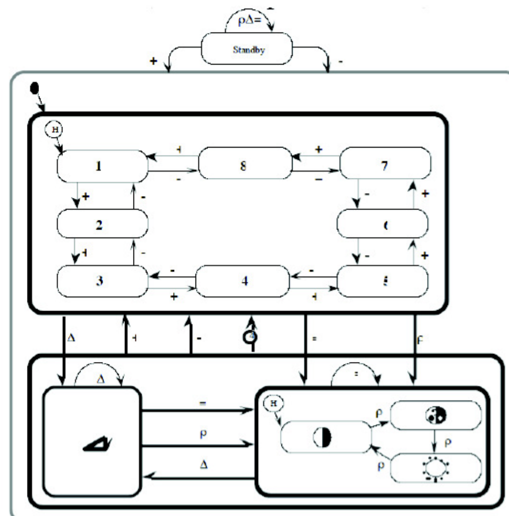


Adding Conditionals



StateChart Example – TV Controller

- 8 channels
- Buttons:
  - Standby
  - Channel: +/–
  - Volume: Δ+, Δ–
  - Contrast/Color/
  - Brightness: ρ
  - Pict. Adjust: ≡+, ≡–
- Missing details for:
  - volume /
  - brightness / color /
  - contrast selection



Note. ≡ means either of the ≡+ or ≡– buttons;
Δ means either of the Δ+ or Δ– buttons

Example – TV Remote

- Independent states
  - Timer
  - Channel – split into two states
  - Sound
  - Clock