

# CHSM

A language system for extending C++ or Java  
for implementing reactive systems.

Paul J. Lucas  
pauljlucas@mac.com

Fabio Riccardi  
fabio.riccardi@mac.com

These slides are available at:  
<http://homepage.mac.com/pauljlucas/software/chsm/chsm-slides.pdf>

# CHSM

## Overview:

- Motivation.
- State-Transition Diagrams vs. Statecharts.
- Statechart Features.
- Statechart Problems / Solution.
- CHSM Language Tour.

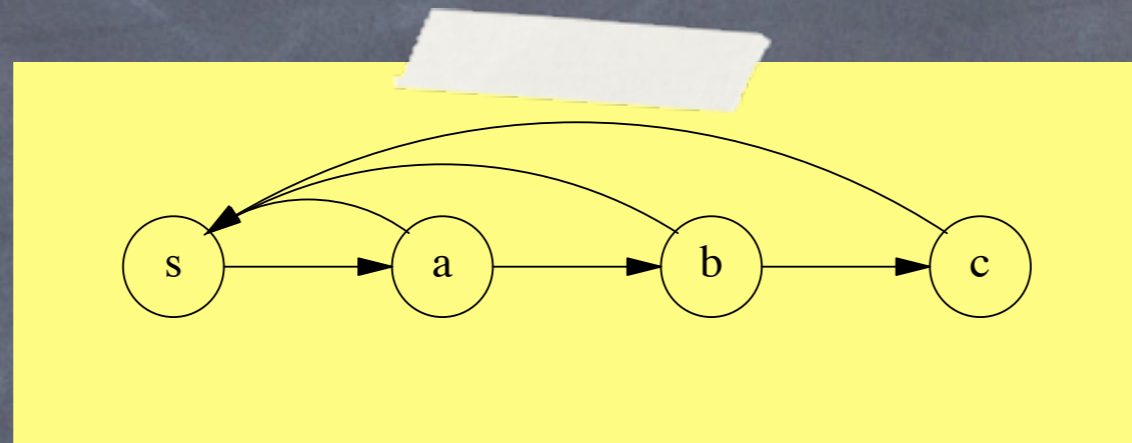
# CHSM

## Motivation:

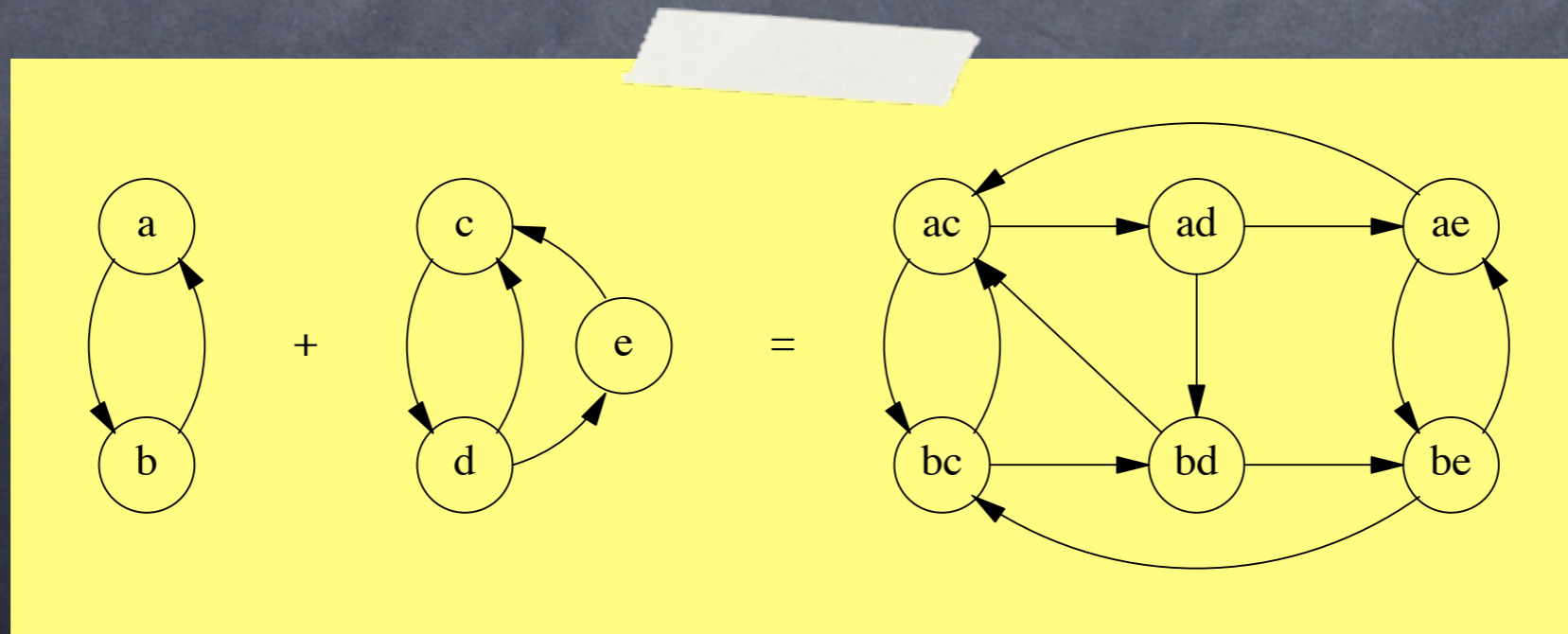
- Finite state machines (FSMs) are used in many computing and electronic applications including:
  - Communications protocols
  - Graphical user interfaces
  - Handheld devices (watches, PDAs, phones)
- FSMs have problems when used in practice.
- Statecharts, and their associated concurrent, hierarchical, finite state machines (CHSMs) address these problems.

## State-Transition Diagrams:

- Are "flat" and "global."
- Tend to have replicated transitions:

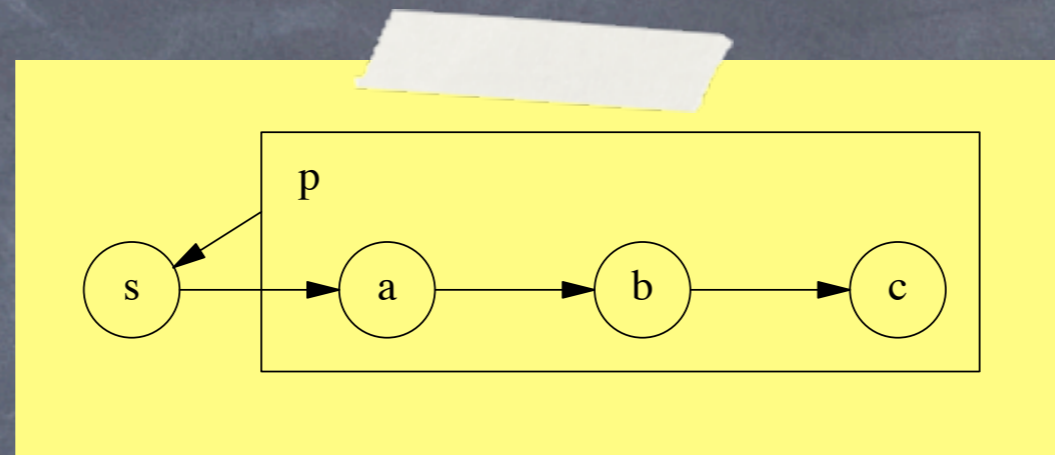


- Experience exponential growth:

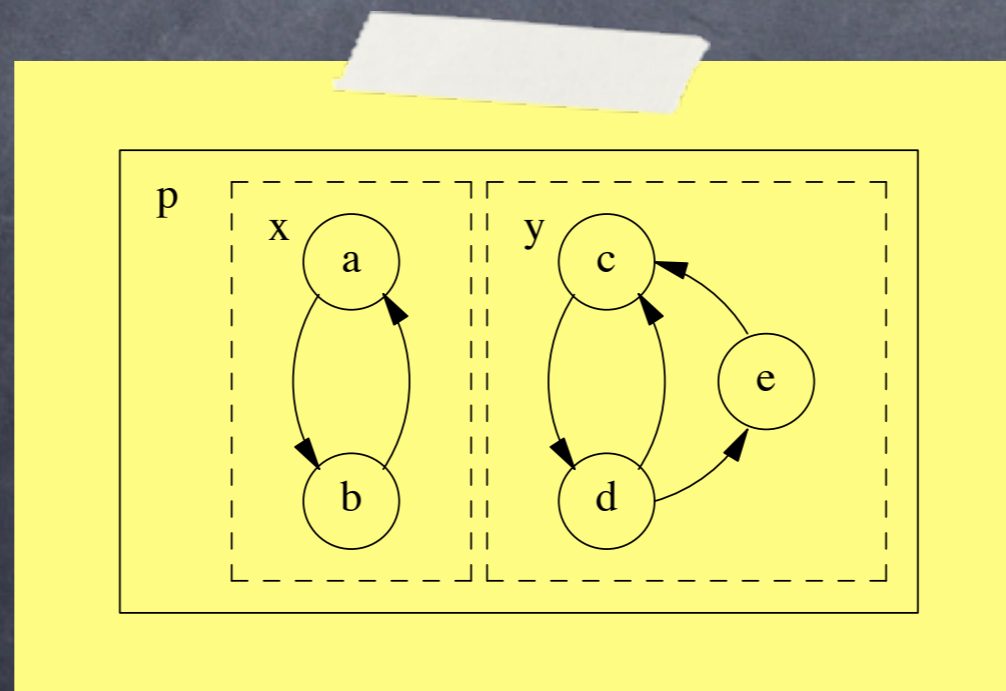


## Statecharts 1: Immediate benefits

- Are hierarchical.
- Eliminate replicated transitions:

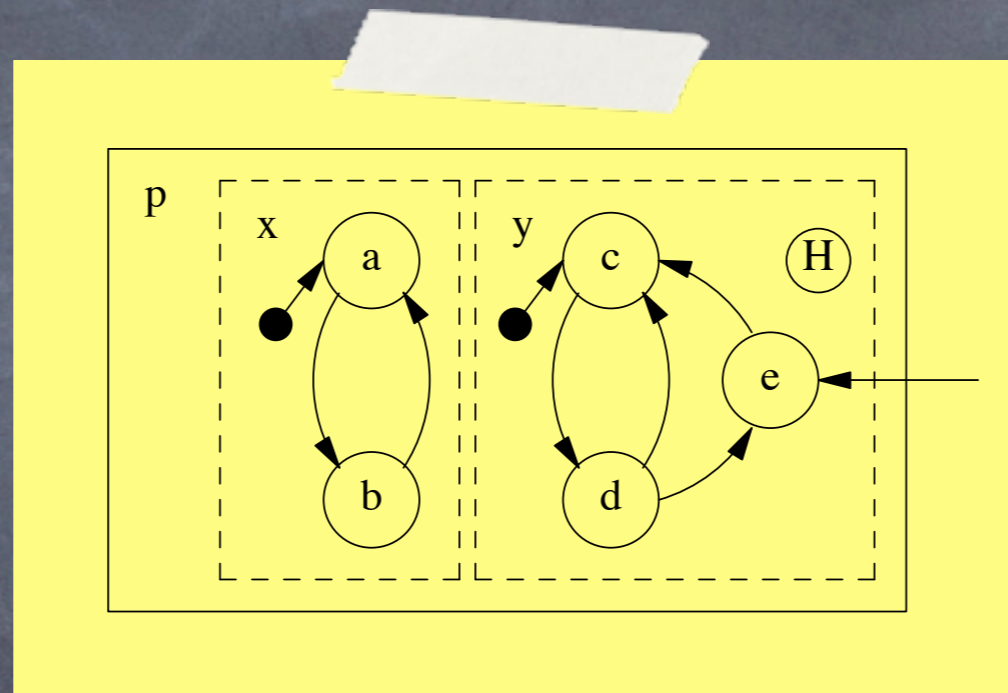


- Eliminate exponential growth:



## Statecharts 2: Default states & history

- Clusters have one default state.
- May be overridden by a history.



- History enters last active child state.
- Clusters may have a deep history.

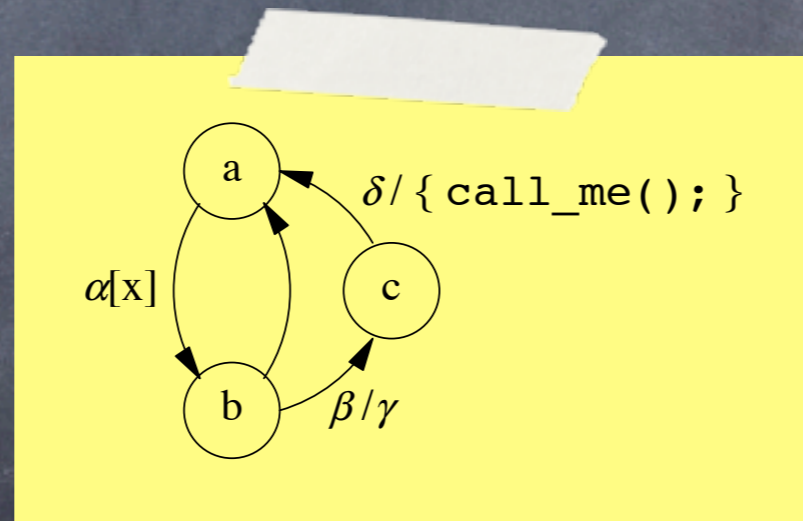
Can be overridden

# CHSM

## Statecharts 3: Conditions, broadcasting, & actions

- Conditions (boolean expressions) can be tied to transitions.
- Broadcasting = send event upon transition.
- Actions performed upon transition.

$\alpha$  has condition  $x$



$\delta$  calls  $\text{call\_me}()$

$\beta$  broadcasts  $\gamma$

## Statecharts: Problems

- Graphical notation requires sophisticated/expensive tools:
  - STATEMATE:
    - Doesn't integrate into traditional environments (compilers/makefiles).
    - Introduces "distance" between engineer and generated code.
    - Costs \$50K.

## Statecharts: Solution

- A textual language system for statecharts solves the problems:
  - Integrates easily into traditional development.
  - Generates straight-forward code.
  - Free (for non-commercial use).

# CHSM

## Language Tour 1: Introduction

- Uses a host language (C++ or Java).
- Adds additional constructs (like lex and yacc).

declaration	<pre>#include &lt;iostream&gt; using namespace std; %%</pre>
description	<pre>chsm hello is {     state say_it {         upon enter %{             cout &lt;&lt; "hello, world!\n";         }     } } %%</pre>
user code	<pre>main() {     hello world;     world.enter(); }</pre>

# CHSM

## Language Tour 2: Event Parameters

- Mechanism to transmit additional data.
- Specified exactly like function formal arguments.

```
int credit = 0;
%%
chsm vending_machine is {
    event coin( int value );

    state idle {
        coin -> collect;
    }
    state collect {
        upon enter %{
            credit += coin->value;
        }
        // ...
    }
}
```

declare parameter

access via -> operator

## Language Tour 3: Derived Events

- Allow event hierarchies to be modeled:

```

event core;
event<core> user_input;
  event<user_input> mouse( int x, int y );
    event<mouse> mouse_down;
    event<mouse> mouse_up;
  event<user_input> keystroke( unsigned modifiers );
    event<keystroke> key_down( char key );
    event<keystroke> key_up( char key );
  // ...

```

- Broadcast all base events.
- Inherit base-event parameters:

```

ui.key_down( modifiers, key );

```

"modifiers" inherited from keystroke

key\_down also broadcasts:  
keystroke, user\_input, & core

## Language Tour 4: Event Preconditions

- Mechanism to determine event suitability.
- Can be specified as a simple C++/Java expression:

```
event mouse( int x, int y ) [ x >= 0 && y >= 0 ];
```

- Or as a C++/Java function:

```
event login( int pin ) %{  
    if ( pin == atm_card.pin )  
        return true;  
    display( "INCORRECT PIN" );  
    return false;  
%}
```

# CHSM

## Language Tour 5: Derived CHSMs

- Allow data and functions to be added to a CHSM:

```
class vm_data : public CHSM::machine {  
public:  
    struct item {  
        char const *name;  
        int price;  
    };  
    vm_data( CHSM_MACHINE_ARGS, item const *items ) :  
        CHSM::machine( CHSM_MACHINE_INIT, item_( items ) ) {}  
protected:  
    item const *item_;  
    int num_items_;  
    int credit_;  
};  
  
%% CHSM uses <> syntax to specify derived CHSM  
chsm<vm_data> vending_machine( item const *items ) is {  
    // ...  
}
```

## Language Tour 6: Vending Machine Example 1

- Event specification with preconditions:

```

chsm<vm_data> vending_machine( item const *items ) is {
  event coin( int value ) %{
    switch ( value )
      case 1: case 5: case 10: case 25: return true;
    cerr << "No foreign coins!\n";
    return false;
  }%
  event select( int id ) %{
    if ( id < 0 || id >= num_items_ ) {
      cerr << "Invalid selection\n";
      return false;
    }
    if ( credit_ >= item_[ id ].price )
      return true;
    cerr << "Please deposit another "
      << item_[ id ].price - credit_ << " cents\n";
    return false;
  }%

  // ... continues ...

```

"coin" event precondition

"select" event precondition

## Language Tour 6: Vending Machine Example 2

- "idle" state specification:

```
state idle {
    upon enter %{
        cout << "\nVending Macine:\n";
        num_items_ = 0;
        for ( item const *i = &item_[0]; i->name; ++i )
            cout << char('A' + num_items_++) << ". "
                << i->name << " (" << i->price << ")\n";
        credit_ = 0;
    }

    coin -> collect;
}

// ... continues ...
```

## Language Tour 6: Vending Machine Example 3

- "collect" state specification:

```
state collect {
  upon enter %{
    credit_ += coin->value;
    cout << "Credit: " << credit_ << " cents\n";
  }

  coin -> collect;

  select -> idle %{
    cout << "Enjoy your " << item_[ select->id ].name << "\n";
    int change = credit_ - item_[ select->id ].price;
    if ( change > 0 )
      cout << "Change: " << change << " cents\n";
  };
}

// ... continues ...
```

## Language Tour 6: Vending Machine Example 4

- Interacting with a machine in a program:

```

%%
vending_machine::item yummy_treats[] = {
    { "Cheese Puffs", 65 },
    { "Potato Chips", 80 },
    { "Pretzels", 60 },
    0
};
main() {
    vending_machine vm( yummy_treats );
    vm.enter();
    while ( !cin.eof() ) {
        cout << "Enter coins or selection: ";
        char buf[10];
        cin.getline( buf, sizeof buf );
        if ( isdigit( *buf ) )
            vm.coin( atoi( buf ) );
        else if ( isalpha( *buf ) )
            vm.select( toupper( *buf ) - 'A' );
        else
            cerr << "Invalid input\n";
    }
}

```

This is a "toy" UI to show how to interact with a machine.

## Language Tour 7: Derived States 1

- Allow data and behavior to be added to a state.
- Example: Petri Nets:

derived from state class

```
class token : public CHSM::state {
public:
    token( CHSM_STATE_ARGS ) :
        CHSM::state( CHSM_STATE_INIT ), tokens_( -1 ) {}
    bool enter( CHSM::event const &event, CHSM::state *from ) {
        if ( !CHSM::state::enter( event, from ) )
            return false;
        ++tokens_;
        return true;
    }
    int operator--=( int used ) { return tokens_ -= used + 1; }
    operator int() const { return tokens_; }
private:
    int tokens_;
};
```

Override "enter"  
method to  
augment behavior

## Language Tour 7: Derived States 2

- Petri Nets Example: Making water molecules:

```

int water_molecules;
%%
chsm make_water is {
  set H2O( H2, O2 ) {
    hydrogen[ ${H2} >= 1 && ${O2} >= 1 ] -> H2O %{
      ${H2} -= 1, ${O2} -= 1; water_molecules += 2;
    };
    oxygen[ ${H2} >= 2 ] -> H2O %{
      ${H2} -= 2; water_molecules += 2;
    };
  } is {
    state<token> H2 { hydrogen -> H2; }
    state<token> O2 { oxygen -> O2; }
  }
}

```

$\${...}$  notation (borrowed from Unix shells) accesses state objects

uses  $\langle \rangle$  syntax to specify derived state

# CHSM

## Real-World Example:

- Used at European Laboratory for Particle Physics (CERN) in control systems for experiments.
- Hardware: 40 embedded VME CPUs controlled by a cluster of 10 Unix workstations.
- Software: 40K lines of C++ for data acquisition.
- Statechart: 48 states, 114 transitions, 79 events, 34 conditions, 88 actions. Other, smaller statecharts in subsystems running concurrently.
- Staff: 100+ people; core system produced by 2.

# CHSM

## License & Downloading:

- Free software under the GPL.
- Alternate commercial licenses available.
- Downloading:

<http://homepage.mac.com/pauljluкас/software/chsm/>