

UMDES-LIB:  
LIBRARY OF COMMANDS FOR  
DISCRETE EVENT SYSTEMS  
MODELED BY FINITE STATE MACHINES

Developed by the DES Group at the University of Michigan  
under the coordination of  
*Stéphane Lafortune* and *Demosthenis Teneketzis*

WODES 2000 – August 22, 2000

## Features

- Approximately 27,000 lines of C code
- Commands for
  - manipulating FSMs ( $\approx 30$  commands)
  - failure diagnosis ( $\approx 10$  commands)
  - supervisory control ( $\approx 10$  commands)
- No GUI at present...
  - interactive I/O
  - command line

## Manipulation of FSMs

- Unary operations:  
`acc`, `co_acc`, `trim`, `live`, `comp_fsm`, `minimize_std`  
`fsm_query`
- Composition operations:  
`product`, `par_comp`, `concat`, `union`
- Comparison operations:  
`equiv`, `incl`, `conflict`
- Other operations:  
`obsvr`, `refine`, `inv_proj`, `inv_p_L`

## Failure diagnosis

- Model-building:  
`write_st`, `sensmap`, `compose`, `write_o`, `eventmap`
- Construction of diagnosers:  
`diag`, `dmf`, `idiag`, `dmulti`
- Verification of diagnosability (*indeterminate cycles*):  
`dcycle`, `dmfcycle`, `dicycle`

## Supervisory control

- Verification of properties:  
**ctrb, obs, coobs, normal**
- Synthesis operations:
 

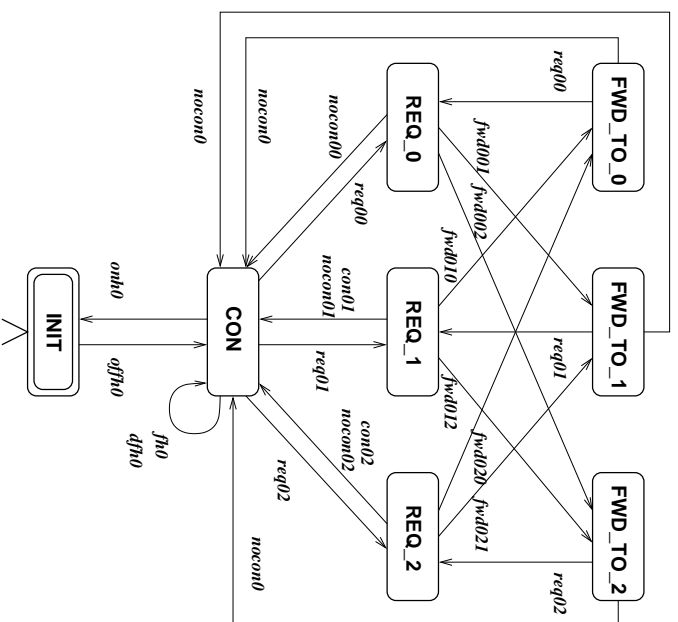
<b>supcon_std</b>	supremal controllable sublanguage
<b>infcon</b>	infimal closed controllable superlanguage
<b>supnorm</b>	supremal normal sublanguage (prefix-closed case)
<b>supcn1</b>	supremal controllable normal sublanguage (prefix-closed case)
<b>vlp_s</b>	supremal controllable sublanguage (different algorithm)

## Representation of FSMs

- Stored as linked lists (“doubly-linked” and nested)
- States and events: character strings
- $\times$  ,  $||$  : component state names are preserved
- Observers and diagnosers: state estimate information is preserved

# Some Examples

- “.fsm” file for an eight-state FSM:



Destination state	CON	INIT	CON	uc	o
CON	0	6	1	1	0
INIT	0	0	1	1	0
CON	0	0	1	1	0
REQ_0	0	0	1	1	0
REQ_1	0	0	1	1	0
REQ_2	0	0	1	1	0

Number of states: 8  
 Marking: INIT, CON  
 Number of out-transitions: 1 (for INIT, CON), 6 (for CON)  
 Observability property: uc, o  
 Controllability property: c, o

- Parallel composition of six FSMs:

```
[ 80 ] ex --: par_comp
```

```
Enter the number of components for parallel composition : 6
```

```
[ 1] Enter the FSM data file: controller.fsm
```

```
[ 2] Enter the FSM data file: valve.fsm
```

```
[ 3] Enter the FSM data file: pump.fsm
```

```
[ 4] Enter the FSM data file: fan.fsm
```

```
[ 5] Enter the FSM data file: boiler.fsm
```

```
[ 6] Enter the FSM data file: load.fsm
```

```
Enter the output file : sync.fsm
```

```
Processing files . . . .
```

```
The output machine has been written to sync.fsm
```

```
END OF PROGRAM : par_comp
```

```
[ 81 ] ex --:
```

- Format of `sync.fsm`:

90

```
C1,V1,P1,F1,B1,L0 0 5
fon C2,V1,P1,F2,B1,L0 c o
cfoff C21,V1,P1,F1,B1,L0 c no
cfon C11,V1,P1,F1,B1,L0 c no
sol C1,V3,P1,F1,B1,L0 c no
sc1 C1,V4,P1,F1,B1,L0 c no
```

```
C2,V1,P1,F2,B1,L0 0 4
sol C2,V3,P1,F2,B1,L0 c no
sc1 C2,V4,P1,F2,B1,L0 c no
spi C3,V1,P1,F2,B1,L2 c o
spd C10,V1,P1,F2,B1,L1 c o
```

...

- Computation of supremal controllable sublanguage:

```
[ 120 ] ex-:  supcon_std
*****
This routine requires the fsm event properties to be
included in the fsm files.
If not, run 'change_cprop' routine first
*****
Enter the FSM H data file
(for supremal controllable algorithm): H_OCS
Enter the reference FSM G data file: G_0
Enter the output file : Sup_H_OCS
Print which states were deleted and why? (Yes/No) n
Processing ....
The output machine has been written to Sup_H_OCS
END OF PROGRAM : supcon_std
```

- Construction of an observer:

```
[ 122 ] ex-: obsvr
```

```
Enter the FSM data file : sync.fsm
```

```
Enter the output file for *.o format : sync-obs.o
```

```
Enter the output file for *.fsm format (This is optional).
```

```
Enter 0 if you do not need this output : 0
```

```
END OF PROGRAM : obsvr
```

```
[ 123 ] ex-:
```

- Format of an observer FSM, sync-obs.o:

Total Observer States = 31

Id = 1

C1, V1, P1, F1, B1, L0

C21, V1, P1, F1, B1, L0

C11, V1, P1, F1, B1, L0

C1, V3, P1, F1, B1, L0

C1, V4, P1, F1, B1, L0

C21, V3, P1, F1, B1, L0

C21, V4, P1, F1, B1, L0

C11, V3, P1, F1, B1, L0

C11, V4, P1, F1, B1, L0

Total pairs = 9

fon -> 2

...

## On the Web

<http://www.umich.edu/umdes>

- List of commands, with brief description (links to more detailed descriptions)
- Executables (UNIX/LINUX) for download
- PDF of this talk plus some “demos”

## Acknowledgements

Students who have participated in the development of UMDES-LIB so far:

George Barrett	Nejib Ben Hadj-Alouane	John Chay
Yi-Liang Chen	Rami Debouk	Shailesh Humbad
Meera Sampath	Jeremy Shapiro	Irwan Siregar
Kin-Yip Sit	David Watson	Tae-Sic Yoo

## Failure Diagnosis Example

### Heating Ventilation & Air Conditioning System

In this section<sup>1</sup>, we illustrate the failure diagnosis routines of the UMDES library. For that purpose we consider a heating, ventilation, and air conditioning (HVAC) system. Figure 1 illustrates part of a complete HVAC system. It depicts a single air handling system with its associated primary conditioning system. The system is composed of the following components: a pump, a valve, a fan, a boiler, and a controller. Also of consideration is a model of the load that triggers the operation of the system. The models of these components are shown in Figure 2.

Further explanations about this example can be found in:

- M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Failure diagnosis using discrete event models," *IEEE Transactions on Control Systems Technology*, vol. 4, no. 2, pp. 105–124, March 1996.

---

<sup>1</sup>This section was written by Rami Debouk.

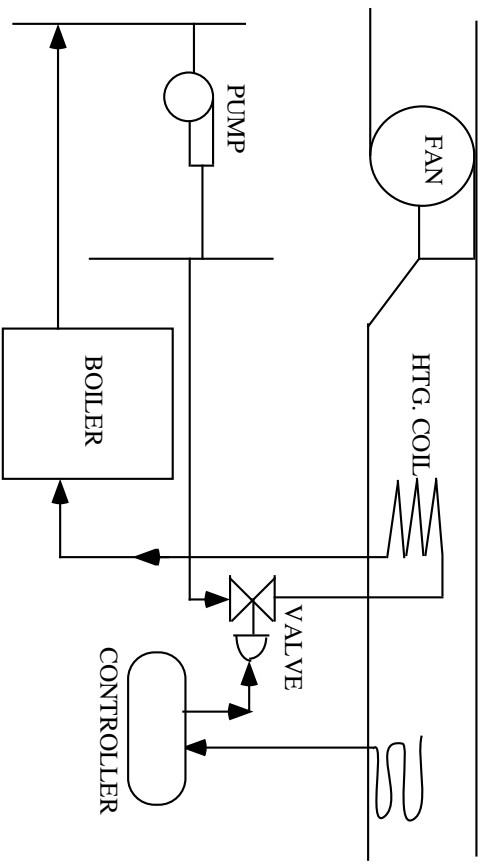


Figure 1: Part of an HVAC system

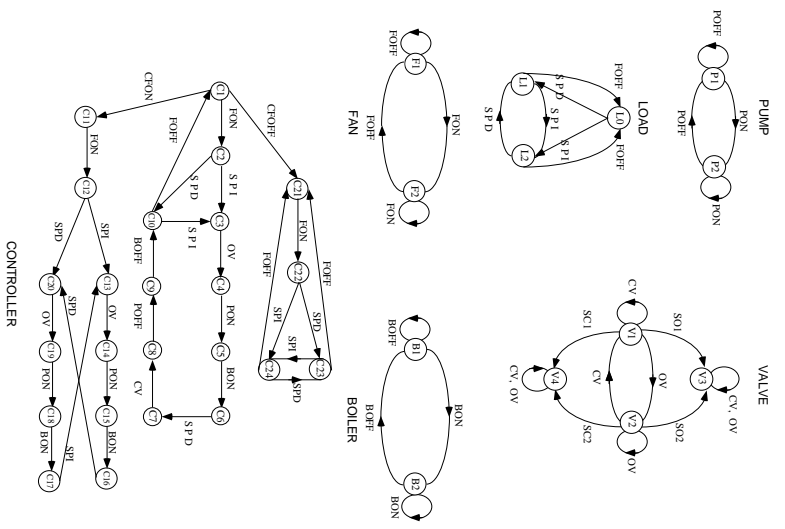


Figure 2: Components models of the HVAC system

Every finite state machine component is created interactively. As an example let us create that of the pump. The routine to be used is `create_fsm`.

```
[ 13 ] ex -: create_fsm
```

Please enter the name of the fsm you wish to create -> pump\_fsm

Enter the number of states in the fsm -> 2

Enter the name of the first state -> P1

Is this state marked? (Yes/No) No

Enter the number of transitions from this state -> 2

Enter the name of the first transition -> poff

Enter the name of the resulting state -> P1

Do you want to input the controllability and

observability properties of all the transitions? (Yes/No)

If you choose not to input these properties, all

transitions will be set to controllable and observable

No

Enter the name of the second transition -> pon

Enter the name of the resulting state -> P2

Do you want to input the controllability and

observability properties of all the transitions? (Yes/No)

If you choose not to input these properties, all

transitions will be set to controllable and observable

No

Enter the name of the second state -> P2

Is this state marked? (Yes/No) No

Enter the number of transitions from this state -> 2

Enter the name of the first transition -> poff

Enter the name of the resulting state -> P1

Do you want to input the controllability and

observability properties of all the transitions? (Yes/No)

If you choose not to input these properties, all transitions will be set to controllable and observable

No

Enter the name of the second transition -> pon

Enter the name of the resulting state -> P2

Do you want to input the controllability and observability properties of all the transitions? (Yes/No)

If you choose not to input these properties, all

transitions will be set to controllable and observable

No

The file 'pump.fsm' has been written.

End of program: create\_fsm

[ 14 ] ex -: more pump.fsm

2

P1 0 2

poff P1 c o

pon P2 c o

P2 0 2

poff P1 c o

pon P2 c o

[ 15 ] ex -:

The failures we are interested in diagnosing are the following:

- F1 = {SO1,SO2},
- F2 = {SC1,SC2},
- F3 = {CFON}, and
- F4 = {CFOFF}.

For the considered HVAC system we only have one sensor that measures whether there is flow through the pump or not. The corresponding sensor map is shown below:

- (P1,●,●,●,●)  $\rightarrow$  NF,
- (P2,V1,●,●,●)  $\rightarrow$  NF,
- (P2,V2,●,●,●)  $\rightarrow$  F,
- (P2,V3,●,●,●)  $\rightarrow$  F, and
- (P2,V4,●,●,●)  $\rightarrow$  NF.

where a ● stands for a “don't care” sign of the state of the corresponding component.

The procedure to diagnose the system includes the following steps:

1. Compose the component models using `par_comp`. This synchronizes the operation of all the components.
2. Generate the sensor readings corresponding to every reachable state in the composed model. This is achieved by first applying `write_st` to the composed model to generate all reachable states, and then using `sensmap` that integrates the sensor map presented above into the reachable states to generate the state/sensor map.
3. Integrate the state/sensor map into the composed model in two steps: first generate the unobservable events of the composed model by applying `write_uo`, and then composing the composed model with the state/sensor map by using `compose`. We refer to the last composition as the hvac system.
4. Generate the diagnoser by applying `dmf` to the hvac system along with the failure partition discussed above.
5. Finally, check whether there exist indeterminate cycles by using `dmf_cycle`.

The following is the interactive input/output for applying the 5 steps.

```
[ 80 ] ex -: par_comp
Enter the number of components for parallel composition : 6
[ 1 ] Enter the FSM data file: controller.fsm
[ 2 ] Enter the FSM data file: valve.fsm
[ 3 ] Enter the FSM data file: pump.fsm
[ 4 ] Enter the FSM data file: fan.fsm
[ 5 ] Enter the FSM data file: boiler.fsm
[ 6 ] Enter the FSM data file: load.fsm
Enter the output file : sync.fsm
Processing files ....
The output machine has been written to sync.fsm
END OF PROGRAM : par_comp
[ 81 ] ex -: write_st
Enter the FSM data file : sync.fsm
Enter the output file : sync.states
Processing ....
END OF PROGRAM : write_st
[ 82 ] ex -: sensmap
Enter the states data file : sync.states
Enter the mapping pattern file : glob_sens.map
Enter the output file : sensor_data.map
Processing ....
END OF PROGRAM : sensmap
[ 83 ] ex -: write_no
Enter the FSM data file : sync.fsm
Enter the output file : sync.no
Processing ....
END OF PROGRAM : write_no
```

```
[ 84 ] ex -: compose
Enter the FSM data file : sync.fsm
Enter the unobservable events file : sync.no
Enter the sensor data file : sensor-data.map
Enter the output file : hvac.fsm
Processing ....
The composite system has been written to hvac.fsm
END OF PROGRAM : compose
[ 85 ] ex -: dmf
Enter the FSM data file : hvac.fsm
Enter the failure partition file : hvac.ft
Enter the output file for *.d format: hvac.d
Enter the output file for *.fsm format (This is optional).
Enter 0 if you do not need this output : hvac.d.fsm
Would you like the state names in the diagnoser.fsm output to be
(1) Numerical sequence (i.e. 1, 2,...)
(2) Concatenation of the fsm states
Choose '1' or '2' -> 2
Processing ....
END OF PROGRAM : dmf
[ 86 ] ex -: dmfcycle
Enter the FSM data file : hvac.fsm
Enter the failure partition file : hvac.ft
Enter the output file : hvac.c
Processing ....
END OF PROGRAM : dmfcycle
[ 87 ] ex -:
```

By checking the file `hvac.c`, we realize that there are indeterminate cycles. Let us check whether the system is I-diagnosable. First let us determine indicator events as follows:

- $I(F1) = \{CV\}$ ,
- $I(F2) = \{OV\}$ ,
- $I(F3) = \{CV\}$ , and
- $I(F4) = \{OV\}$ ,

In order to build the I-diagnoser we follow the following 4 steps

1. Generate the observable events in the `hvac` model. This is done using `write_o`.
2. Next we generate the indicator map by associating indicator events to the states reached following observable events using `eventmap`.
3. We build the I-diagnoser using `idiag`.
4. We check for the existence of indeterminate cycles using `dicycle`.

The following is the interactive input/output for applying the 4 steps.

```
[ 102 ] ex -: write_o
Enter the FSM data file : hvac.fsm
Enter the output file : hvac.o
Processing ....
END OF PROGRAM : write_o
[ 103 ] ex -: eventmap
Enter the observable events data file : hvac.o
Enter the indicator events file : hvac.ievents
Enter the output file : hvac.ift
Processing ....
END OF PROGRAM : eventmap
[ 104 ] ex -: iddiag
Enter the FSM data file : hvac.fsm
Enter the failure partition file : hvac.ft
Enter the indicator mapping file : hvac.ift
Enter the output file for *.d format : hvac.i.d
Enter the output file for d in *.fsm format (Optional).
Enter 0 if you do not need this output : hvac.i.d.fsm
Would you like the state names in the i-diagnoser.fsm output to be
(1) Numerical sequence (i.e. 1, 2,...)
(2) Concatenation of the fsm states
Choose '1' or '2' -> 2
Processing ....
END OF PROGRAM : iddiag
[ 105 ] ex -: dicycle
Enter the FSM data file : hvac.fsm
Enter the failure partition file : hvac.ft
Enter the indicator mapping file : hvac.ift
```

```
Enter the output file : hvac.i.c
Processing .....
EWD OF PROGRAM : dicycle
[ 106 ] ex -: more hvac.i.c
dicycle: No indeterminate cycles found
[ 107 ] ex -:
```

A partial view of the I-diagnoser is shown in Figure 3.

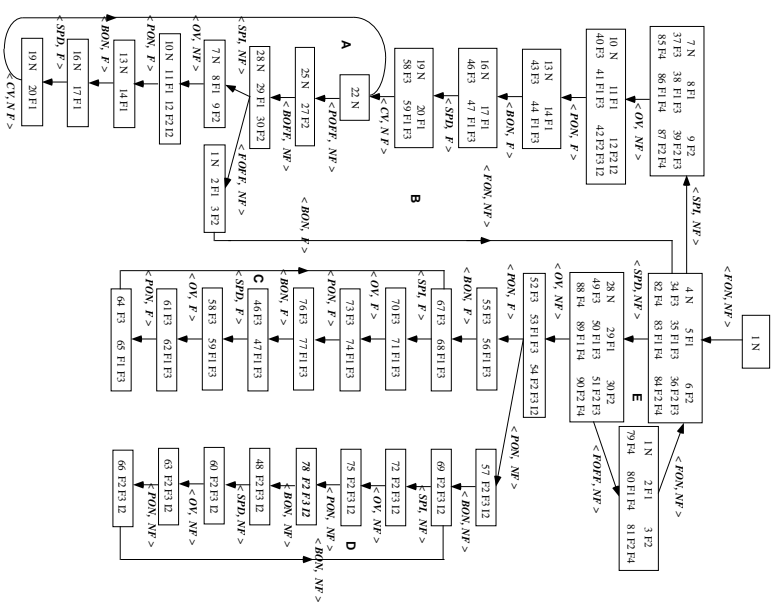


Figure 3: Part of the I-diagnoser for the HVAC system

## Supervisory Control Example

### Feature Interactions in Intelligent Networks

In this section<sup>2</sup>, we illustrate the supervisory control routines of the UMDES library using a simple model of a three-subscriber telephone system along with call forwarding and originating call screening specifications. For further explanations of this example, we refer the reader to the paper:

- Y.-L. Chen, S. Lafortune, and F. Lin, “Resolving feature interactions using modular supervisory control with priorities,” in *Feature Interactions in Telecommunication Networks IV*, P. Dini, R. Boutaba, and L. Logrippo, Editors, IOS Press, 1997, pp. 108–122.

---

<sup>2</sup>This section was written by Tae-Sic Yoo.

## Model of a Three-Subscriber Telephone System

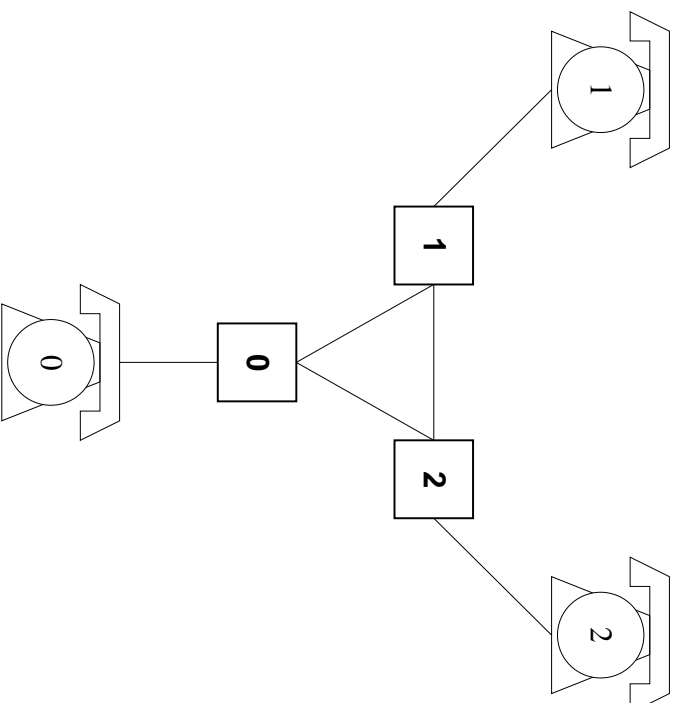


Figure 4: Three-Subscriber Telephone System



- UMDES library command for generating a FSM: `create_fsm`

The global model, denoted by  $G$ , consists of the parallel composition of the models of all users,  $G(0)$ ,  $G(1)$ , and  $G(2)$ . That is,

$$G := G(0) \parallel G(1) \parallel G(2).$$

The resulting machine  $G$  contains 512 states. For the implementation,  $G(i)$  is stored as  $G_i$  for each  $i$  and  $G$  is stored as  $G$ .

- UMDES library command for parallel composition: `par_comp`
- `par_comp`

```

Enter the number of components for parallel composition : 3
[ 1] Enter the FSM data file: G0
[ 2] Enter the FSM data file: G1
[ 3] Enter the FSM data file: G2
Enter the output file : G

Processing files .....
The output machine has been written to G
END OF PROGRAM : par_comp
```

## Local Model at site 0, $G_0$

The local model at site 0 is generated by projecting the global model  $G$  onto the set of locally observable events  $\Sigma_{o,0}$  at site 0. In order to perform this projection operation, we need to specify the locally unobservable events at site 0 among the events defined in the global model  $G$ . We refer the reader to the above-referenced paper by Chen *et al.* for this information.

As constructed above, all the events in the global model  $G$  are observable (globally). We can change the observability properties of events, to reflect the information in  $\Sigma_{uo}$ , by using the command `change-observ`; for simplicity, the resulting machine is still stored in  $G$ . We omit the detail of this operation since it is straightforward. The observability properties of events at site 0 are now included in  $G$  and the desired projection operation can be performed using the command `obsvr`.

$$G_0 := P_0(G).$$

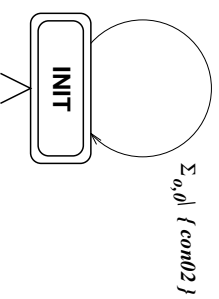
The resulting FSM for the local model, denoted by  $G_0$ , has 72 states. This machine is stored as `G_0`.

- UMDES library command for observer construction: `obsvr`.
- `obsvr`
  - Enter the FSM data file : `G`
  - Enter the output file for \*.o format : `G_0.o`
  - Enter the output file for \*.fsm format (This is optional).
  - Enter 0 if you do not need this output : `G_0`
  - Would you like the state names in the observer.fsm output to be
    - (1) Numerical sequence (i.e. 1, 2,....)
    - (2) Concatenation of the fsm states
  - Choose '1' or '2' → `1`
- END OF PROGRAM : `obsvr`
- `G_0` is written in FSM file style while `G_0.o` is a different file style.

After constructing the local model  $G_0$  at site 0, we need to specify which events are uncontrollable at that site (this information can be found in the above-referenced paper by Chen *et al.*). This can be done using the command `change_cprop`, which is analogous to `change_oprop`. The resulting machine is still stored as `G_0`.

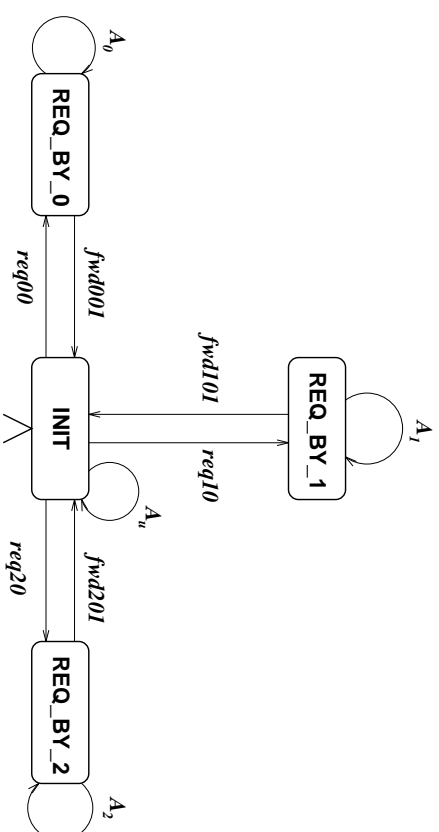
## Specifications

We consider two specifications: Call Forwarding (CF) and Originating Call Screen (OCS). These models are stored as CF and OCS respectively.



$$\Sigma_{OCS|2} = \Sigma_{a,\delta}$$

Figure 6: OCS



$$A_u = \Sigma_0 \setminus \{ req00, req10, req20 \} \setminus \{ fwd001, fwd002, fwd101, fwd102, fwd201, fwd202 \}$$

$$A_0 = \Sigma_0 \setminus \{ nocon00, fwd001, fwd002 \}$$

$$A_l = \Sigma_0 \setminus \{ con10, nocon10, fwd101, fwd102 \}$$

$$A_2 = \Sigma_0 \setminus \{ con20, nocon20, fwd201, fwd202 \}$$

Figure 7: CF

## Legal Languages

Legal languages are generated by composing  $G_0$  with each specification. That is,

$$\mathcal{L}(\text{H\_CF}) := \mathcal{L}(\text{CF} \parallel_{G_0}), \quad \mathcal{L}(\text{H\_OCS}) := \mathcal{L}(\text{OCS} \times G_0).$$

H\_CF and H\_OCS have 128 and 72 states respectively.

- UMDES library commands: **product**, **par\_comp**
- **par\_comp**
  - Enter the number of components for parallel composition : 2
  - [ 1] Enter the FSM data file: CF
  - [ 2] Enter the FSM data file: G\_0
  - Enter the output file : H\_CF
- Processing files . . . .
- The output machine has been written to H\_CF
- END OF PROGRAM : par\_comp
- **product**
  - Enter the first FSM data file : OCS
  - Enter the second FSM data file : G\_0
  - Enter the output file : H\_OCS
- Processing . . . .
- The product machine has been written to H\_OCS
- END OF PROGRAM : product

## Test of Controllability

- UMDES library command: `ctrb`
- `ctrb`  
Enter the H FSM data file for controllability testing: `H_CF`  
Enter the reference G FSM data file : `G_0`  
  
Enter the uncontrollable events file.  
This input is optional. If given, the events  
will override the uncontrollable events in `g.fsm`.  
Enter '0' to use the uncontrollable events in `G → 0`  
Print which states and transitions violate  
the controllability property (active event set constraint)? (Yes/No) `n`  
  
Processing ....  
  
`L(H)` is CONTROLLABLE with respect to `L(G)`  
and the set of controllable events.  
  
END OF PROGRAM : `ctrb`

## • ctrb

Enter the H FSM data file for controllability testing: H\_OCS  
Enter the reference G FSM data file : G\_0

Enter the uncontrollable events file.

This input is optional. If given, the events  
will override the uncontrollable events in g.fsm.

Enter '0' to use the uncontrollable events in G → 0

Print which states and transitions violate  
the controllability property (active event set constraint)? (Yes/No) n

Processing ....

L(H) is NOT CONTROLLABLE with respect to L(G)  
and the set of controllable events.

END OF PROGRAM : ctrb

## Synthesis of Supremal Controllable Sublanguage

Since OCS is not controllable, we construct the supremal controllable sublanguage corresponding to OCS.

- UMDES library command: `sup_con_std`
- `supcon_std`

```
*****
```

```
This routine requires the fsm event properties to be
included in the fsm files.
```

```
If not, run 'change_cprop' routine first
```

```
*****
```

```
Enter the FSM H data file
```

```
(for supremal controllable algorithm): H_OCS
```

```
Enter the reference FSM G data file: G_0
```

```
Enter the output file : Sup_H_OCS
```

```
Print which states were deleted and why? (Yes/No) n
```

```
Processing ....
```

```
The output machine has been written to Sup_H_OCS
```

```
END OF PROGRAM : supcon_std
```

Originally H\_OCS had 72 states. Now, Sup\_H\_OCS 63 has states.