Statecharts aka Harel Charts

Visual Modelling

1. Higraph formalism
2. Statechart formalism (combines Higraphs and State Automata)

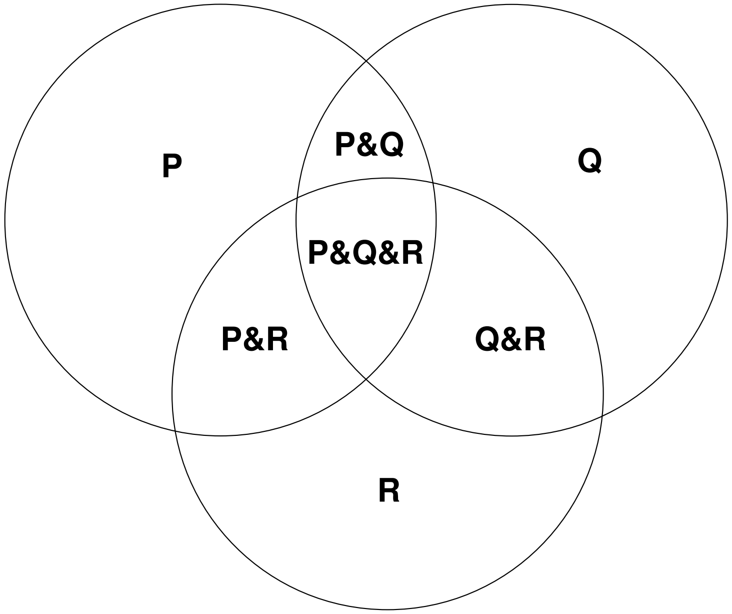
Diverse applications.

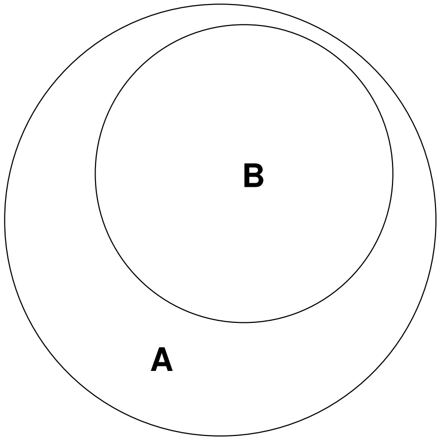
In particular: concurrent systems behaviour

Higraphs: Visualising Information

- complex
- non-quantitative, structural
- topological, not geometrical
- Euler
  - Venn diagrams (Jordan curve: inside/outside): enclosure, intersection
  - graphs (nodes, edges: binary *relation*); hypergraphs

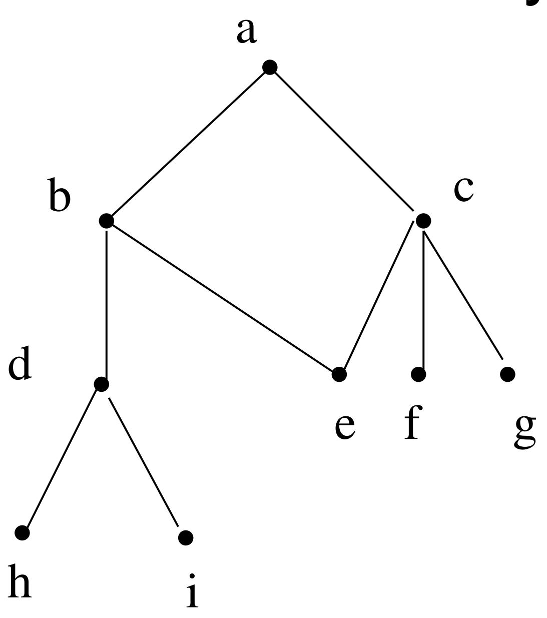
Venn diagrams, Euler circles



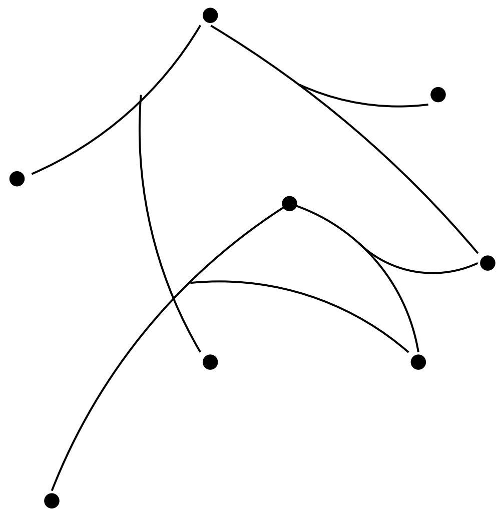


- *topological* notions (syntax):
  enclosure, exclusion, intersection
- Used to represent (denote) *mathematical* set operations:
  union, difference, intersection

Hypergraphs



a graph



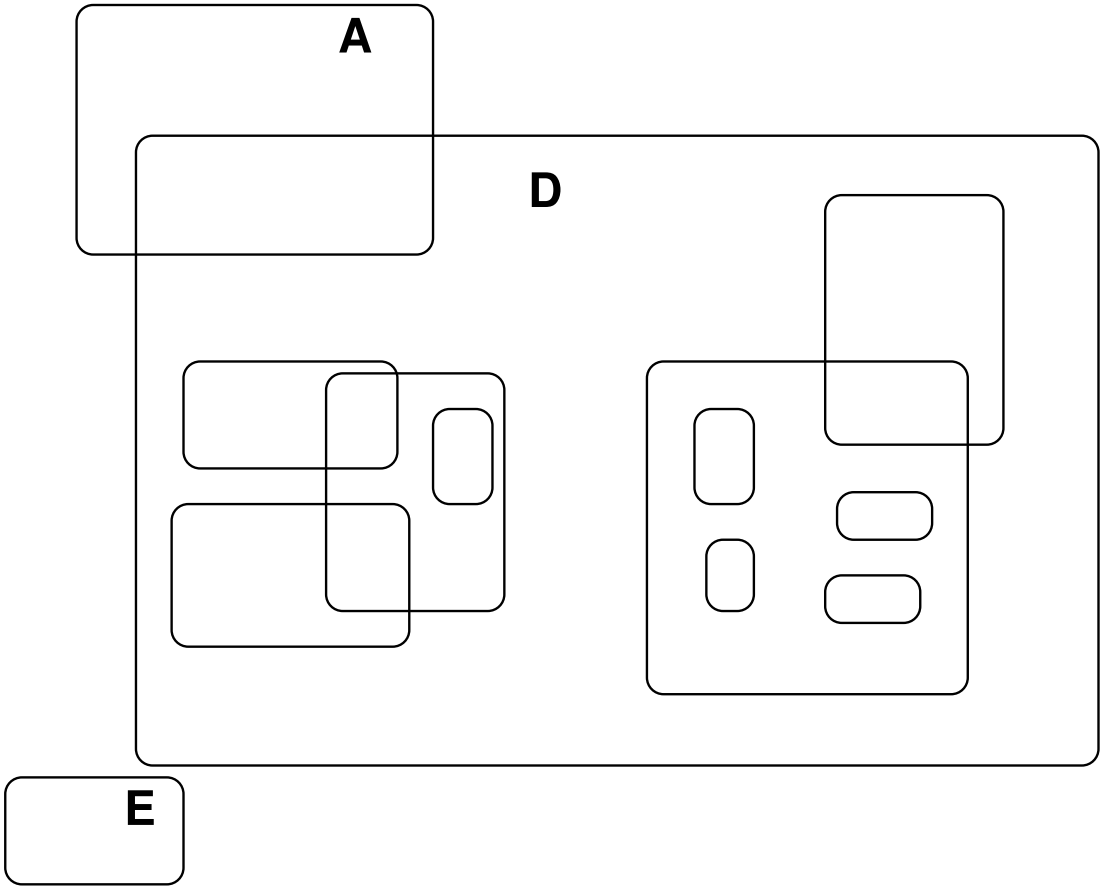
a hypergraph

- *topological* notion (syntax): connectedness
- Used to represent (denote) *relations* between sets.
- Hyperedges: non longer binary relation ($\subseteq X \times X$):
  $\subseteq 2^X$ (undirected), $\subseteq 2^X \times 2^X$ (directed).

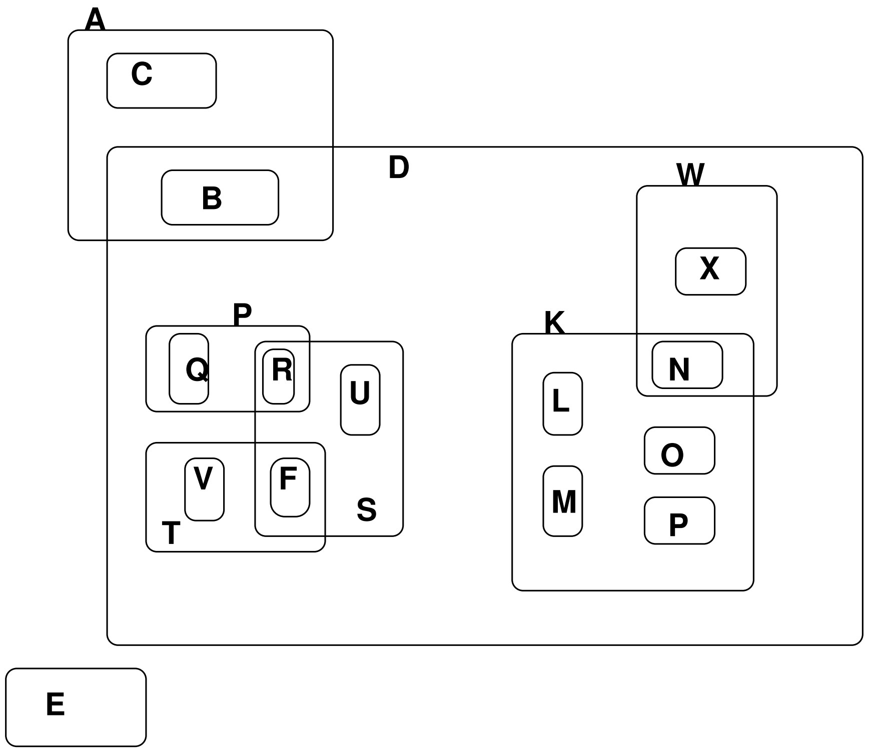
Higraphs: combining graphs and Venn diagrams

- sets + cartesian product
- hypergraphs

Blobs: set *inclusion, not membership*

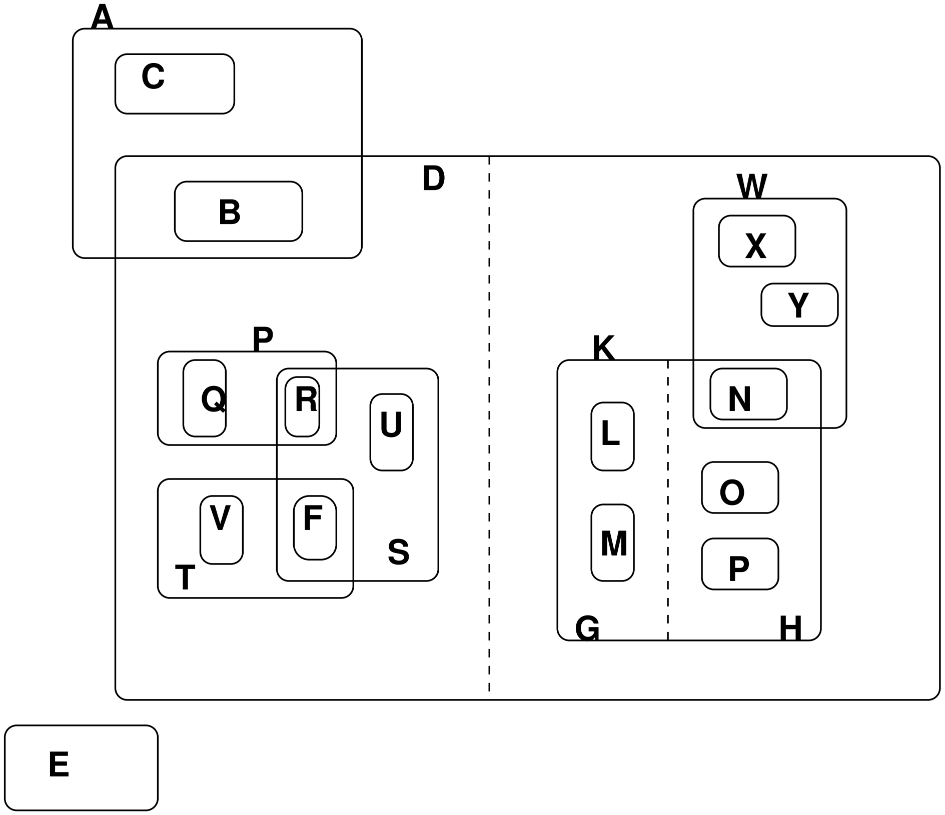


Unique Blobs (atomic sets, no intersection)



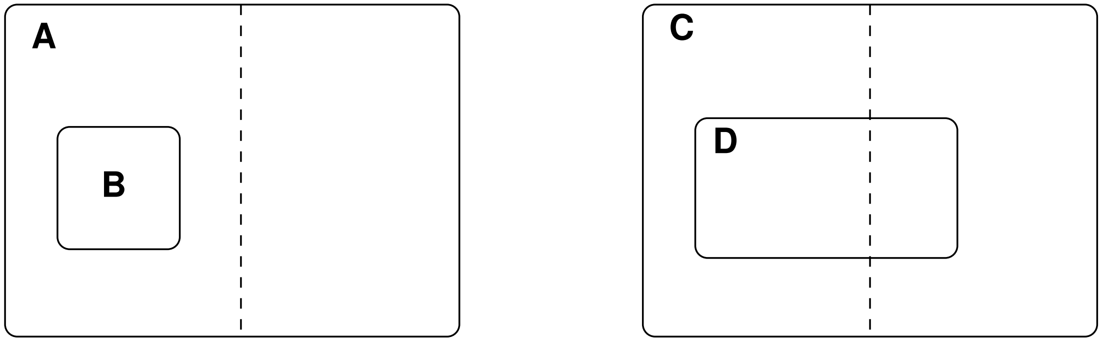
- atomic blobs are identifiable sets
- other blobs are union of enclosed sets (*e.g.,* $K = L \cup M \cup N \cup O \cup P$)
- empty space meaningless, identify intersection (*e.g.,* $N = K \cap W$)

Unordered Cartesian Product: *Orthogonal* Components

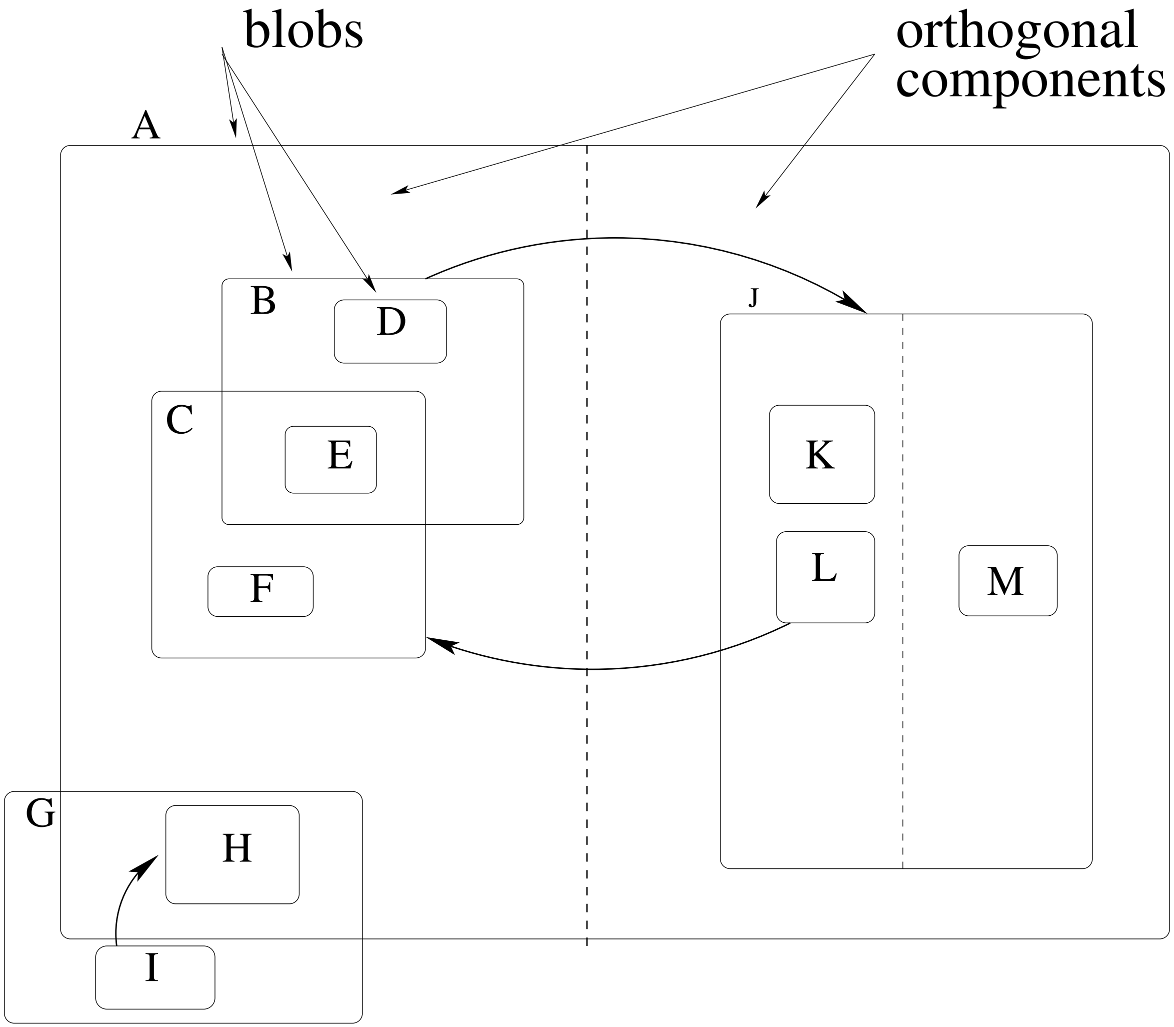


$$K = G \times H = H \times G = (L \cup M) \times (N \cup O \cup P)$$

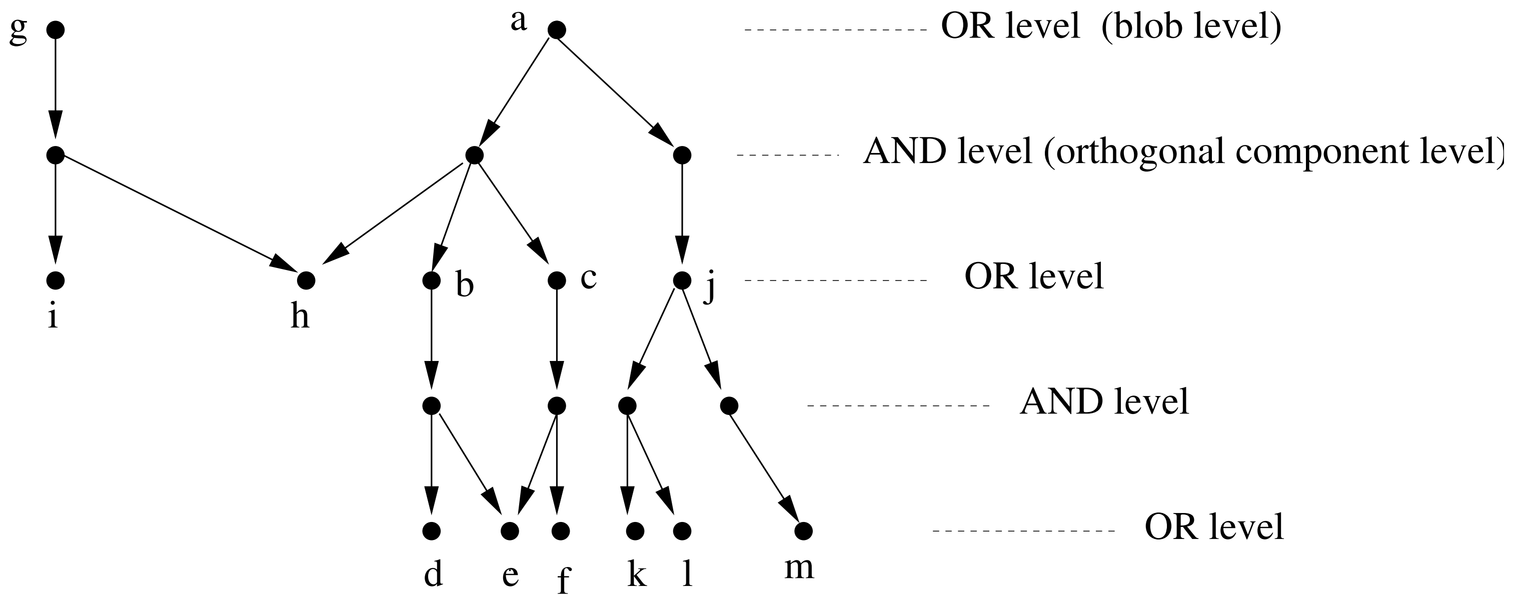
Meaningless syntactic constructs



Simple Higraph



Induced Acyclic Graph (blob/orth comp alternation)



OR level (blob level)

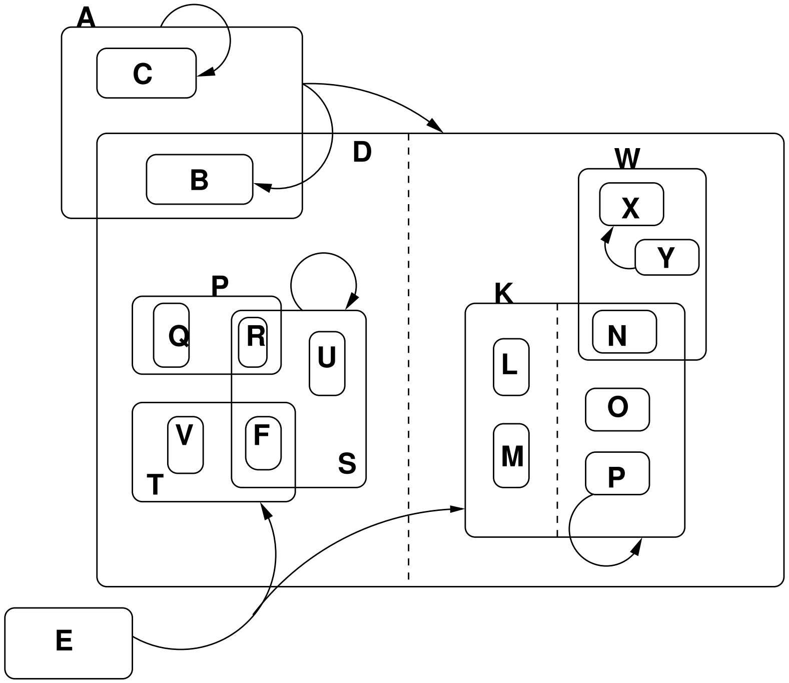
AND level (orthogonal component level)

OR level

AND level

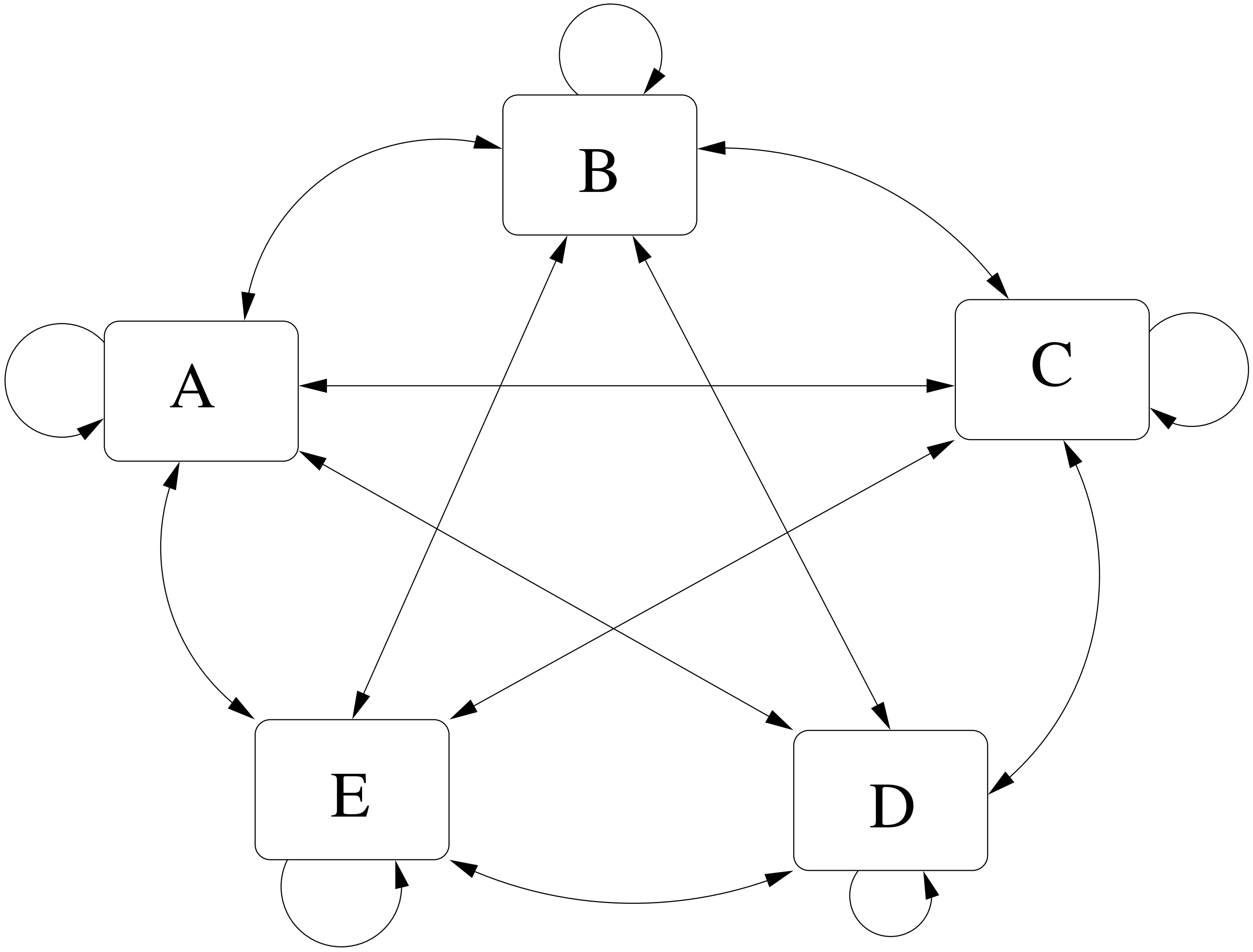
OR level

Adding (hyper) edges

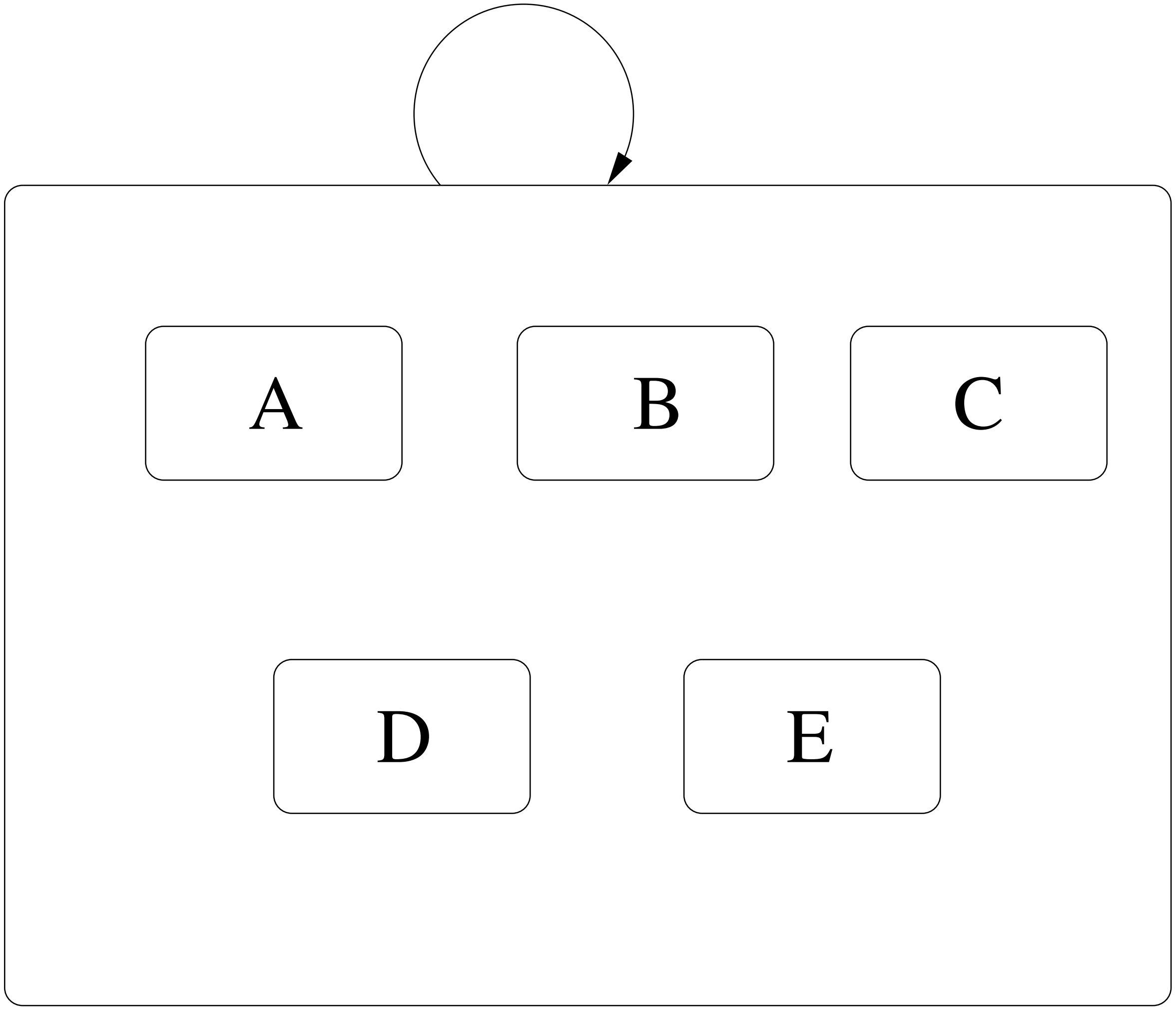


- *hyper*edges
- attach to contour of *any* blob
- inter-level possible (*e.g.,* denote global variables binding)

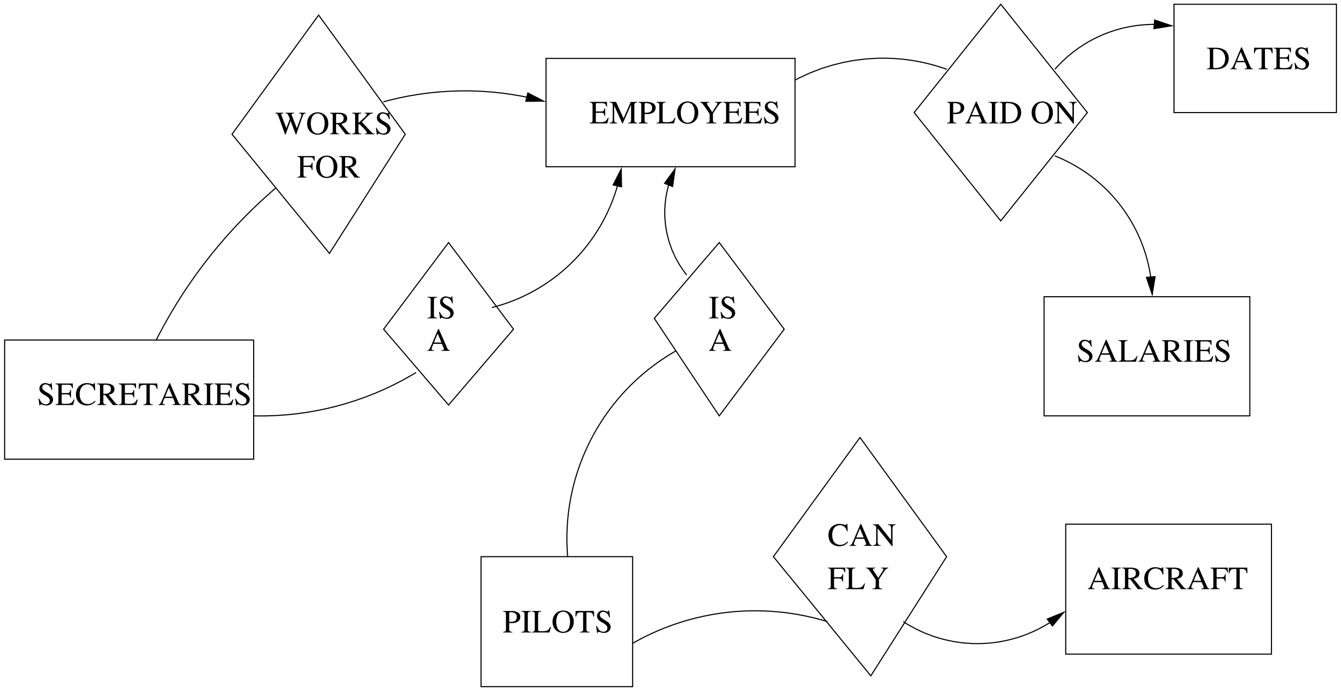
Clique Example



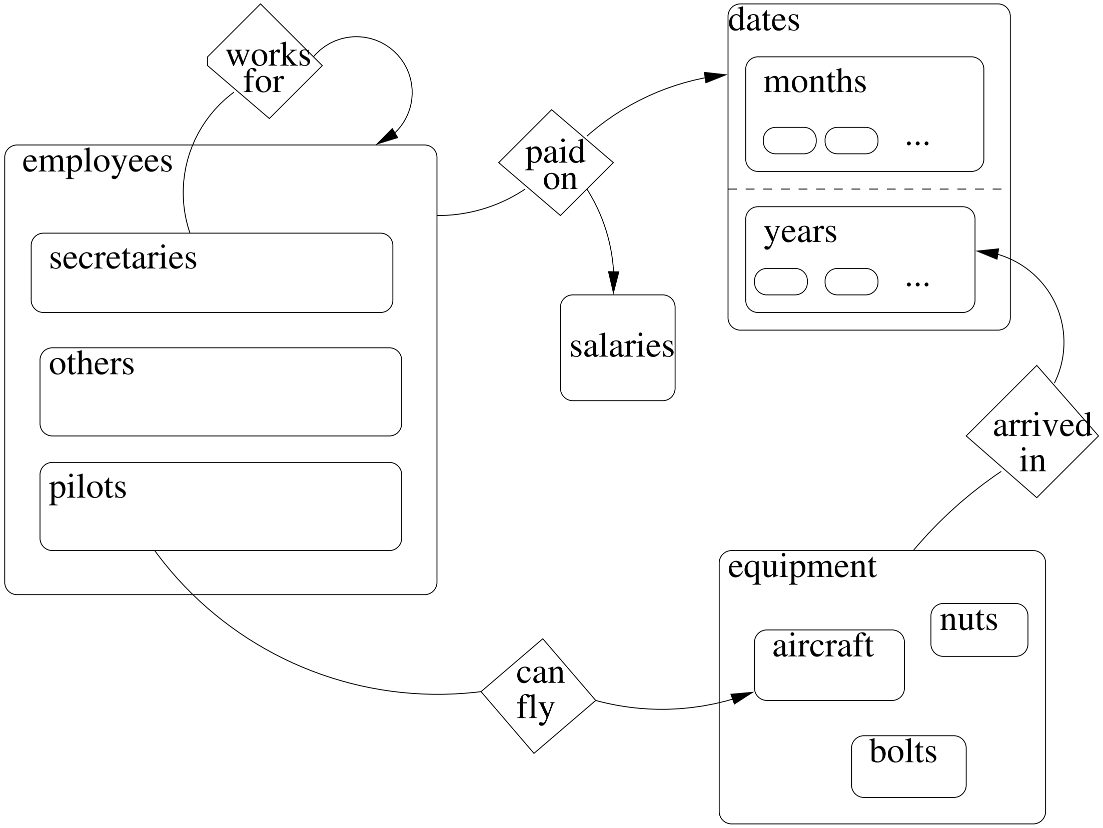
Clique: fully connected semantics



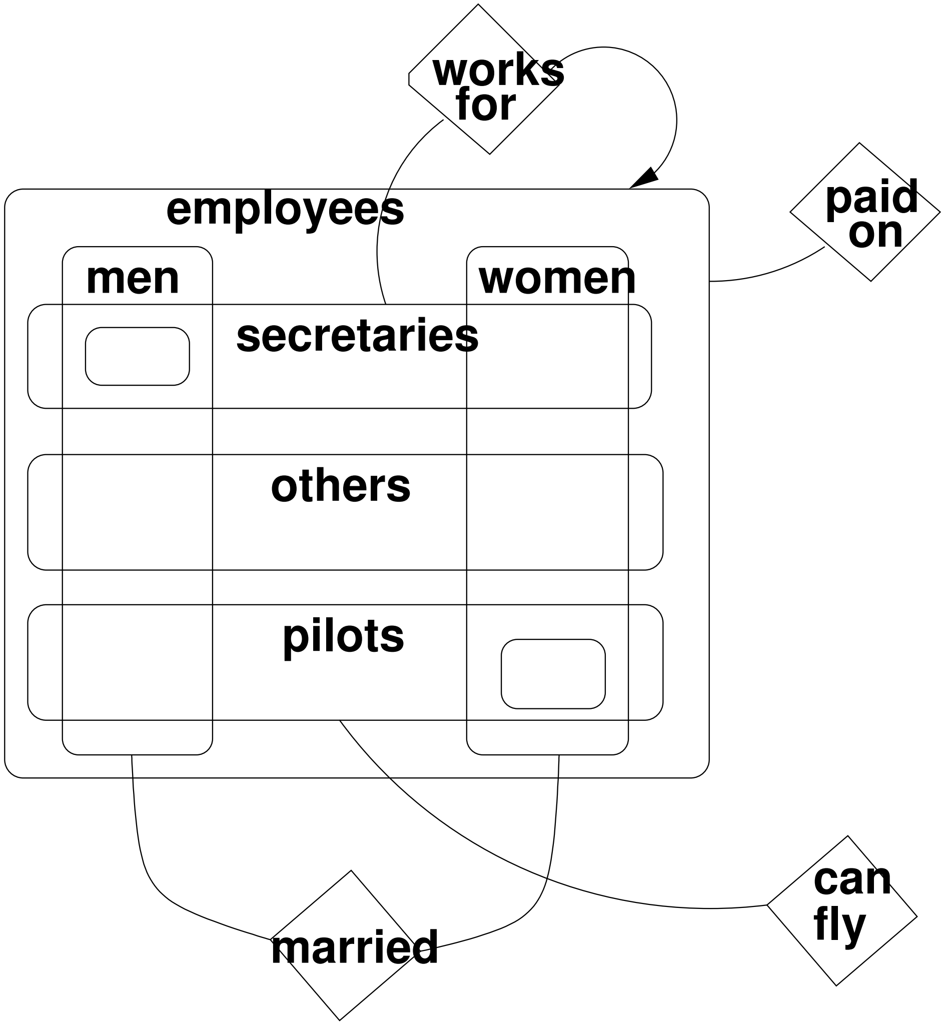
Entity Relationship Diagram (`is-a`)



Higraph version of E-R diagram



Extending the E-R diagram



Formally (syntax)

A higraph H is a quadruple

$$H = (B, E, \sigma, \pi)$$

B: finite set of all unique *blobs*

E: set of hyperedges

$$\subseteq X \times X, \quad \subseteq 2^X, \quad \subseteq 2^X \times 2^X$$

The subblob (direct descendants) function σ

$$\sigma : B \to 2^B$$

$$\sigma^0(x) = \{x\},\ \sigma^{i+1} = \bigcup_{y \in \sigma^i(x)} \sigma(y),\ \sigma^+(x) = \bigcup_{i=1}^{+\infty} \sigma^i(x)$$

Subblobs^{+} cycle free

$$x \notin \sigma^{+}(x)$$

The partitioning function π associates *equivalence relationship* with x

$$\pi : B \rightarrow 2^{B \times B}$$

Equivalence classes π_i are *orthogonal components* of x

$$\pi_1(x), \pi_2(x), \ldots, \pi_{k_x}(x)$$

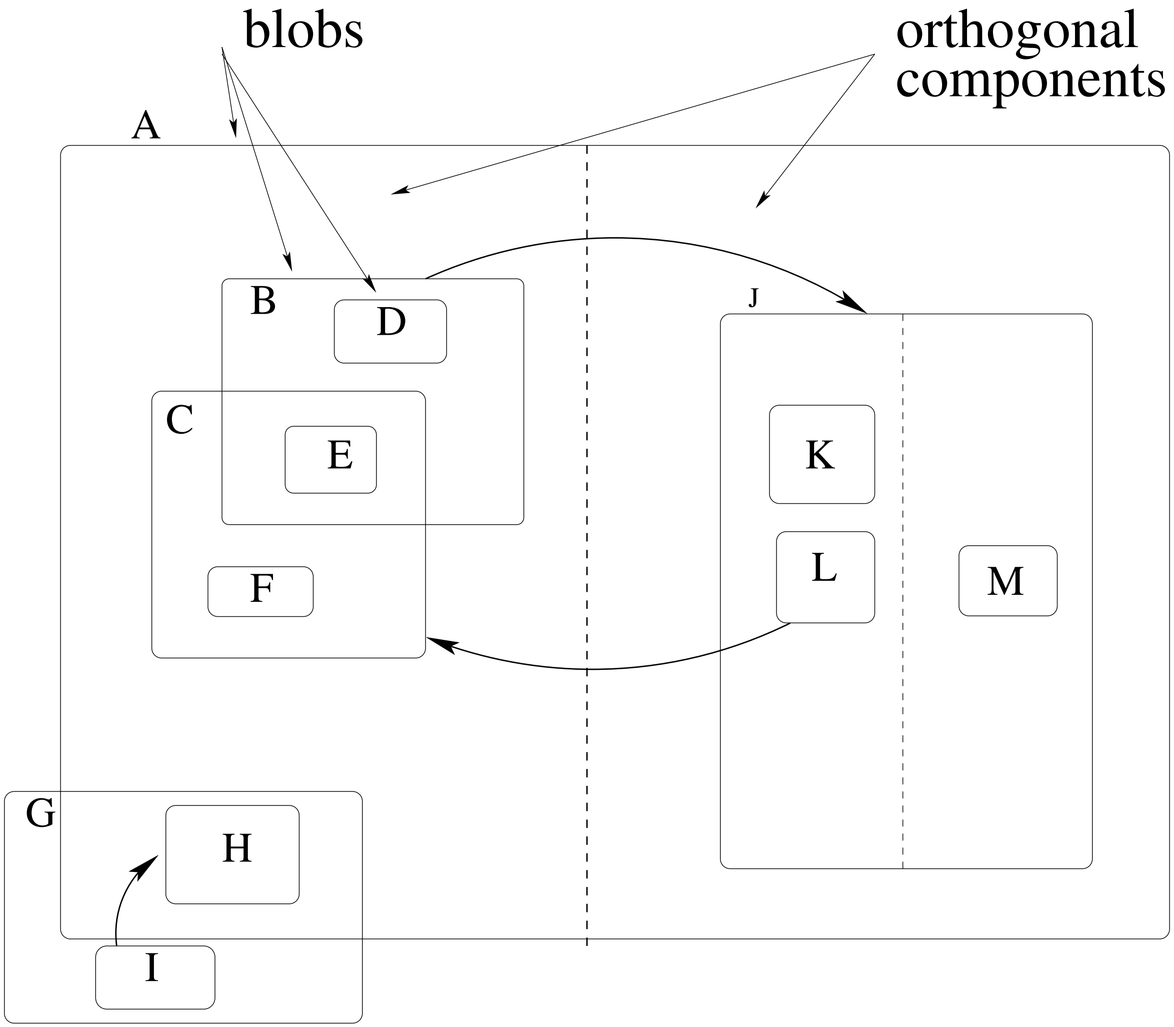
$k_x = 1$ means a single orthogonal component (no partitioning)

Blobs in different orthogonal components of x are *disjoint*

$$\forall y, z \in \sigma(x) : \sigma^{+}(y) \cap \sigma^{+}(z) = \emptyset$$

unless in the same equivalence class

Simple Higraph



Induced Orthogonal Components

$$B = \{A, B, C, D, E, F, C, G, H, I, J, K, L, M\}$$

$$E = \{(I, H), (B, J), (L, C)\}$$

$$\rho(A) = \{B, C, H, J\}, \rho(G) = \{H, I\}, \rho(B) = \{D, E\}, \rho(C) = \{E, F\},$$

$$\rho(J) = \{K, L, M\}$$

$$\rho(D) = \rho(E) = \rho(F) = \rho(H) = \rho(I) = \rho(K) = \rho(L) = \rho(M) = \emptyset$$

$$\pi(J) = \{(K, K), (K, L), (L, L), (L, K), (M, M)\}$$

Induces *equivalence classes* $\pi_1(J) = \{K, L\}$ and $\pi_2(J) = \{M\}$, ...

These are the *orthogonal components*

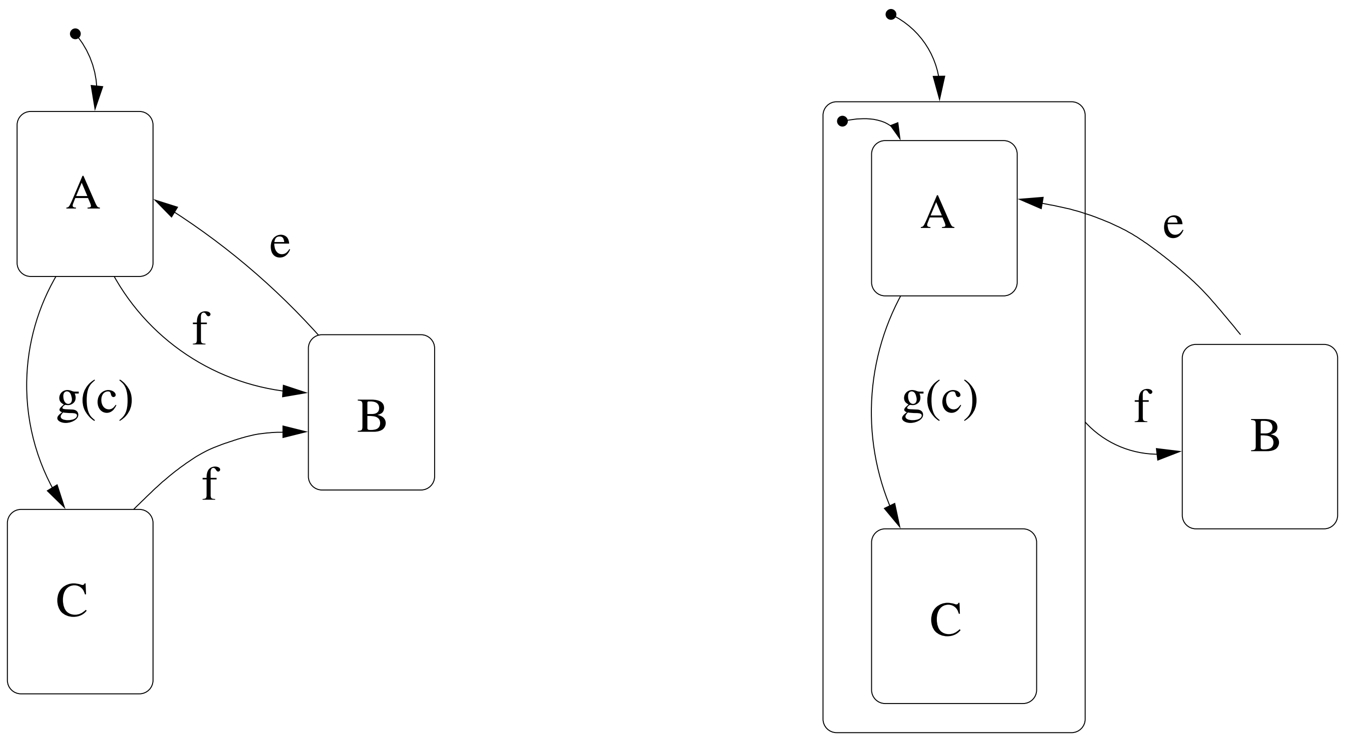
Higraph applications (add specific meaning)

1. E-R diagrams
2. data-flow diagrams (activity diagrams)
   edges represent (flow of) data
3. inheritance
4. Statecharts

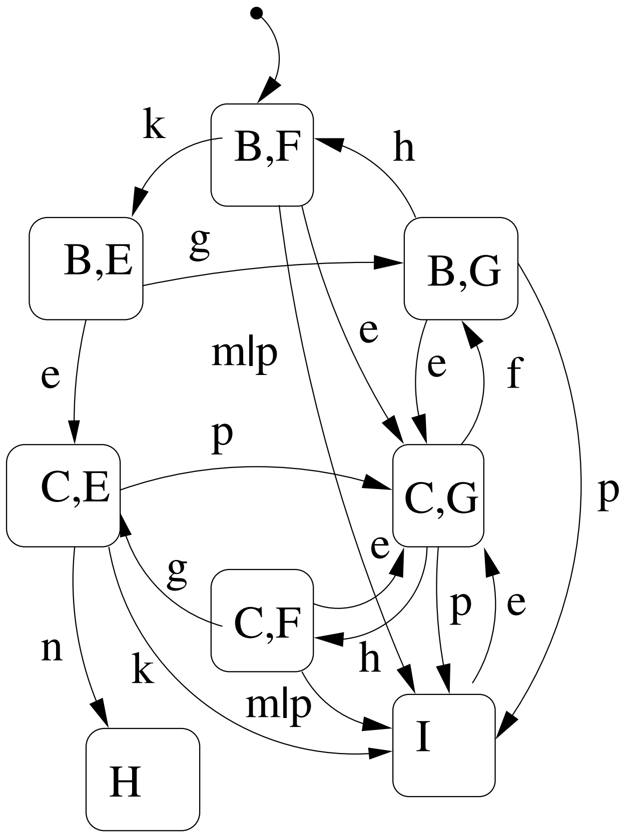
Statecharts =
state diagrams + depth + orthogonality + broadcast

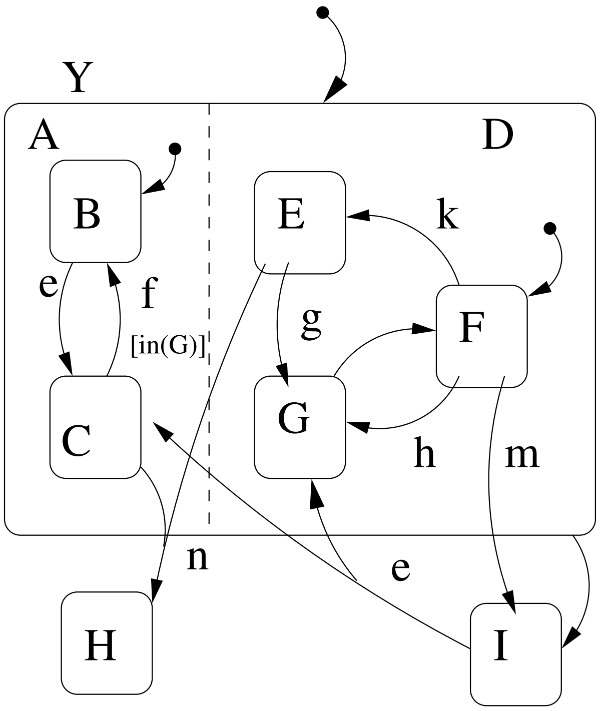
- Reactive Systems (event driven, react to internal and external stimuli)
- like Petri Nets, CSP, CCS, sequence diagrams, . . .
- graphical but formal and rigourous for
  - analysis
  - code generation
- solve FSA problems:
  - flat ⇒ hierarchy ⇒ re-use
  - represent large number of *transitions* concisely
  - represent large number of (product)*states* concisely
  - sequential ⇒ concurrent

Depth (XOR), semantics through flattening

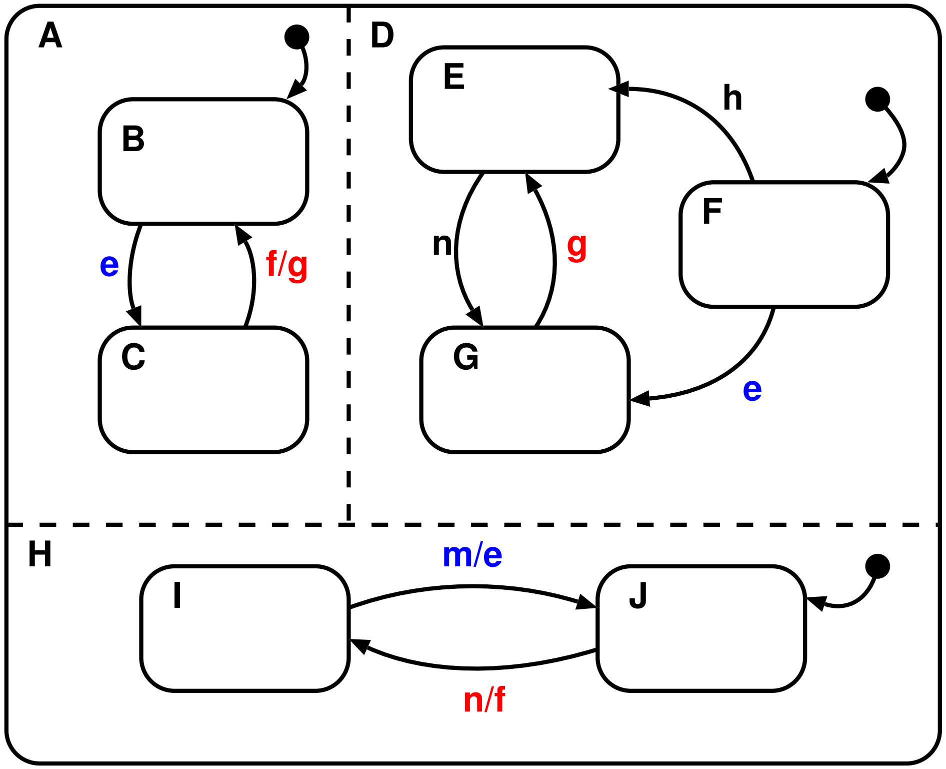


Orthogonality (AND), semantics through flattening



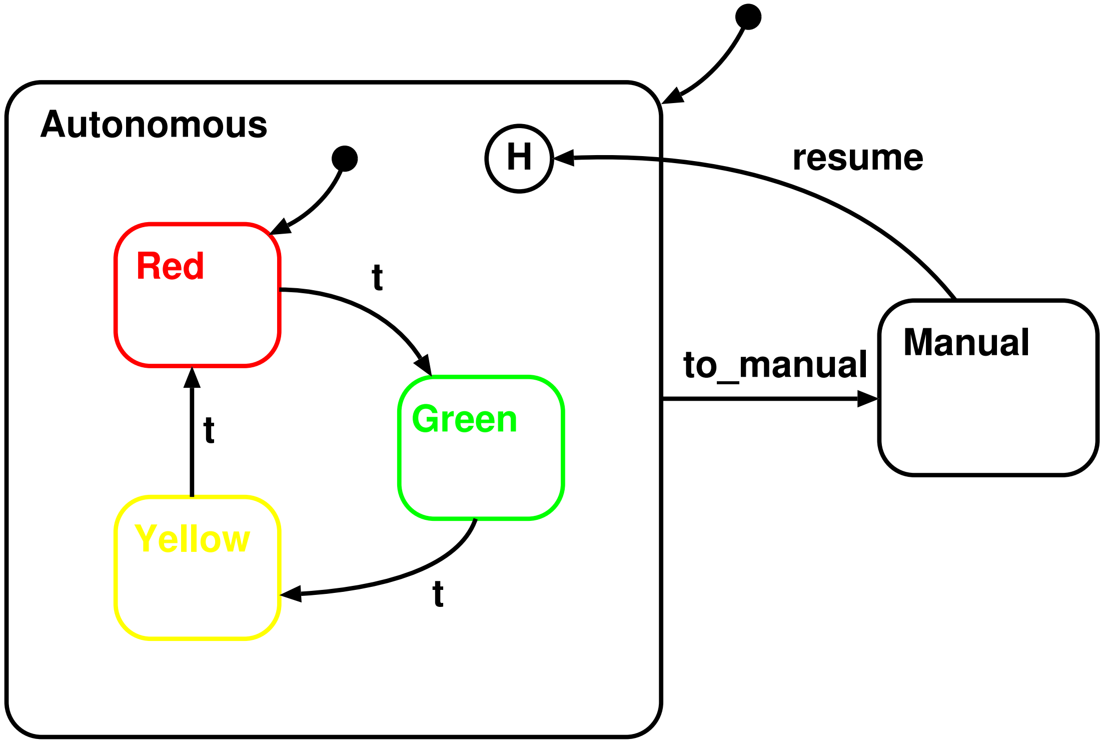


Broadcasting (output events)

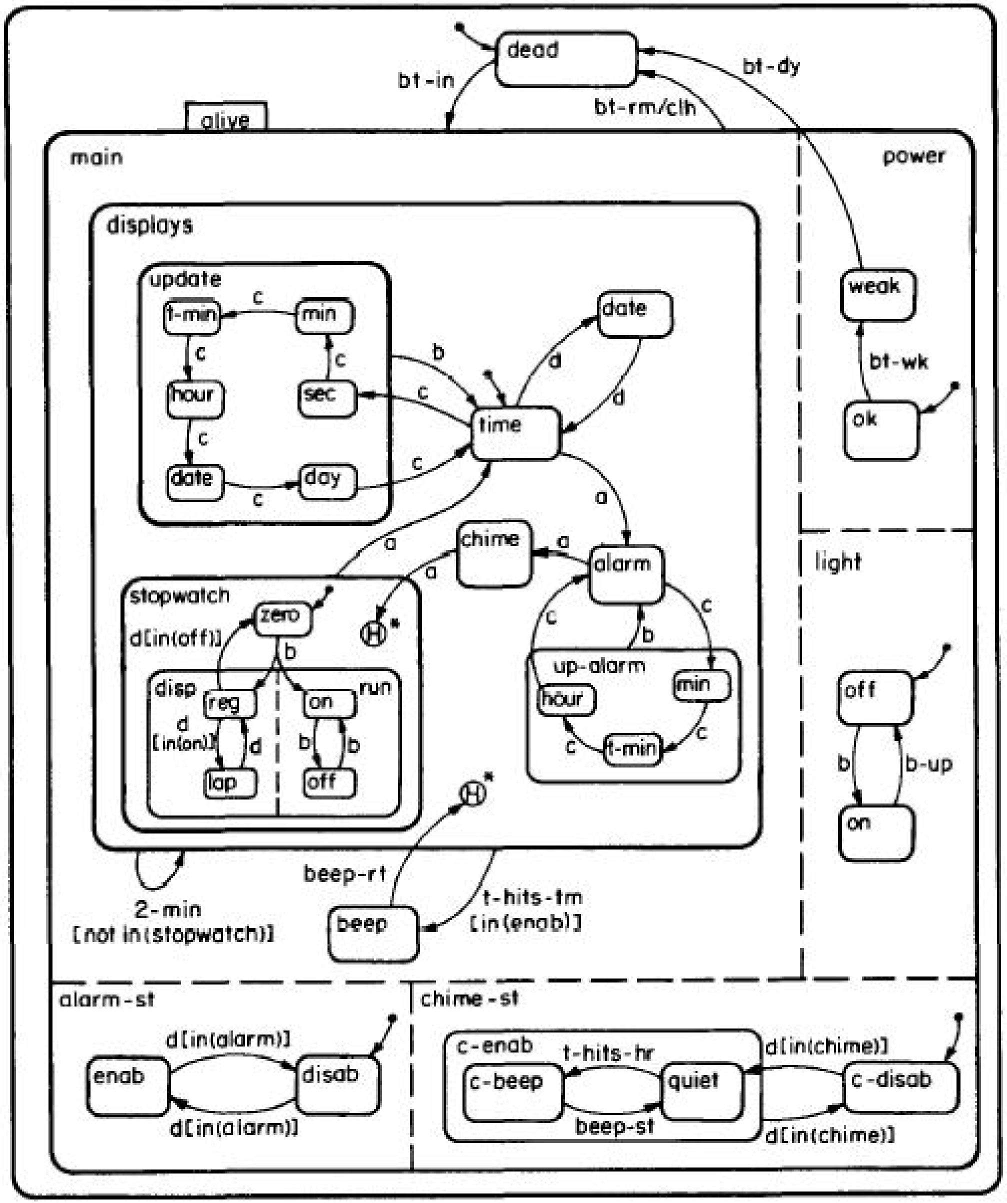


Input Segment: **nmnn**

History States



Stopwatch Example


dead
bt-in
bt-dy
bt-rm/clh
alive
main
power
displays
update
c
t-min
min
c
c
hour
sec
c
c
date
day
c
c
b
d
d
date
time
weak
bt-wk
ok
a
a
chime
a
a
alarm
stopwatch
zero
d[in(off)]
b
H*
c
c
b
up-alarm
min
disp
reg
on
run
hour
d
[in(on)]
d
b
b
c
t-min
c
lap
off
light
off
b
b-up
on
H*
beep-rt
2-min
[not in(stopwatch)]
beep
t-hits-tm
[in(enab)]
alarm-st
chime-st
d[in(alarm)]
enab
disab
d[in(alarm)]
c-enab
t-hits-hr
c-beep
quiet
beep-st
d[in(chime)]
c-disab
d[in(chime)]

Extensions

- time: `after(10s)`
- guards: `[OC in(C)]`
- parametrized events: `ev(p1,p2)`
- narrowcast: `destination.ev(p1,p2)`, `destination->ev(p1,p2)`
- states *vs.* variables
- arrow: R, negative arrow: not R, absence of arrow: don't know
- don't know blobs
- Zoom outs (interface)