Node Minimization in Logic

Node minimization can be applied to

- Combinational Logic
- Sequential Logic
- ω-automata

In all cases, synthesis/optimization can be divided into:

- deriving all the flexibility (allowed from at a node) from from surround environment and specification
- synthesizing/optimizing an implementable behavior

Outline:

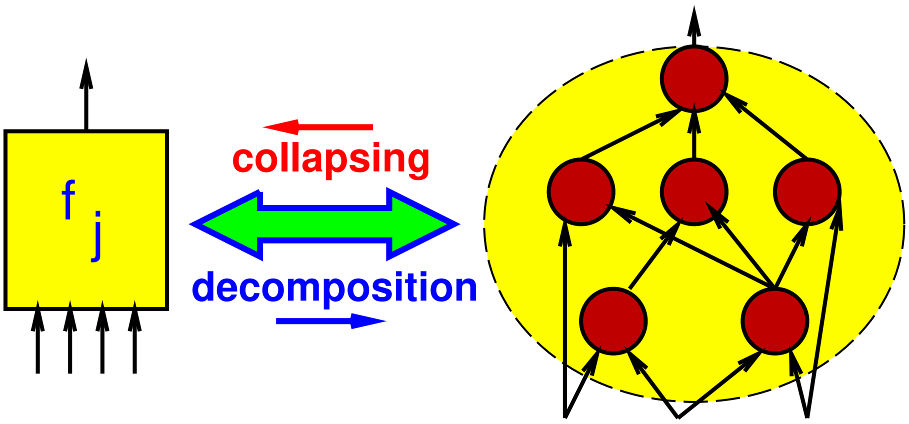
- Quick review of combinational logic
- Derivation of E-machine for sequential logic
- Extension of E-machine to automata

Combinational Logic - Three Main Optimization Steps

All Heuristic

1. Decomposition / Collapsing

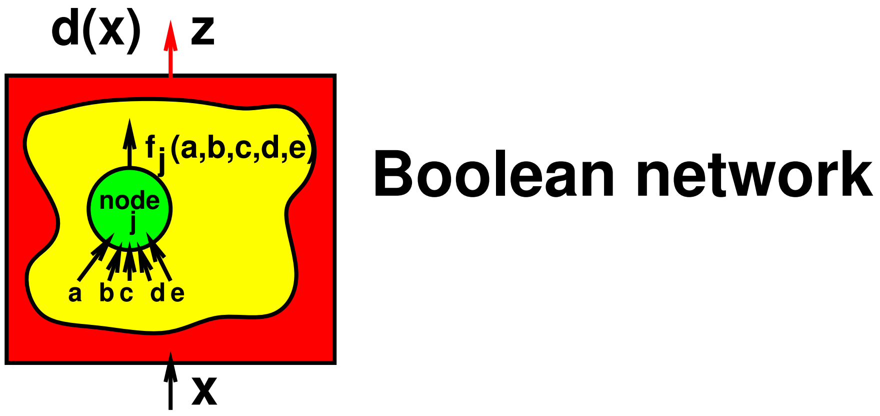
   - factorization
   - kerneling
   - boolean division



2. **Node Minimization**
3. Technology Mapping

Combinational Logic - Node Flexibility

Problem: Simplify logic function f_j



Don't Cares: (summarize effect of rest of network on node j)

$EXDC$ - given external don't cares
$d = (d_1(x), \ldots, d_m(x))$

SDC_j - local input patterns that never occur

ODC_j - primary input values for which f_j has no effect on primary outputs

Theorem: g_j is permissible at node j if and only if

$$f_j\overline{\delta_j} \leq g_j \leq f_j + \delta_j$$

where $\delta_j \equiv SDC_j + ODC_j + EXDC$.

Thus f_j, δ_j effectively **summarize** all functions permissible at node j.

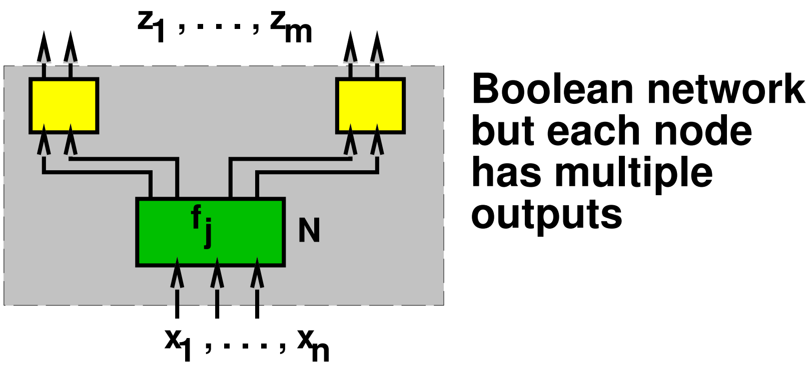
Can represent this set by its characteristic function

$$O(x, y_j) = (f_j\overline{\delta_j} \Rightarrow y_j) \wedge (\overline{(f_j + \delta_j)} \Rightarrow \overline{y}_j)$$

Node Minimization: Can use standard two-level minimizers to do node minimization.

Combinational Logic - Multiple Output Functions

Network of Multiple Output Functions:



What is the set of permissible functions for f_j at N?

- Don't care formulation does not work

- **Boolean Relation:** (completely specified)

$$\mathcal{B} : B^n \longrightarrow 2^{B^m}$$

- **Compatible functions:** A function g_j is compatible with a Boolean relation $\mathcal{B}$ (or contained in):

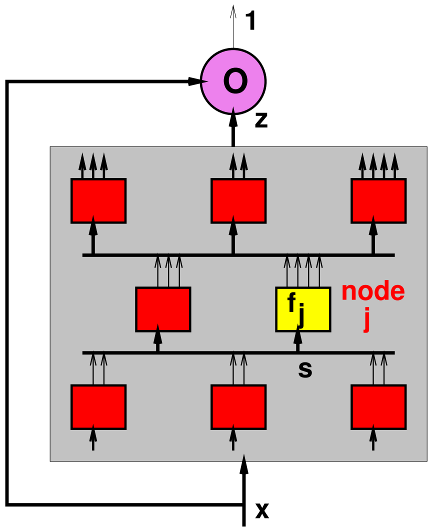
$$g_j \prec \mathcal{B} \Longleftrightarrow \forall x, g_j(x) \in \mathcal{B}(x)$$

Can represent $\mathcal{B}$ as its characteristic (or observability) function:

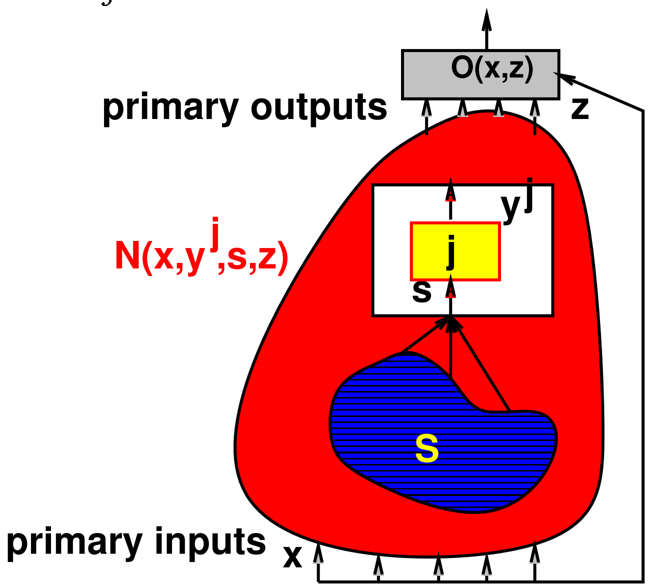
$$\mathcal{O}(x,z) = 1 \Longleftrightarrow z \in \mathcal{B}(x)$$

General Problem

Given network of multiple-output functions and Observability specification:



Summarize all implementable functions at node j, i.e. derive $\mathcal{O}_j(\mathbf{s}, \mathbf{y^j})$ such that: $f(\mathbf{s})$ is implementable at node $j \iff f \prec \mathcal{O}_j(\mathbf{s}, \mathbf{y^j})$



$$\mathcal{O}_j(\mathbf{s}, \mathbf{y^j}) = \forall \mathbf{x}, \mathbf{z}[N(\mathbf{x}, \mathbf{y^j}, \mathbf{s}, \mathbf{z}) \Rightarrow \mathcal{O}(\mathbf{x}, \mathbf{z})]$$

Node Minimization

Optimization Problem: Given $O(\mathbf{s}, \mathbf{y_j})$, find "best f_j such that $O(\mathbf{s}, f(\mathbf{s})) \equiv 1$

Boolean Relation Minimizers:

- Exact - Somenzi (1989),

  Like Quine-McCluskey

  - define prime implicants
  - generate all primes
  - solve covering (binate) problem

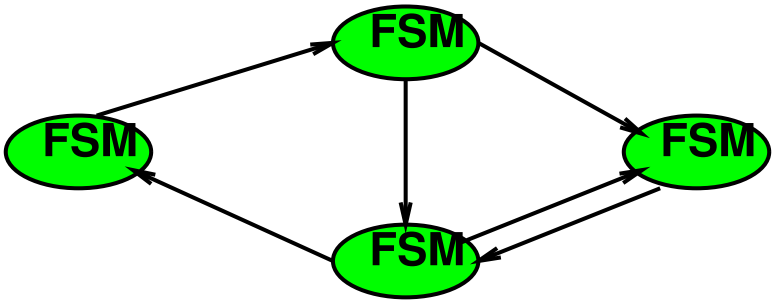
- Heuristic - Watanabe (1991)

  Uses BDD's to represent Boolean relation - iterates reduce, expand irredundant until no change.

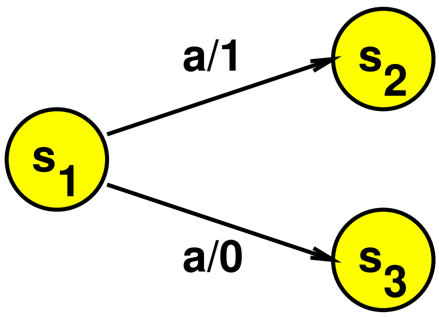
Sequential Synthesis

(Combinational logic and memory)

- **Computational Model**
  Network of FSM's: *(possibly cyclic)*



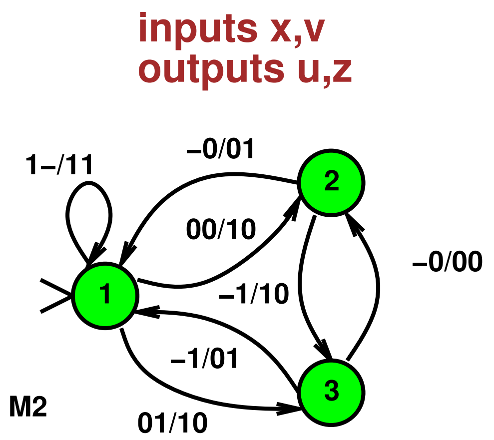
- **FSM's**
  - Mealy or Moore
  - symbolic ($v \in \{stop, go, caution\}$)
  - nondeterministic



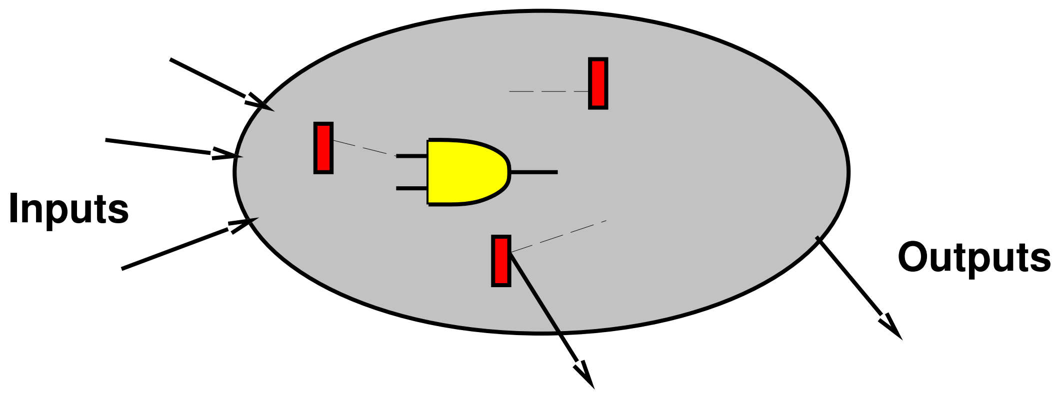
  - several inputs and outputs per node

FSM Specification

FSM's can be given by a state transition graph (STG): (KISS format)

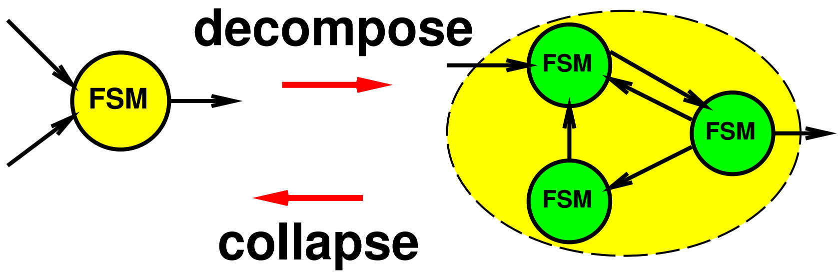


or by a netlist of gates and latches: (BLIF-MV format)



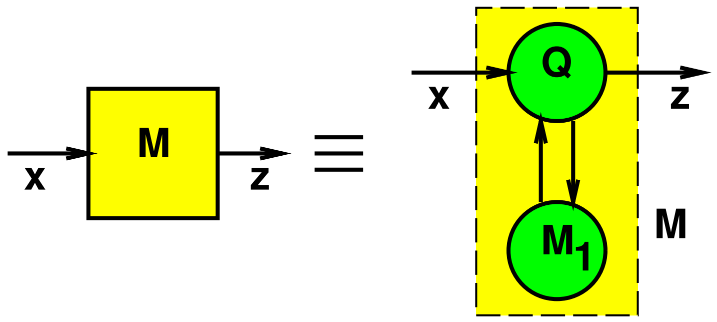
Sequential Synthesis

- **Decomposition / Collapsing**



"Boolean" Division:

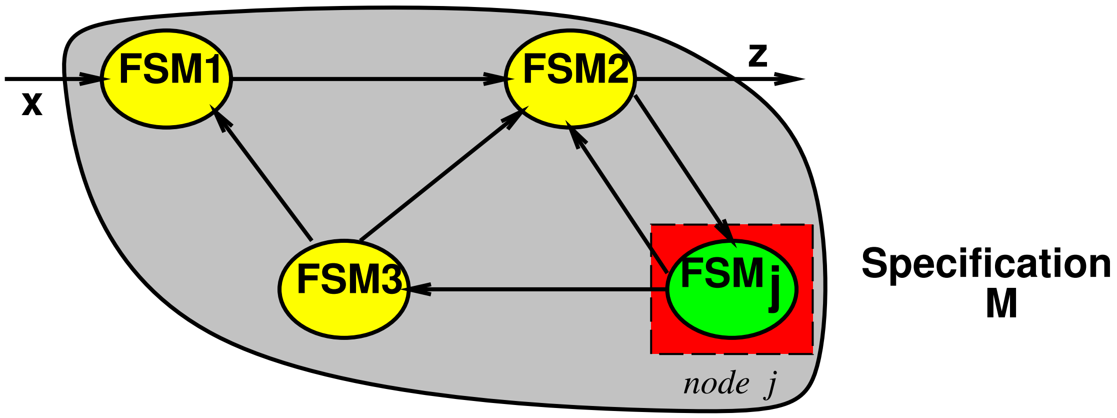
Given two FSM's M and M_1, find FSM Q (quotient) such that $M_1 \times Q = M$.



- **Node Minimization**

Node Minimization: Two Problems

Given FSM Network and specification $M : x \longmapsto z$



Problem 1: Summarize and represent all permissible (implementable) machines at

$$\text{node } j : \mathcal{S}_j$$

Want : M_j *is permissible* $\Longleftrightarrow M_j \prec \mathcal{S}_j$ (i.e. S_j "simulates" M_j)

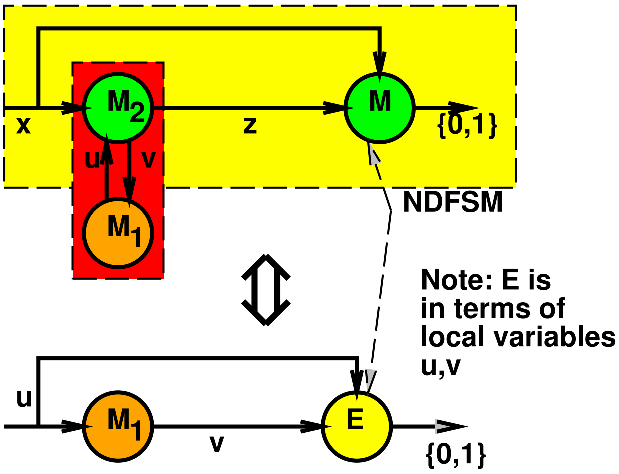
- Past (partial) results: "don't care" sequences
  - input (*those that can't occur*)
    Kim and Newborn (1972)
    Devadas and Newton (1990)
    Somenzi (1991), Wang and Brayton (1993)
  - output (*pairs that can't be distinguished*)

Problem 2: Find **optimum** M_j, such that $M_j \prec \mathcal{S}_j$

The E-Machine (Watanabe ICCAD'93)

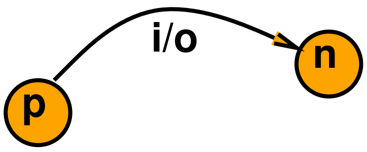
Theorem:

Given specification $M : x \longmapsto z$ (a NDFSM) and FSM's M_1, M_2, there exists a NDFSM E (called the **E-machine**) such that a FSM N is permissible (**implementable** *would be a better word here*) at $M_1 \Longleftrightarrow N \prec E$



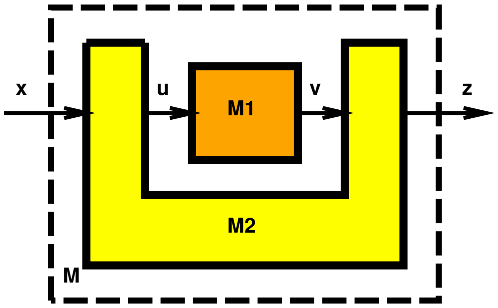
- E can be computed by a)*complicated*) fixed point calculation

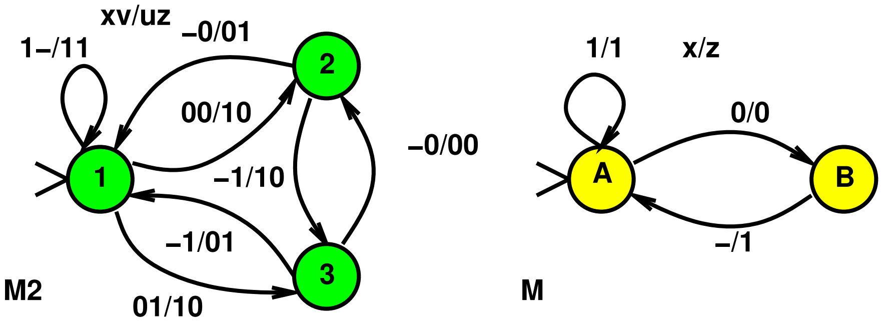
- NDFSM's are represented using characteristic function of transition relation: $T(p, i, n, o) = 1$



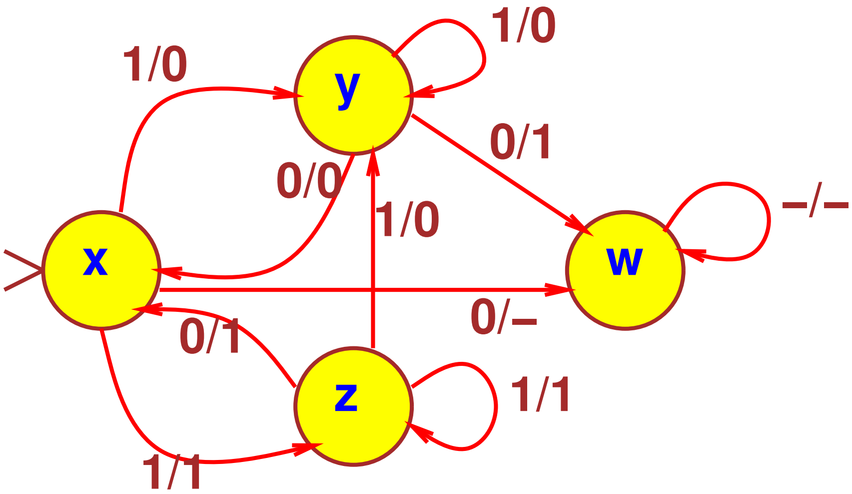
Thus E summarizes the permissible machines for M_1 (*solution to Problem 1*)

Example:





E-machine:



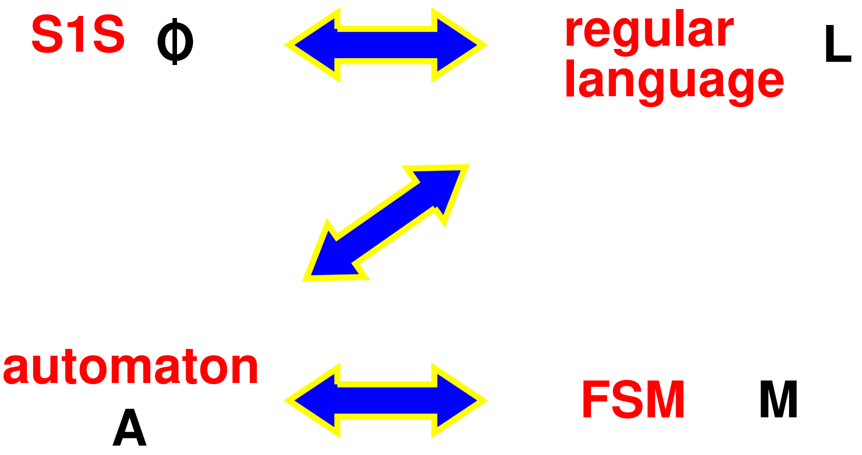
E-machine Construction Using Logic S1S

S1S stands "**S**econd order logic with **1** **S**uccessor"
Example: *If a 1 appears at any even clock tick in X a 1 also appears in Y at that time.*

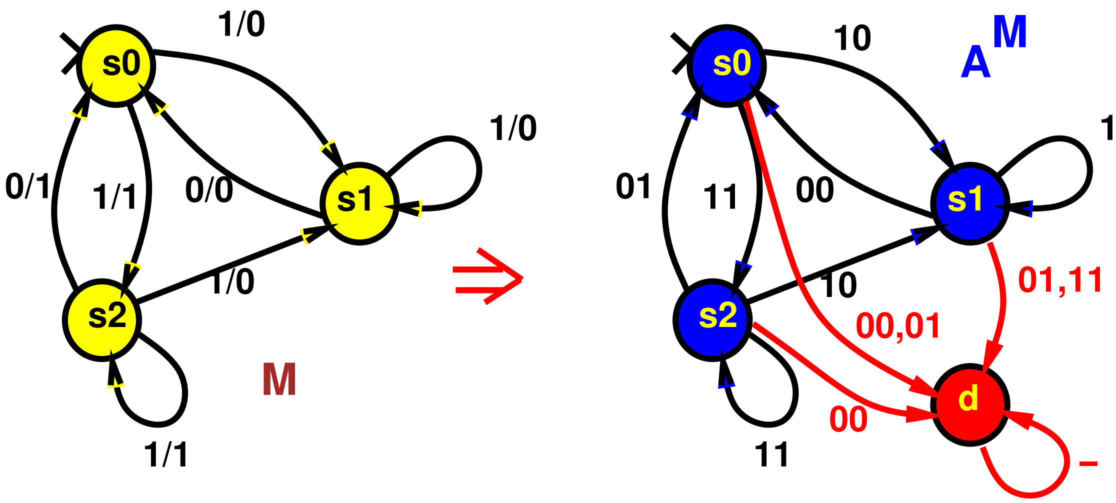
$$\phi_2(X,Y) = (\forall e)[((e \in \phi_{even}) \wedge (e \in X)) \Rightarrow e \in Y)]$$

Theorem:[Büchi 1961] *A $*$-language $\mathcal{L}$ is definable in* S1S *if and only if $\mathcal{L}$ is regular*

A language is regular $\Leftrightarrow$ accepted by an automaton.



Every FSM $M \Rightarrow$ automaton A^M obtained by



S1S -DEFINITIONS & BACKGROUND

***-Languages:** Given finite set Σ

- Σ^*: set of all finite sequences over Σ
- A *-language is a subset of Σ^*

Notation:
lower case variable takes value in Σ,
upper case in Σ^*

Finite Automata (FA) – 5-tuple (Σ, S, s_0, T, A)

- Σ – finite set called the *alphabet*
- S – finite set of *states*
- $s_0 \in S$ – *initial* state
- $T \subset S \times \Sigma \times S$ – *transition relation*
- $A \subset S$ – *accepting* states

Deterministic FA – $(\forall s)(\forall x)\,[\,|\,\{t : (s,x,t) \in T\}\,| \leq 1\,]$;
otherwise *non-deterministic*

DEFINITIONS & BACKGROUND-II

- String $X \in \Sigma^*$ *accepted* by FA if there exists a sequence of states (**a run**) $\sigma_0\sigma_1 \ldots \sigma_n$ such that
  1. $n = \mid X \mid$,
  2. $\sigma_0 = s_0$, $\sigma_n \in A$, and
  3. $(\forall i)\,[(\sigma_i, X_i, \sigma_{i+1}) \in T]$
- Language of FA = set of strings accepted by it; a $*$-language is *regular* if it can be constructed by $\{x \in \Sigma, ., +, *, (,)\}$, e.g. $(a(b+c)*)*$.

Facts:

1. Language is regular $\Leftrightarrow$ it is accepted by a DFA (Power set construction)

2. Class of regular languages is closed under
   - Union, Intersection (product construction)
   - Projection (eraser construction)
   - Complementation (determinize with power set construction $\Rightarrow$ trivial complementation).

INTRODUCTION & BACKGROUND-III

Logic `S1S` (*Second order theory of one successor*) - Natural logic for sets of sequences.

- formalism for describing properties of sequences;
- powerful mechanism for analysis and manipulation of sequential systems (makes available full power of logic)

Alphabet: $\{0, S, =, <, \in, \wedge, \neg, \exists, x_1, x_2, \ldots, X_1, X_2, \ldots\}$

Syntax:

- *Terms:* $0 \mid x_i \mid St_1$ where t_1 is a term.

  Examples of terms – 0, $SS0$, $SSSSx_3$.

- *Atomic formulae:* $t_1 = t_2 \mid t_1 < t_2 \mid t_1 \in X_k$ where t_1, t_2 are terms, and X_k is a finite set of terms.

  Examples of atomic formulas – $0 < S0$, $x_3 = SSSx_5$, $Sx_7 \in X_2$.

- `S1S` *formulas:* $\phi \wedge \psi \mid (\neg\phi) \mid (\exists x_i)\phi \mid (\exists X_i)\phi$ where ϕ, ψ are `S1S` formulae or atomic formula.

  Examples of `S1S` formulas – $(0 < S0) \wedge (Sx_7 \in X_2)$, $(\exists X)\,(\exists x)\,[(x \in X) \wedge (Sx \in X)]$.

INTRODUCTION & BACKGROUND-V

S1S **Semantics:** Interpret over $\omega = \{0, 1, 2, \ldots\}$.

Example 1: (Existence of least elements in non empty subsets of ω)

$$\begin{aligned}\psi &= (\forall X)\,[(\exists x)\,(x \in X) \rightarrow \\ &\quad (\exists y)\,((y \in X) \wedge \neg((\exists z)\,(z \in X \wedge (z < y))))]\end{aligned}$$

Example 2: (Defining subsets of ω which contain 5 whenever they contain 3.)

$$\phi_0(X) = (SSS0 \in X) \rightarrow (SSSSS0 \in X)$$

Example 3: (Defining the subset of even integers)

$$\phi_1(X) = (0 \in X) \wedge \neg(S0 \in X) \wedge (\forall x)\,(x \in X \leftrightarrow SSx \in X)$$

Example 4: (Defining the relation "every even number in X is in Y")

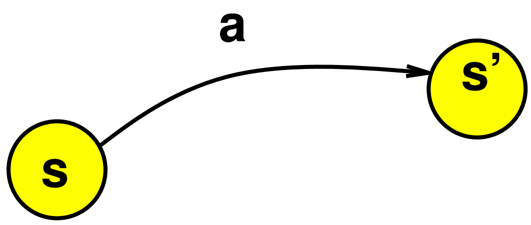
$$\phi_2(X, Y) = (\forall x)[(\forall Z)\,(\phi_1(Z) \wedge x \in Z) \rightarrow (x \in X \rightarrow x \in Y)]$$

Given formula $\theta(X_1)$ class of subsets of ω *defined* by $\theta(X_1)$ is $\{\beta \subset \omega \mid \theta(\beta)$ is true$\}$; Generally, formulae $\phi(X_1, X_2, \ldots, X_n)$ define subsets of $(\{0,1\}^n)^\omega$

Observation: Finite subsets of $\omega \overset{1-1}{\leftrightarrow}$ sequences over $\{0,1\}$. For example $\{0,2,3\}$ is represented by 1011. We have to be careful about 0 sequences, e.g. to distinguish 00 from 000 etc. (Can do this by picking a length and extending each set to that length by putting 0's at the end) Thus, $\mathtt{S1S}$ formulae define $*$-languages.

Theorem:[Büchi 1961] *A $*$-language is definable in* $\mathtt{S1S}$ *if and only if it is regular*

Proof:(*sketch*)
Reverse direction: Given an automaton, there is a straightforward coding up of it as a formula. This would be a big OR of the coding of each transition in the automaton.



Proof Continued

Forward direction: Given a formula, we want to produce an equivalent automaton. We use induction on length of formula.

- Base Case: Trivial automaton for atomic formulae. For example, $x = y$ is given by the two-state automaton with inputs x, y. It goes from the good state to the bad state if at any point, $x_i \neq y_i$.

- $\exists$: Automaton projection (erase some labels on the edges)

- $\wedge$: Automaton Intersection (product automaton)

- $\neg$: Automaton Complementation

Advantages of S1S:
Relationship between automata and `S1S` $\Rightarrow$ formally, succinctly express behaviors as formulae in `S1S`. In particular,

- **Automatic** procedure for deriving automaton from formula

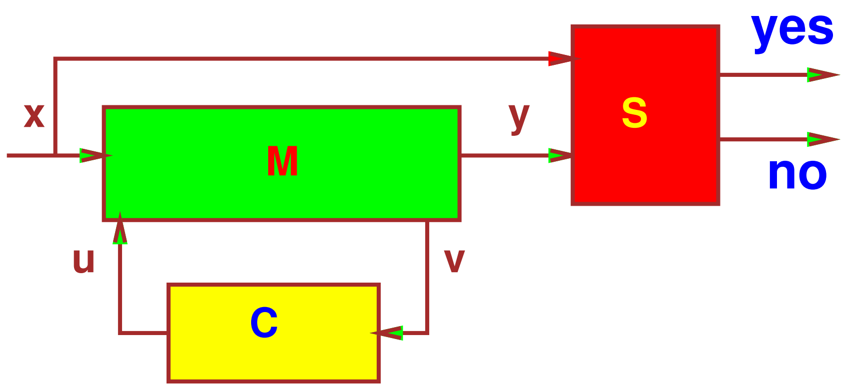
- **Rigorous and constructive** proofs for synthesis procedures

The E-MACHINE-I

Consider machines in configuration below. x, y observable inputs, outputs, $\mathcal{L}^S \subset (\Sigma_X \times \Sigma_Y)^*$ is regular specification, i.e. the set of all acceptable (x, y) sequences

$$((x_1, y_1), (x_2, y_2), \ldots)$$

is accepted by some automaton S.



Theorem: *The set of* **all** *behaviors on u, v which will yield behavior on the inputs and outputs x, y compatible with the specification $S(x, y)$ is given by the following* `S1S` *expression:*

$$\phi^E(U, V) \quad = \quad (\forall X, Y)\, [\phi^M(X, Y, U, V) \rightarrow \phi^S(X, Y)]$$

Proof: Convert each machine to S1S expression.

Constructing the E-Machine in Four Steps

$$\phi^E(U,V) = (\forall X,Y)\,[\phi^M(X,Y,U,V) \rightarrow \phi^S(X,Y)]$$

can be rewritten in terms of manipulating automata:

$$E(u,v) = \neg[\exists(x,y)[M(x,y,u,v) \times \neg S(x,y)]]$$

We use the correspondence between automata and FSM's. Each of the operations in S1S, $\wedge, \neg, \exists$ corresponds to the automata constructions of product, complementation, and projection.

Implies four-step construction for E-machine:

1. Complement the specification,

$$S(x,y) \rightarrow \overline{S}(x,y)$$

2. Form the product machine,

$$P(x,y,u,v) = M(x,y,u,v) \times \overline{S}(x,y)$$

3. Existentially quantify out x, y from P,

$$\widetilde{P}(u,v) = \exists(x,y)[P(x,y,u,v)]$$

4. Complement $\widetilde{P}$,

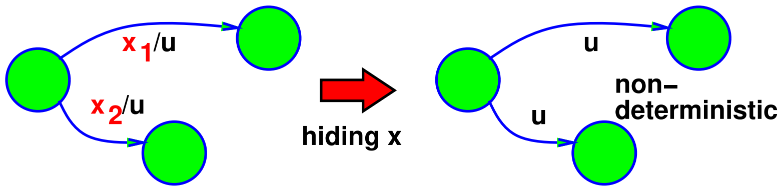
$$E(u,v) = \overline{\widetilde{P}(u,v)}$$

Comments on the Four Steps

1. $S(x, y) \rightarrow \overline{S}(x, y)$
   If specification is non-deterministic, this may be exponential in $|S|$ (*number of states*); a subset construction is required to complement S.

2. $P(x, y, u, v) = M(x, y, u, v) \times \overline{S}(x, y)$
   Easy; just the usual product machine construction. Using BDD's, just multiply the two transition relations.

3. $\widetilde{P}(u, v) = \exists(x, y)[P(x, y, u, v)]$
   Easy; existential quantification is simply a "hiding" variables operation. In general quantifying variables leads to non-deterministic automata.

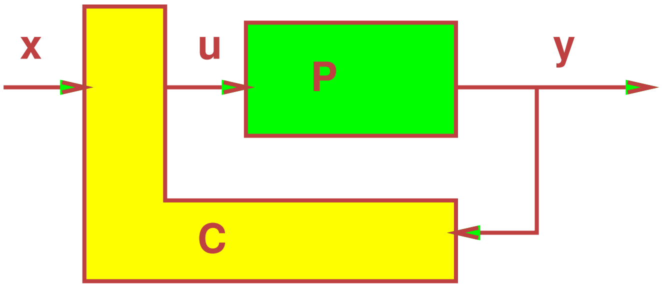


4. $E(u, v) = \overline{\widetilde{P}(u, v)}$
   If $\widetilde{P}(u, v)$ is non-deterministic, then potentially exponential; we have to determinize with the subset construction. However, the resulting automaton E is deterministic, and hence a "pseudo-non-deterministic" FSM.

There are special cases (topologies), where the worst case behavior does not occur.

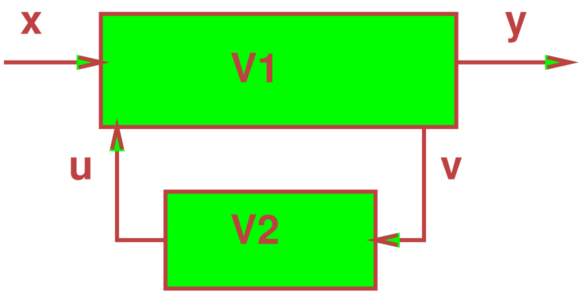
Applications of the E-MACHINE

Supervisory Control: Construct a controller C which controls a plant P so that it has a given behavior.



$$C^*(x, y, u) = \overline{(P \times \overline{S})}$$

Rectification/Engineering Change: Modify an original design so that it conforms to a new specification.

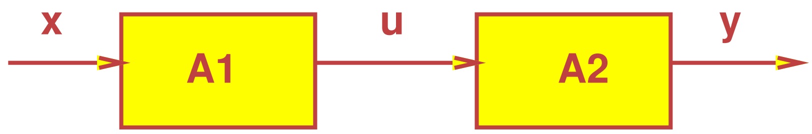


$$V^*(x, v, u, y) = \overline{(V_2 \times \overline{S})}$$

Note: These constructions have no quantification, so that the potential for introducing non-determinism and hence a subset construction is absent. If the specification S is deterministic, then the construction is polynomial.

The E-MACHINE-II

Cascade-I:



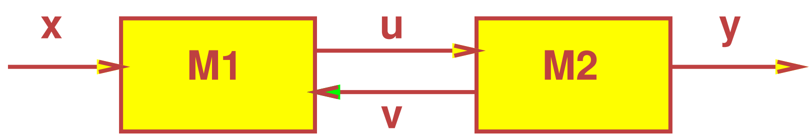
$$\begin{aligned}\phi^{A_1{}^*}(X,U) &= (\forall Y)[\phi^{A_2}(U,Y) \to \phi^{S}(X,Y)]\\ \phi^{A_2{}^*}(U,Y) &= (\forall X)[\phi^{A_1}(X,U) \to \phi^{S}(X,Y)]\end{aligned}$$

Here, $S(x,y) = (\exists u)[A1(x,u) \times A2(u,y)]$. In this case, the machine A_2^* produced by the above construction is exactly that produced by the Kim and Newborn construction.

Cascade-II:

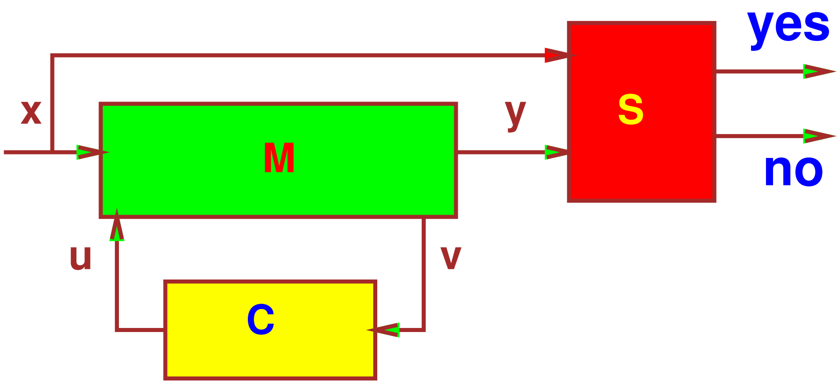
$$\phi^{B_2{}^*}(U,V) = (\forall X,Y)\,[\phi^{B_1}(X,U) \wedge \phi^{B_3}(V,Y) \to \phi^{S}(X,Y)]$$

2-way Cascade:



$$\phi^{M_1{}^*}(X,V,U) = (\forall Y)[\phi^{M_2}(V,U,Y) \to \phi^{S}(X,Y)]$$
$$\phi^{M_2{}^*}(U,V,Y) = (\forall X)[\phi^{M_1}(X,V,U) \to \phi^{S}(X,Y)]$$

E-Machine Approximations



Definition 1 *The* strong satisfiability don't care set *for C is defined by the following formula:*

$$\phi^{SDC_0^C}(V) = \neg(\exists \tilde{X}, \tilde{U}, \tilde{Y})\, \phi^M(\tilde{X}, \tilde{U}, V, \tilde{Y})$$

It is precisely the set of sequences over v which can **never be generated**, no matter what replacement is used for C.

This set gives a certain amount of flexibility in choosing implementations for C; namely any behavior in the machine C_0 defined below is acceptable.

$$\begin{aligned} \phi^{C_0}(V,U) &= \phi^{SDC_0^C}(V) + \phi^C(V,U) \\ &\text{or} \\ C_0(v,u) &= \overline{(\exists x, \tilde{v}, y)[M(x,u,\tilde{v},y)] \times \overline{C}(v,u)} \end{aligned}$$

It can be shown that C_0 is an ISFSM, and also that C_0 does not provide all the flexibility available in optimizing C.

More Approximations

Definition 2 *The* weak satisfiability don't care set *for C is given by the following expression:*

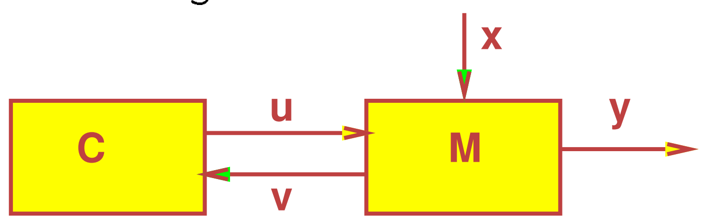
$$\phi^{SDC_1^C}(V) = \neg(\exists \widetilde{X}, \widetilde{U}, \widetilde{Y})\,[\phi^M(\widetilde{X}, \widetilde{U}, V, \widetilde{Y}) \wedge \phi^C(\widetilde{U}, V)]$$

It is precisely the set of sequences over v which can **never be generated in the product machine**, $M \times C$.

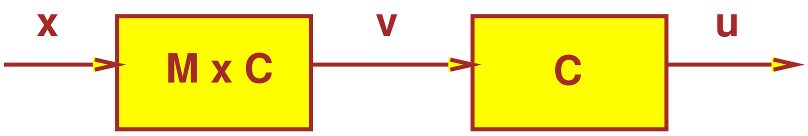
This set gives additional flexibility over SDC_0^C in choosing implementations for C; namely any behavior in the machine C_1 defined below is acceptable.

$$\phi^{C_1}(V,U) = \phi^{SDC_1^C}(V) + \phi^C(V,U)$$

and corresponds to the *input don't care* sequences of Wang when in configuration:



and is the Kim and Newborn flexibility when in configuration:



It can be shown that C_1 is an ISFSM, and also that C_1 does not provide all the flexibility available in optimizing C.

More Approximations

Definition 3 *The* strong observability equivalence relation *for C is given by the following expression:*

$$\phi^{\mathcal{O}_0^C}(V) \;=\; (\forall \widetilde{X}, \widetilde{Y}, \widetilde{U})\,[\phi^M(\widetilde{X}, \widetilde{U}, V, \widetilde{Y}) \rightarrow \phi^{M\times C}(\widetilde{X}, \widetilde{Y})]$$

It is precisely the set of sequences over v for which **any output sequence over u is permissible.**

Clearly, for input sequences which are never generated any output is acceptable, so $\phi^{SDC_1^C}(V) \subseteq \phi^{\mathcal{O}_0^C}(V)$. This set gives additional flexibility over SDC_1^C in choosing implementations for C; namely any behavior in the machine C_2 defined below is acceptable.

$$\phi^{C_2}(V, U) \;=\; \phi^{\mathcal{O}_0^C}(V) + \phi^C(U, V)$$

It can be shown that C_2 is an ISFSM, and that C_2 does not provide all the flexibility available in optimizing C.

It can be proved that

$$\phi^{C_0} \subseteq \phi^{C_1} \subseteq \phi^{C_2}$$

Realizability

The set $\mathcal{L}^{C^*}$ defined by $\phi^{C^*}(U,V)$ is the set of all acceptable controller behaviors. In general, $\mathcal{L}^{C^*}$ may not be realizable. This can happen in two ways.

1. There may be *blocking* input sequences $\widetilde{V}$, i.e. sequences for which there is **no** $\widetilde{U}$ such that $(\widetilde{U},\widetilde{V}) \in \mathcal{L}^{C^*}$.

2. A realization may not exist because of *non-causality*, viz the output may depend on future values of the input. For example, suppose

$$(1,0),(1,1),(1,1),\ldots \in \mathcal{L}$$

but

$$(1,0),(1,1),(0,-),\ldots \notin \mathcal{L}.$$

Thus, at the second step, the output can be 1 only if the next input is a 1.

Definition 4 *We say that C is compatible with C^* ($C \prec C^*$) provided that*

$$(\forall V,U)\,[\phi^{C}(V,U) \to \phi^{C^*}(V,U)]$$

Realizability Problem: *Given a regular $\mathcal{L}^{C^*} \subset (\Sigma_V \times \Sigma_U)^*$, determine the existence of an implementable finite state machine $\widetilde{C}$ such that $\widetilde{C} \prec C^*$.*

This can be solved by calling the following algorithm.

Realizability Algorithm

Mealy:

Realizable_States: (*DFA* **D:** $(S_D, s_0, \Sigma_V \times \Sigma_U, T_D, A_D)$)
{
$S_C = S_D, T_C = T_D, A_C = A_D$;
  while (TRUE) {
    remove states s from S_C, T_C, A_C such that
      $\neg[(\forall v)(\exists u, t)\,[(s, v, u, t) \in T_C \wedge (t \in A_C)]$
    if (no states were removed)
      break;
  }
return S_C, T_C, A_C;
}

Lemma 1 *C^* is realizable if and only if the set of states S_C remaining after convergence contains the initial state. Further, there exists an implementable machine (completely specified, deterministic), $\widetilde{C}$, $\widetilde{C} \prec C^*$ if and only if $\widetilde{C} \prec C^{\mathsf{Mealy}}$, where C^{Mealy} is C^* restricted to the states S_C returned by the Mealy algorithm.*

Moore Realizability

Sometimes we want to to restrict to Moore machines to ensure that there are no combinational cycles.

Moore:

Realizable_States: (*DFA* **D:** $(S_D, s_0, \Sigma_V \times \Sigma_U, T_D, A_D)$)
$S_C = S_D, T_C = T_D, A_C = A_D$;
  while (TRUE) {
    remove states s from S_C, T_C, A_C such that
      $\neg[(\forall v)(\exists u)(\forall t)[(s,v,u,t) \in T_C \wedge (t \in A_C)]$
    remove transitions s, v, u, t from T_C such that
      $\neg[(s,v,u,t) \in T_C \Longrightarrow (\forall \widetilde{v})(\exists \widetilde{t})(s, \widetilde{v}, u, \widetilde{t}) \in T_C]$
    if (no states or transitions were removed)
      break;
  }
return S_C, T_C, A_C;

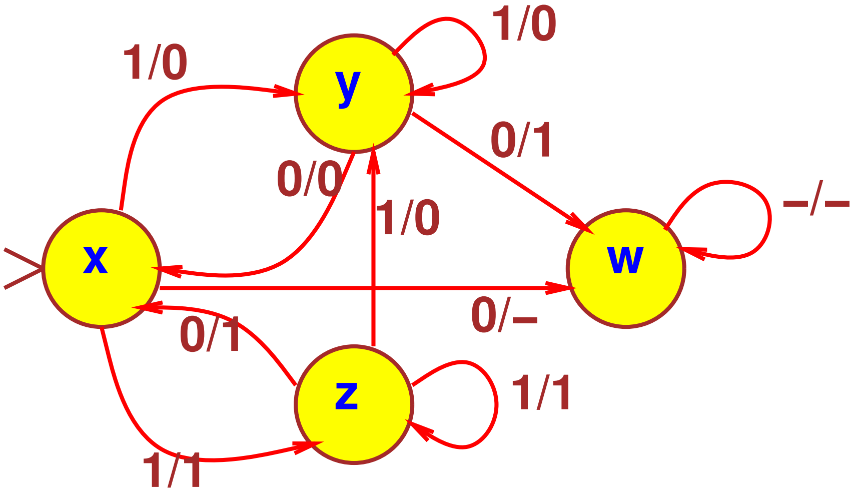
Lemma 2 *C^* is realizable as a Moore machine if and only if the set of states S_C remaining after the above Moore algorithm converges contains the initial state. Further, there exists an implementable Moore machine (completely specified, deterministic), $\widetilde{C}$, $\widetilde{C} \prec C^*$ if and only if $\widetilde{C} \prec C^{\text{Moore}}$, where C^{Moore} is C^* restricted to the states S_C returned by the Moore algorithm.*

However, even though there may not exist a Moore machine, there may exist a Mealy machine $D \prec C^{\text{Mealy}}$ i.e. not all behaviors contained in C^{Mealy} are Moore.

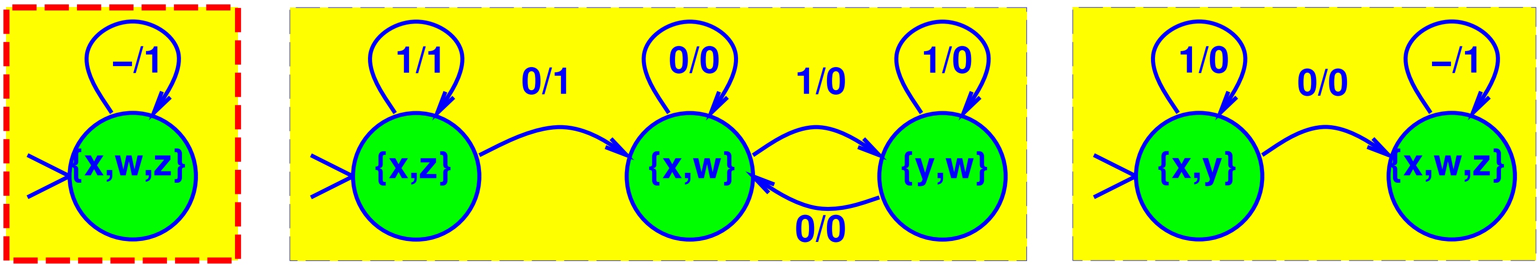
Problem 2 - Minimizing the node $M1$

Contained Behavior:

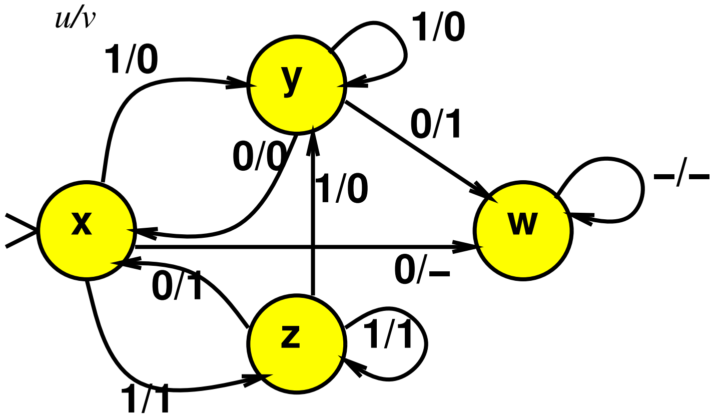
E-Machine:



Some contained behaviors:(deterministic FSM's) contained in this:



E-machine is Pseudo Nondeterministic

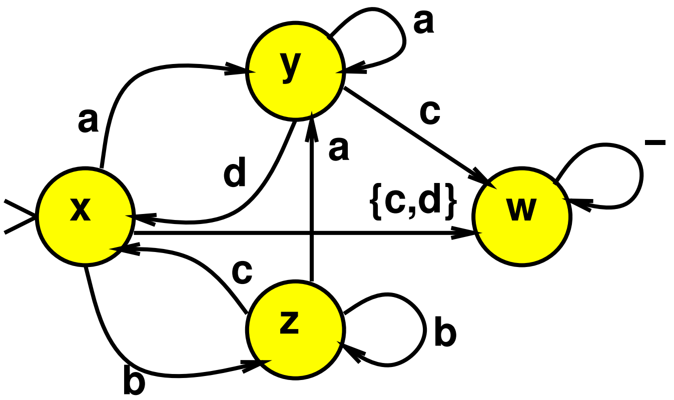


E–machine for M_1

u is the input
v is the output

Definition 5 $E = (U, V, S, T, r)$ *is a* **pseudo NDFSM** *(PNDFSM) if* $\forall(s_p, u, v)$ *there exists at most one* $s_n \in S$ *such that* $T(s_p, u, v, s_n) = 1$*, i.e.* A^E *is deterministic.*

Thus when viewed as an automaton, a PNDFSM is deterministic:



a = 1/0
b = 1/1
c = 0/1
d = 0/0

Deterministic Finite Automaton

(incompletely specified)

Node Minimization

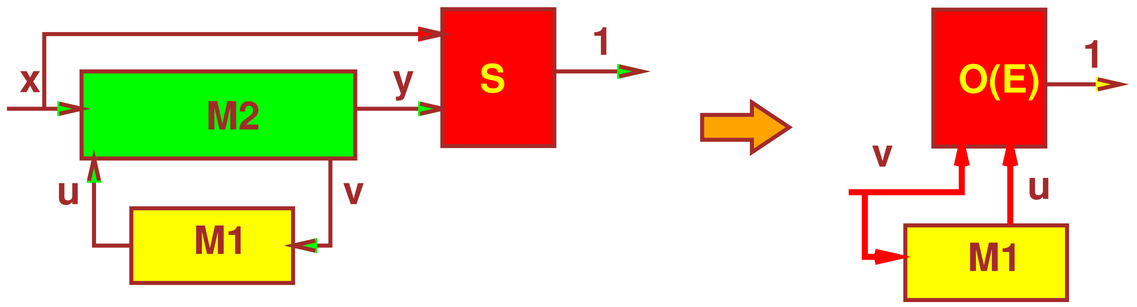
One way to minimize a node in an FSM network is to "state minimize" the E-machine.

Can pose problem of finding a **minimum state machine contained in E-machine**.

"State Minimization" Methods:
Exact and heuristic methods exist for this. These methods use the notion of compatible sets of states generalized from incompletely specified machines to PSNDFSM.

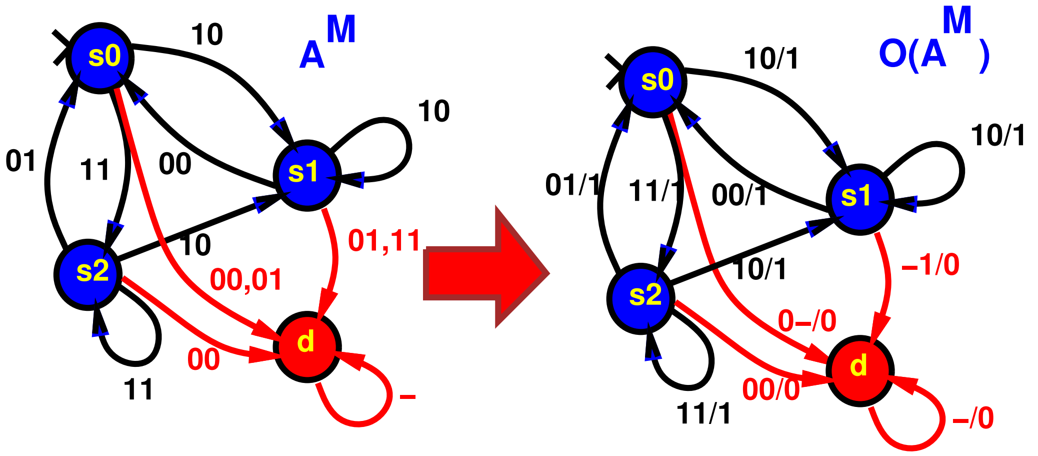
- Watanabe - 1994 (exact and heuristic)
- Damiani - 1994 (exact)
- Kam and Villa - 1995 (exact method using implicit computation of compatibles).

FSM Node Minimization via ATPG



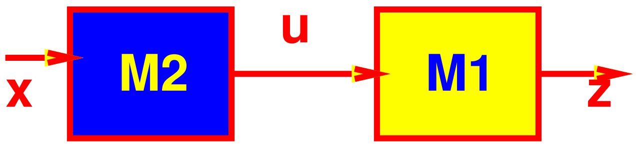
where $\mathcal{O}(E)$ is the **observability FSM** of the E-machine E.

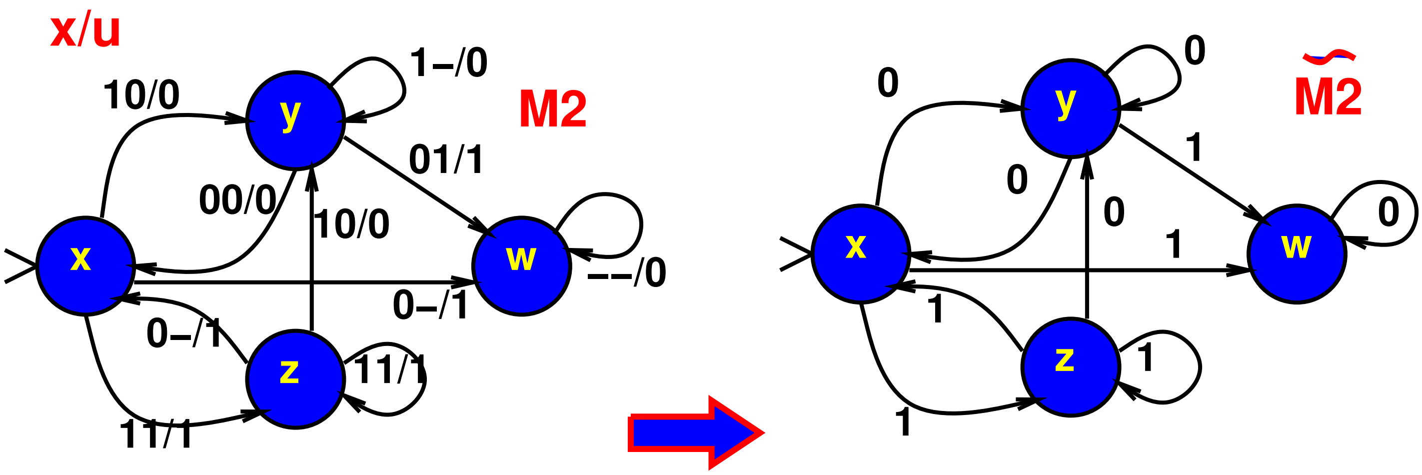
$\mathcal{O}(E)$ is obtained from E by converting each i/o pair to an input io, completing it by adding a dummy state d, and one output which is 0 if i/o is a transition to d and 1 otherwise.



Since $\mathcal{O}(E)$ is a completely specified FSM, we can implement it as a netlist and use **normal sequential redundancy removal** on $M1$. [H.Y. Wang - 1995].

Using Sequential Input Don't Cares





Before sequential ATPG is used we can transform the driving machine $M2$ by removing its inputs $\widetilde{M2}$, simplifying and input determinizing. Then sequential redundancy removal is used on the $M1$ part of the composed machine.

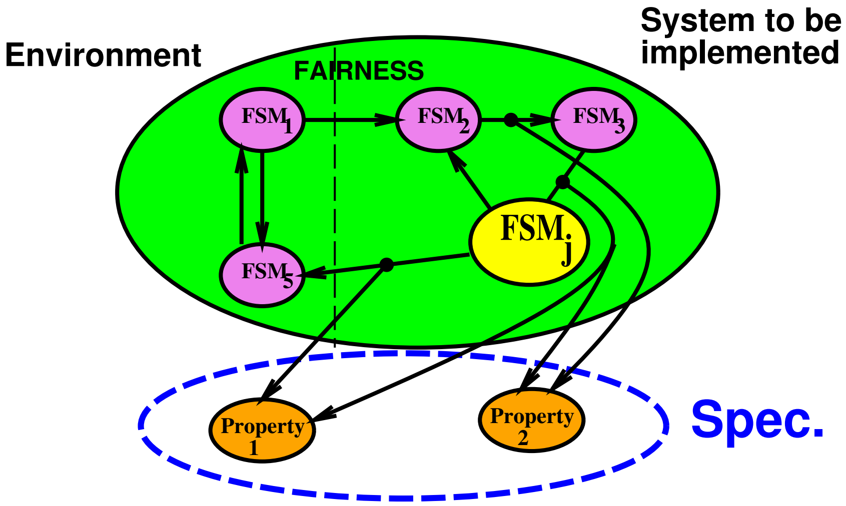
Summary - Sequential Synthesis

Node Minimization

- know how to derive complete set of permissible behaviors (E-machine)
- E-machine is a complex computation (possibly doubly exponential) but some topologies are easier.
- have methods for approximating E-machine.
- E-machine is pseudo-nondeterministic, but some approximations are ISFSM's
- exact and heuristic state minimizers for PNDFSM's
- use of sequential ATPG for minimizing given network relative to E-machine
- limited right now to small machines
- need more examples and experiments on use of approximate methods.

Design Verification and Refinement

Problem: We want to verify that a given design satisfies a set of desired properties.



Computational Model

- Network of NDFSM's with **fairness constraints**. This models
    - system to be synthesized
    - its environment
- Properties - expressed by either
    - ω-**automata** or
    - computational tree logic (CTL) with fairness constraints.

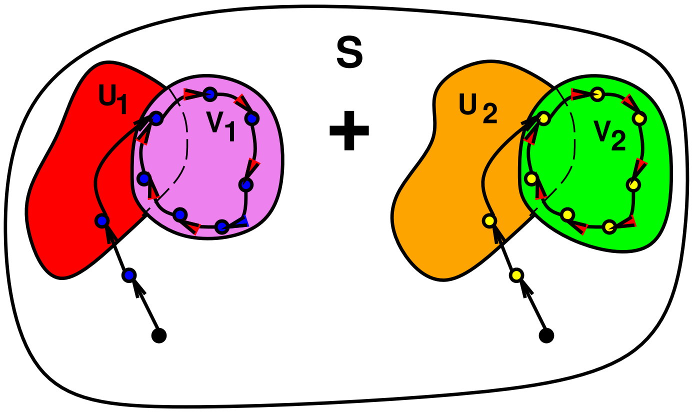
ω-Automata and Fairness Constraints

Rabin : fair (acceptable) path $r = (s_0, s_1, s_2, \ldots)$ is

$$\sum_i (F^\infty(U_i) \wedge G^\infty(V_i))$$

where

$$F^\infty(U_i) \Rightarrow \text{inf}(r) \cap U_i \neq \phi \quad G^\infty(V_i) \Rightarrow \text{inf}(r) \subseteq V_i.$$



Rabin Automaton

With Rabin automata, one can express properties like

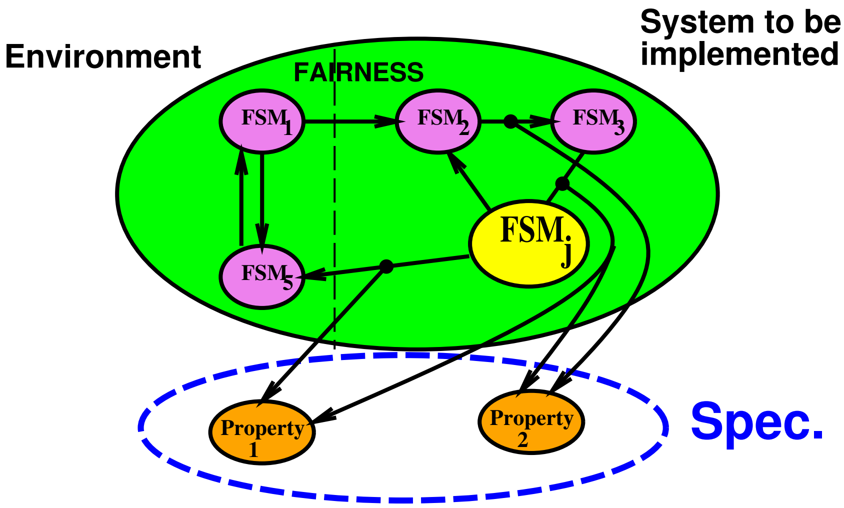
> infinitely often each time C requests service, eventually it gets access to service

- a liveness property.

A **Street** automaton is like a Rabin but with complementary fairness constraints:

$$\prod_i (F^\infty(U_i) \vee G^\infty(V_i))$$

Verification and Refinement

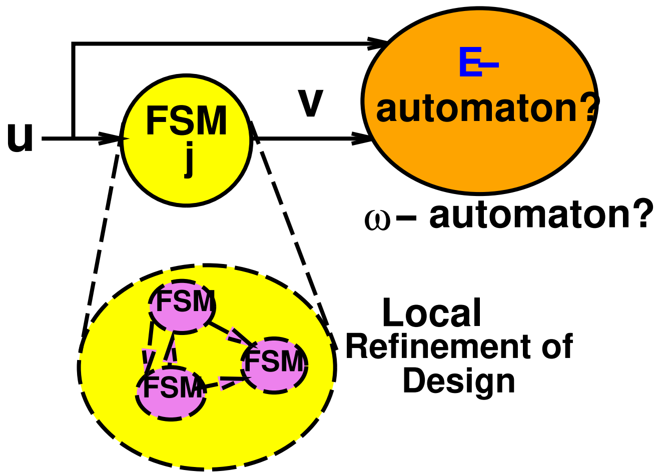


Now the set of properties $\{P_1, P_2, \ldots, P_n\}$ acts like our system specification.

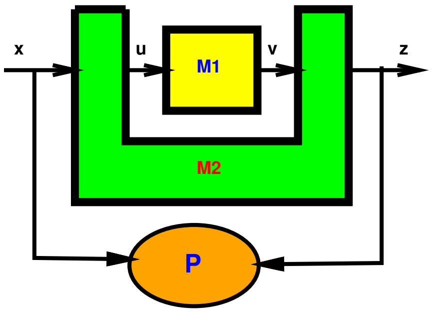
Problems:

1. Summarize complete set of behaviors allowed locally at a single node, FSM_j.

2. Refine (synthesize) a suitable implementation of FSM_j (if one exists).



Generalized E-machine for Design Refinement



P is desired property expressed as an ω-automaton. $M2$ is the rest of the system.

Restated as S1S Expression on Infinite Strings:

Theorem 1 *(Buchi (1960)) A ω-language is expressible as an* S1S *expression if and only if it is ω-regular.*

Thus

$$E(u,v) = \forall(x,z)(M2(x,v,u,z) \Rightarrow P(x,z))$$

is an expression in S1S.

Thus operationally,

$$E(u,v) = \overline{\exists(x,z)(M2 \times \overline{P})}$$

exists. (*It turns out that $E(u,v)$ is a Rabin automaton.*)

Generalized E-machine Construction

$$E(u,v) = \overline{\exists(x,z)(M2 \times \overline{P})}$$

1. Complement P. Generally, can express properties as deterministic Rabin automata (DR), so complementation into a deterministic Streett automata (DS), is easy.

2. Form $M2 \times \overline{P}$. Easy since if $M2$ is Streett.

3. Form $Q = \exists(x,z)(M2 \times \overline{P})$. Easy, but Q generally becomes NS. (*Hiding variables generally causes nondeterminism*).

4. Complement Q. We can do this by

   $$NS \longrightarrow DR.$$

   The complexity is:

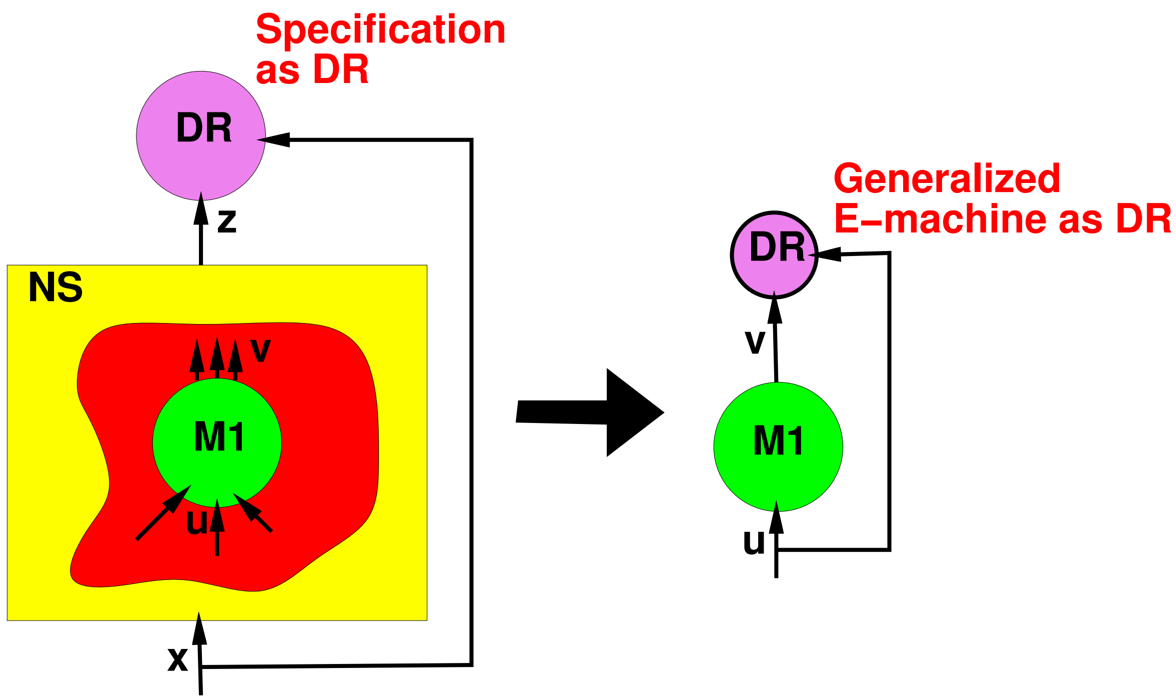
   $$NS(n,f) \longrightarrow DR(2^{n2^f \log(n2^f)})$$

   where n is the number of states and f the number of fairness constraints (Saffra - 1988).

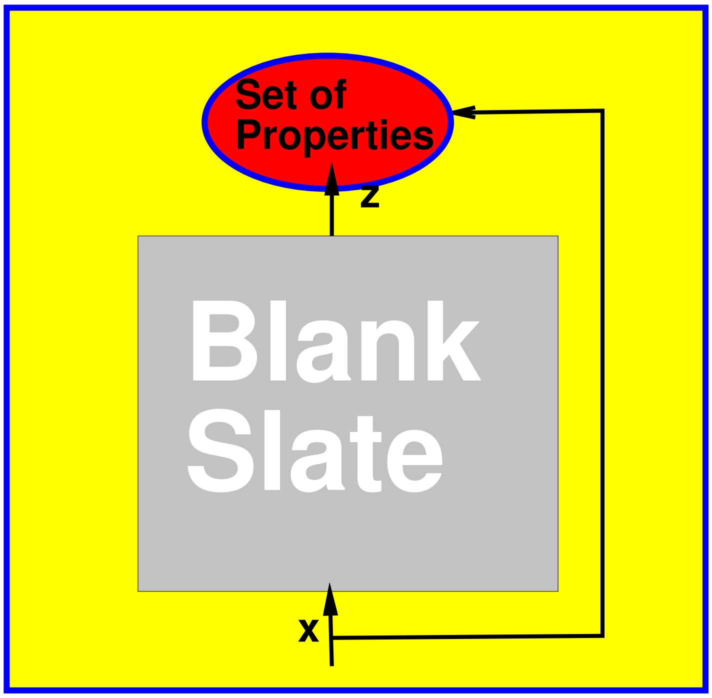
We have recently implemented this complementation method (Tasiran & Hojati).

Summary - Design Refinement

In theory, we have a solution to the problem of finding all permissible behaviors when our initial specification is a set of properties. (*The construction is hierarchical*).



The Ultimate Synthesis Problem



Open Questions:

- how to find a contained behavior, i.e. a completely specified deterministic FSM.
- how to find a minimum contained behavior

Conclusions

- Common theme for synthesis of combinational, sequential, and ω-automata for the node minimization problem.

- Problems of synthesis and verification are closely related in all three areas.

- Common model - network of automata