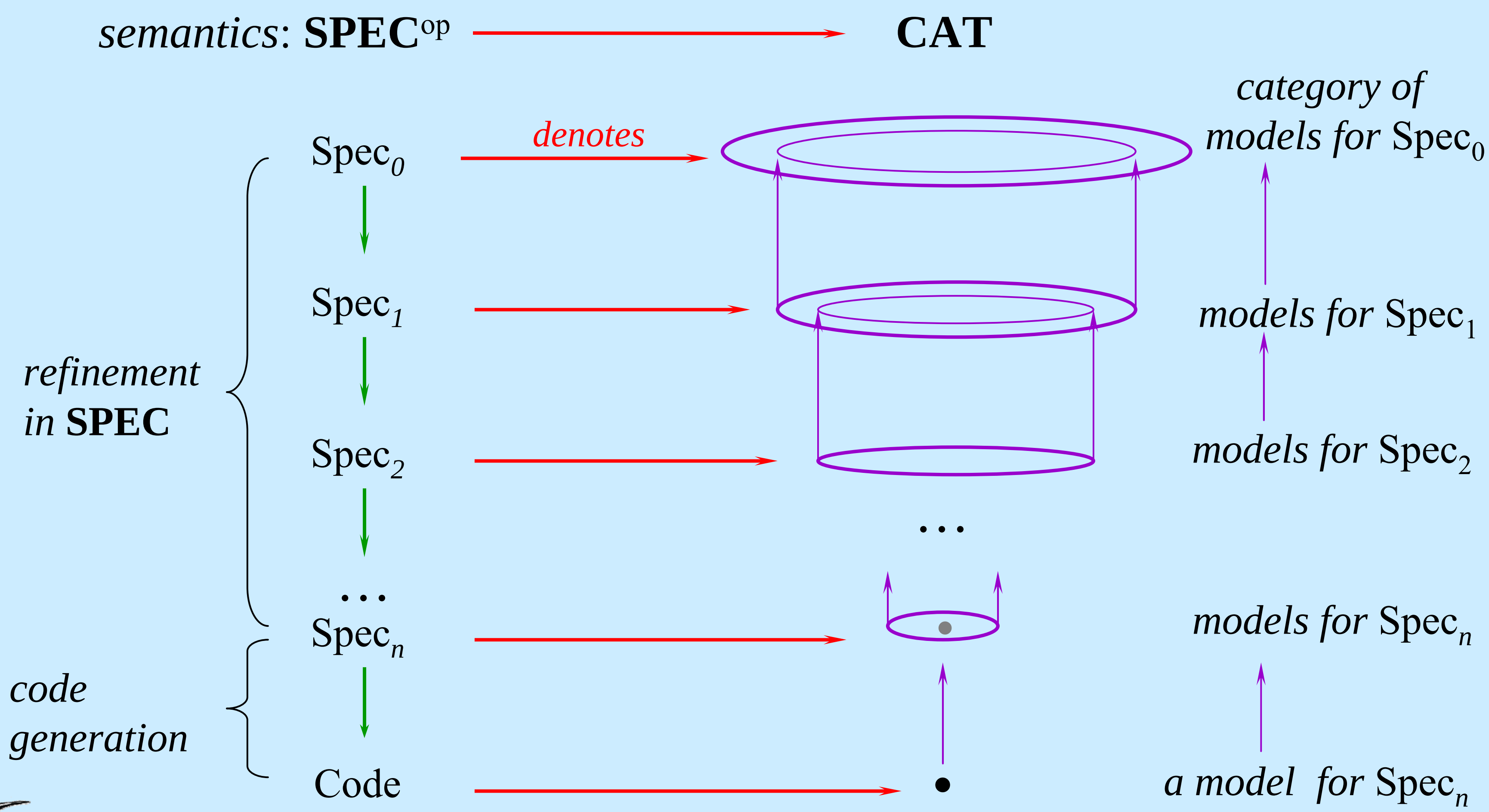
Requirement Enforcement by Transformation Automata

Douglas R. Smith

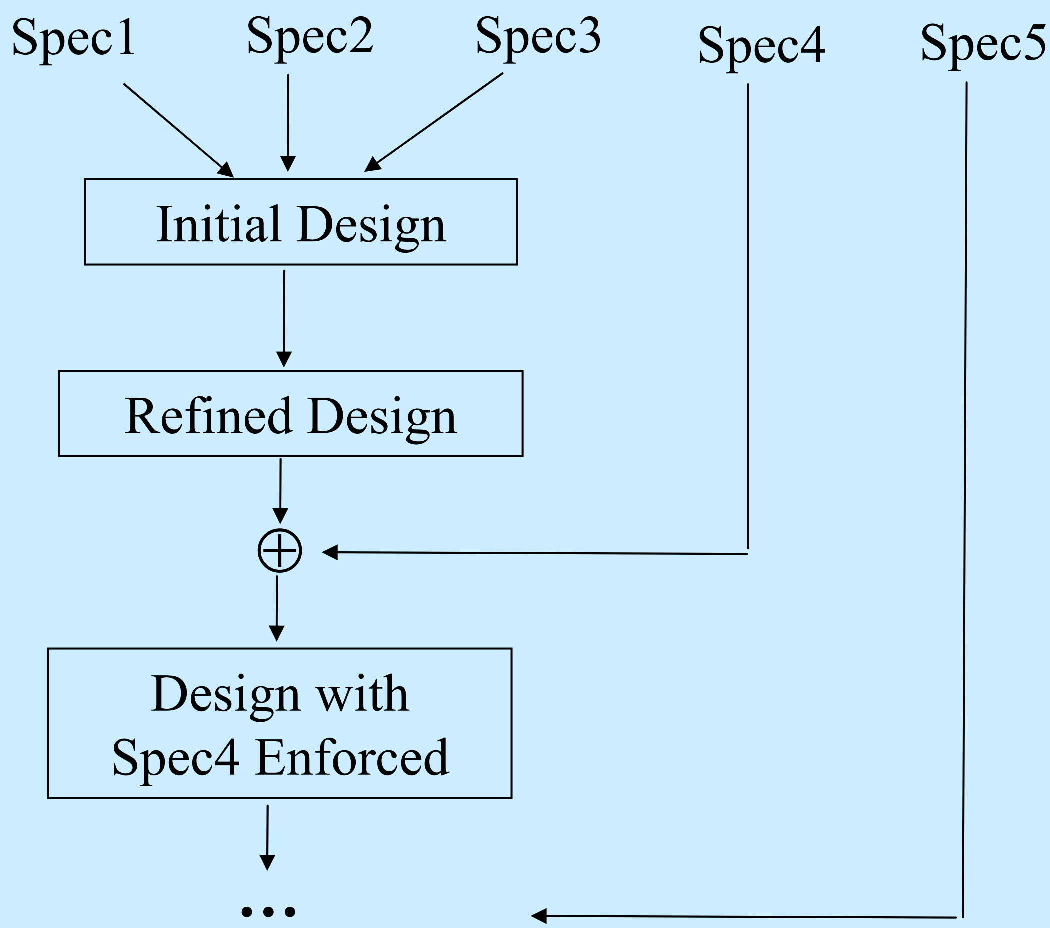
Principal Scientist
Kestrel Institute
Palo Alto, California

Exec. V.P. and CTO
Kestrel Technology LLC
Los Altos, California

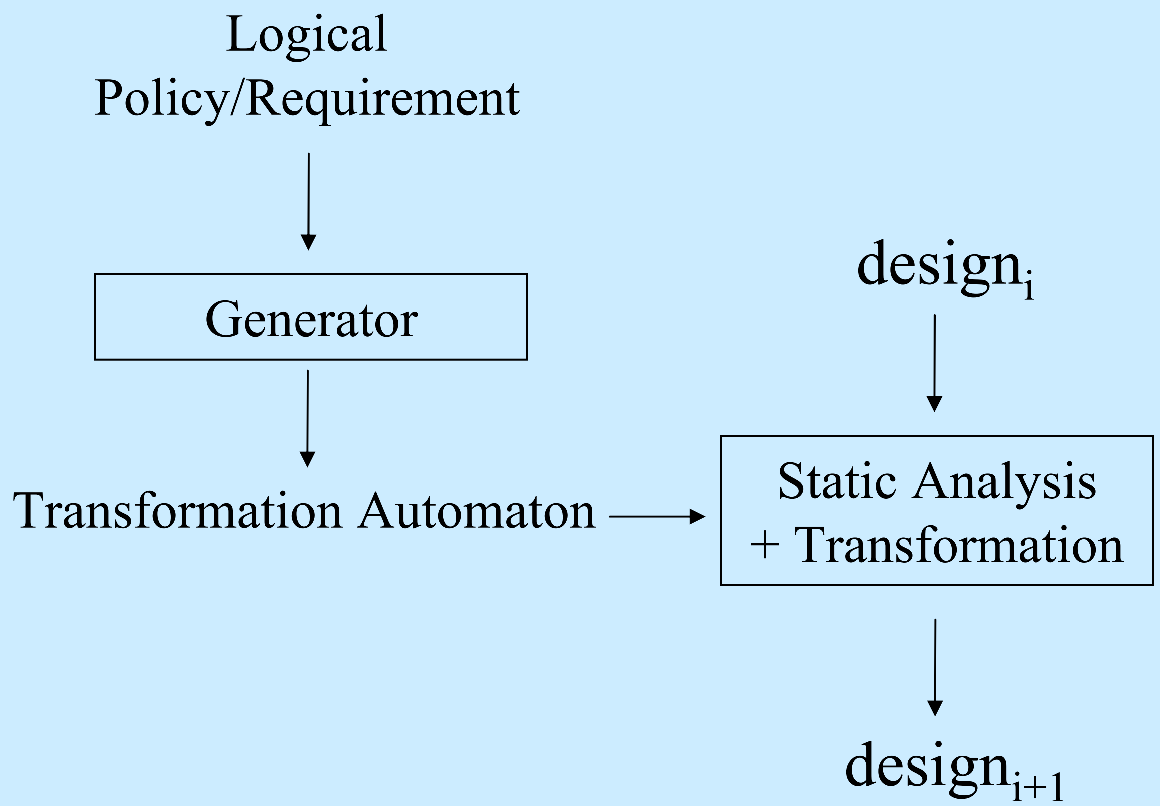
Software Development by Refinement



Requirement Specifications to Code

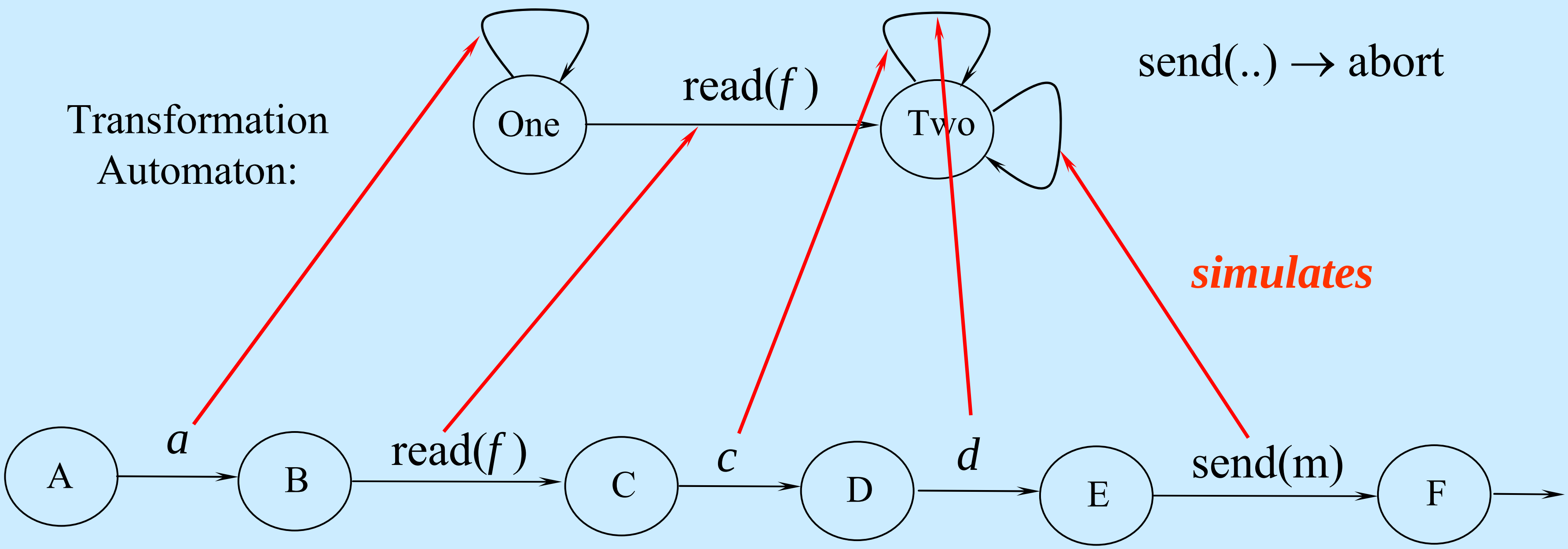


Enforcing a Policy



Enforce a Security Policy

Policy: No send actions allowed after file *f* is read



Build simulation map,
then generate new code for corresponding actions

send(m) action
is replaced
by abort action

Transformation Automata

code transformation
with guards on state and code bindings

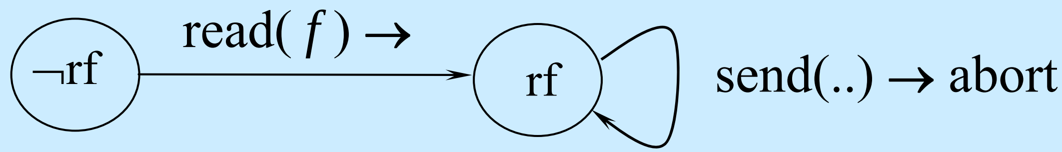
Policy State 1 —— guard & $\Phi(\alpha) \vdash$ \`α´ $\rightarrow [\textit{pre}]\{\textit{pattern}\}[\textit{post}]$ ——→ Policy State 2

if α matches the current action
  and guard holds
  and $\Phi(\alpha)$ holds
then replace α by an instance of *pattern*
    that satisfies the given precondition and postcondition

Simple Information Flow Policy

Policy: No send actions allowed after file *f* is read

policy automaton:




```
policy MumAfterRead {
  boolean rf
   init       →  rf := false
  read( f ) →  read( f ) || rf := true
  send(..)  →  abort    if rf
}
```

Transformation Automaton

Abbreviation: let

Policy State 1 —— $cg \vdash `\alpha' \rightarrow$ **achieve R** ——→ Policy State 2

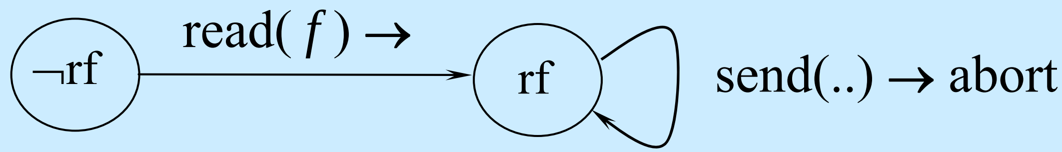
denote

Policy State 1 —— $cg \vdash `\alpha' \rightarrow \mathbf{[pre_{\alpha}]\{\}[post_{\alpha} \wedge R\ ]}$ ——→ Policy State 2

Simple Information Flow Policy

Policy: No send actions allowed after file *f* is read

policy automaton:



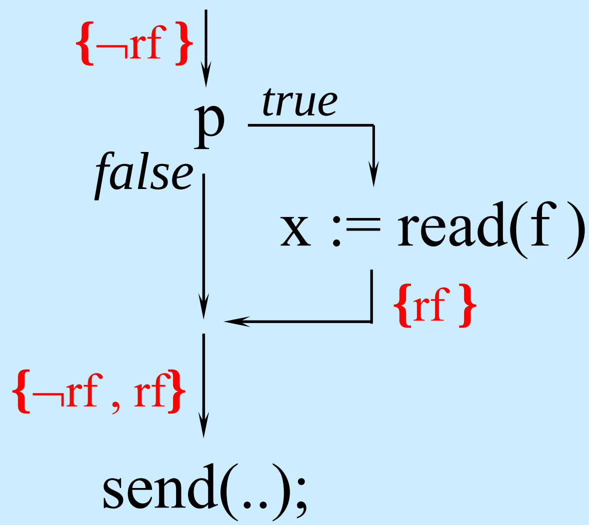

```
policy MumAfterRead {
  boolean rf
   init       →  achieve rf´ = false
  read( f ) →  achieve rf´ = true
  send(..)  →  abort                     if  rf
}
```

Example

Design fragment

```
{ …
  if  (p) { x := read(f ) }
  send(..);
  …
}
```

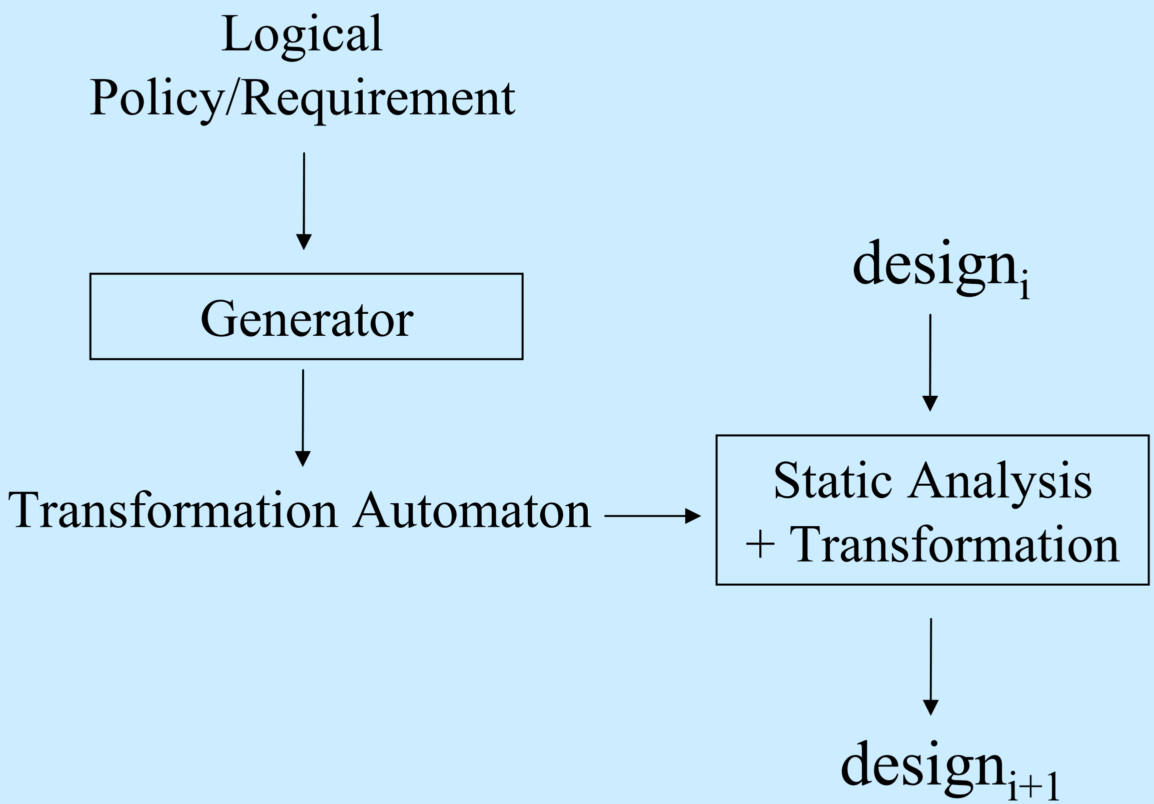
Control-Flow Graph with results of static analysis



Transformed Design fragment

```
{ rf := false;  …
  if  (p) { x := read(f ); rf := true; }
  if (rf ) {abort}
  else {send(..); …}
  …
}
```

Enforcing a Policy



Expressing System Constraints

Many systems constraints refer to

- history (events, actions, state,…)
- dynamic context (e.g. the call-stack, heap)
- environment behavior
- substrate properties (e.g. instruction timing, latence, …)
- heap
- agency

Reified Variables

key idea: extend state with a virtual history variable

$$\mathbf{s_0} \xrightarrow[\textit{hist} := \langle s_0,\, act_0,\, s_1\rangle]{act_0} \mathbf{s_1} \xrightarrow[\textit{hist} := \textit{hist} :: \langle s_1,\, act_1,\, s_2\rangle]{act_1} \mathbf{s_2} \xrightarrow[\textit{hist} := \textit{hist} :: \langle s_2,\, act_2,\, s_3\rangle]{act_2} \mathbf{s_3} \cdots$$

Reified variables
- exist for purposes of specification
- sliced away prior to code generation

let *actions*⋆*hist* denote the sequence of actions in *hist*

Policy: Save data after every 5 changes

Invariants: $\square\ cnt = (length \cdot dataOp \triangleright actions \star hist)\ mod\ 5$

$\square\ cnt = 0 \ \Rightarrow data = file$

where *dataOp*(*act*) iff *act* changes the data set of interest.

Disruptive Actions: derivable as a necessary condition on disruption of the invariant: $I(x) \neq I(x')$

Calculating a Pointcut Specification

Disruptive Actions: necessary condition on $I(x) \neq I(x')$

Assume: $cnt = (length \cdot dataOp \triangleright actions \star hist)\ mod\ 5$

$\wedge\ hist' = hist :: \langle s, act, s' \rangle$

$\wedge\ cnt = cnt'$

Simplify: $(cnt = (length \cdot dataOp \triangleright actions \star hist)\ \ mod\ 5\ )$

$\neq (cnt' = (length \cdot dataOp \triangleright actions \star hist')\ mod\ 5\ )$

$\equiv (length \cdot dataOp \triangleright actions \star hist)\ \ mod\ 5\ \neq\ (length \cdot dataOp \triangleright actions \star hist')\ mod\ 5$

$\equiv \ldots \neq (length \cdot dataOp \triangleright actions \star hist :: \langle s, act, s' \rangle)\ mod\ 5$

$\equiv \ldots \neq (length \cdot dataOp \triangleright actions(hist) :: act)\ mod\ 5$

$\equiv \ldots \neq if\ \ \neg dataOp(act)$

$then\ (length \cdot dataOp \triangleright actions \star hist))\ mod\ 5$

$else\ (length \cdot dataOp \triangleright actions \star hist) :: act)\ mod\ 5$

$\equiv\ \ if\ \ \neg dataOp(act)\ \ then\ false\ \ else\ true$

$\equiv\ \ dataOp(act)$

Calculating Maintenance Code

Spec for Maintenance Code: for each data-changing action *act*,

Assume: $cnt = (length \cdot dataOp \triangleright actions \star hist)\ mod\ 5$
$\wedge\ hist' = hist :: \langle s, act, s' \rangle$
$\wedge\ dataOp(act)$

Achieve: $cnt' = (length \cdot dataOp \triangleright actions \star hist')\ mod\ 5$

$$= (length \cdot dataOp \triangleright actions \star (hist :: \langle s, act, s' \rangle))\ mod\ 5$$
$$= (length \cdot dataOp \triangleright (actions \star hist) :: act)\ mod\ 5$$
$$= (length \cdot (dataOp \triangleright (actions \star hist)) :: act)\ mod\ 5$$
$$= length \cdot (dataOp \triangleright (actions \star hist)) + 1\ mod\ 5$$
$$= cnt + 1\ mod\ 5$$

General Case

Invariant: $I(x)$

Disruptive Actions: necessary condition on $I(x) \neq I(x')$

Spec for Maintenance Code :

for each such action *act* with specification

Assume: $P(x)$

Achieve: $Q(x, x')$

generate and satisfy new specification

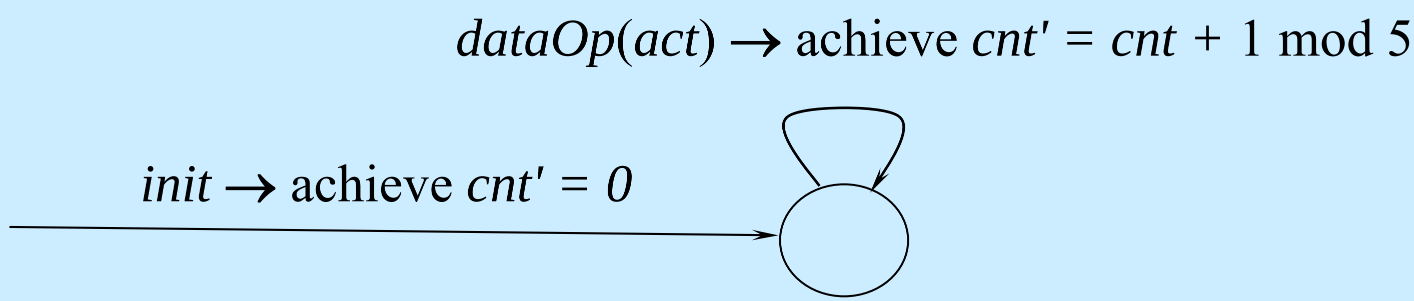
Assume: $P(x) \wedge I(x)$

Achieve: $Q(x, x') \wedge I(x')$

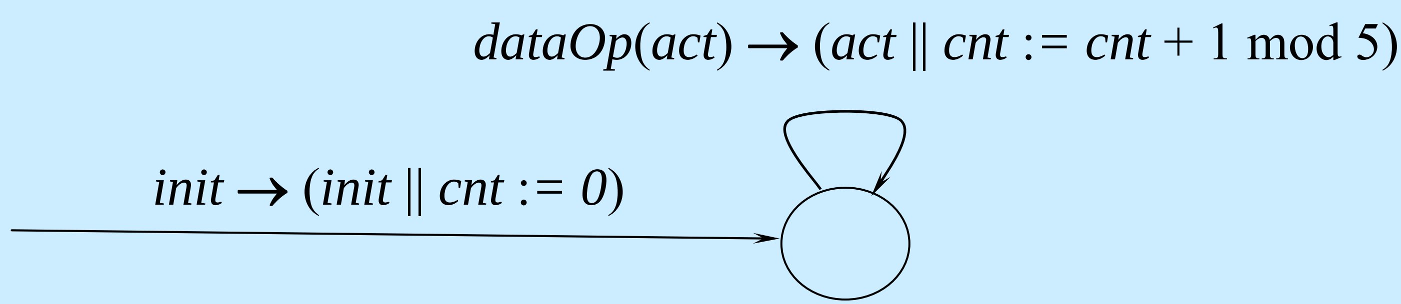
spec typically satisfied by code of the form: $act \,||\, update$

Optimized Transformation Automaton

to establish: $\Box\ cnt = (length \cdot dataOp \triangleright actions \star hist)\ mod\ 5$



After carrying out the syntheses:



Policy: Save data after every 5 changes

Invariant: $\Box\ cnt = 0 \Rightarrow data = file$

Disruptive Actions: derivable as a necessary condition
on disruption of the invariant: $I(x) \neq I(x')$

Calculating a Pointcut Specification

Disruptive Actions: necessary condition on $I(x) \neq I(x')$

Assume: $cnt = 0 \Rightarrow data = file$

$\wedge\ dataOp(act)$

$\wedge\ hist' = hist :: \langle S, act \rangle$

$\wedge\ cnt = (length \cdot dataOp \triangleright actions \star hist)\ \ mod\ 5$

$\wedge\ cnt' = cnt + 1\ mod\ 5$

Simplify: $\neg\,(cnt' = 0 \Rightarrow data' = file')$

$\equiv\ cnt' = 0\ \wedge\ data' \neq file'$

$\equiv\ cnt + 1\ mod\ 5 = 0$

$\equiv\ cnt = 4$

Calculating Maintenance Code

Spec for Maintenance Code: for each data-changing action *act*,

Assume: $cnt = 0 \Rightarrow data = file$

$\wedge\ dataOp(act)$

$\wedge\ hist' = hist :: \langle S, act \rangle$

$\wedge\ cnt = 4$

$\wedge\ cnt' = cnt + 1 \bmod 5$

Achieve: $cnt' = 0 \Rightarrow data' = file'$

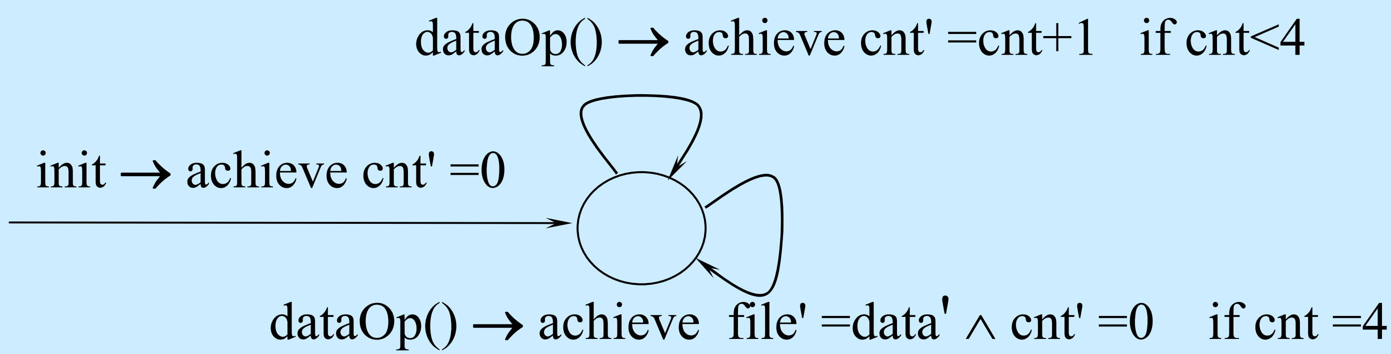
$\equiv\ data' = file'$

This postcondition can be achieved by a *save*() operation

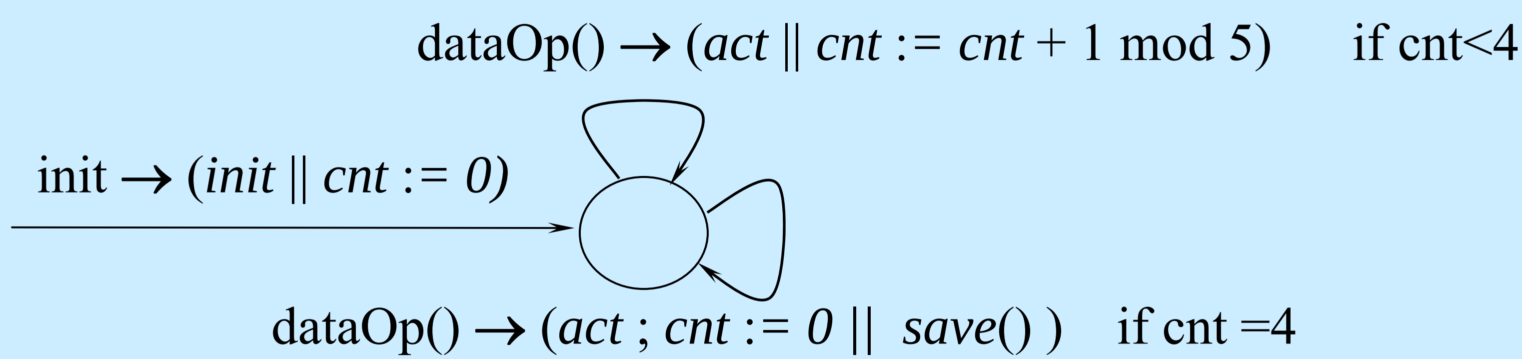
Derived Transformation Automaton

□ *cnt* = (*length* · *dataOp* ▹ *actions* ⋆ *hist*) *mod 5*

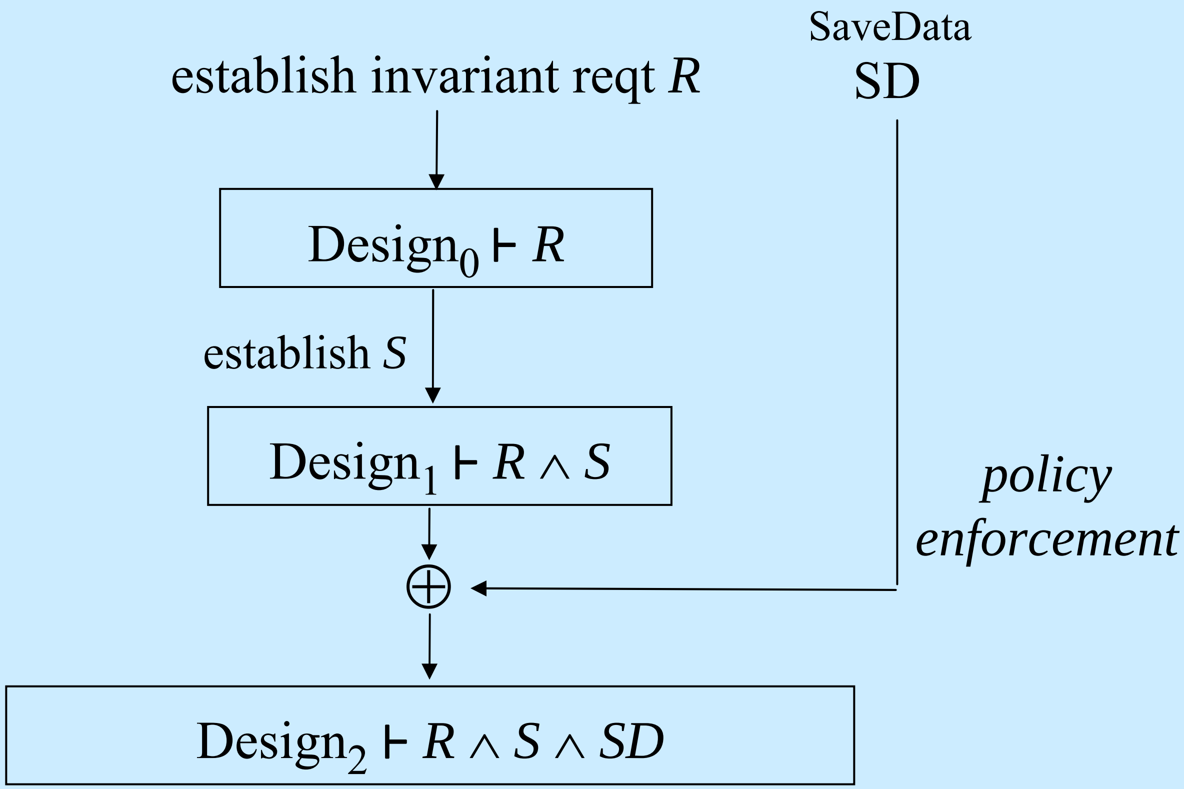
□ *cnt* = *0* ⇒ *data* = *file*



After synthesis:



Refinement


SaveData
SD
establish invariant reqt *R*
$\text{Design}_0 \vdash R$
establish *S*
$\text{Design}_1 \vdash R \wedge S$
policy enforcement
$\oplus$
$\text{Design}_2 \vdash R \wedge S \wedge SD$

Simple Information Flow Policy

Policy: No send actions allowed after file *f* is read

Invariants: $\Box\ rf \Leftrightarrow read(f) \in actions \star hist$

$\Box\ Send(act) \Rightarrow \neg rf$

where *Send(act)* iff *act* is a transmission event

In the following we will skip the derivation of how to maintain

$$\Box\ rf \Leftrightarrow read(f) \in actions \star hist$$

Calculating a Pointcut Specification

Assume: $hist' = hist :: \langle s, act, s' \rangle$

$\qquad \wedge \; rf \Leftrightarrow read(f) \in actions \star hist$

Simplify: $\neg\,(Send(act) \Rightarrow \neg rf)$

$\qquad \equiv Send(act) \wedge rf$

$\qquad \Rightarrow Send(act)$

Calculating Maintenance Code

Spec for Maintenance Code: for each data-changing action *act*,

Assume: $\text{pre}_{act} \wedge hist' = hist :: \langle s, act, s' \rangle$

$\wedge \; rf \Leftrightarrow read(\, f\,) \in actions \star hist$

$\wedge \; Send?(act)$

Achieve: $\text{post}_{act} \wedge (Send(act) \Rightarrow \neg rf\,)$

$= \text{post}_{act} \wedge \neg rf$

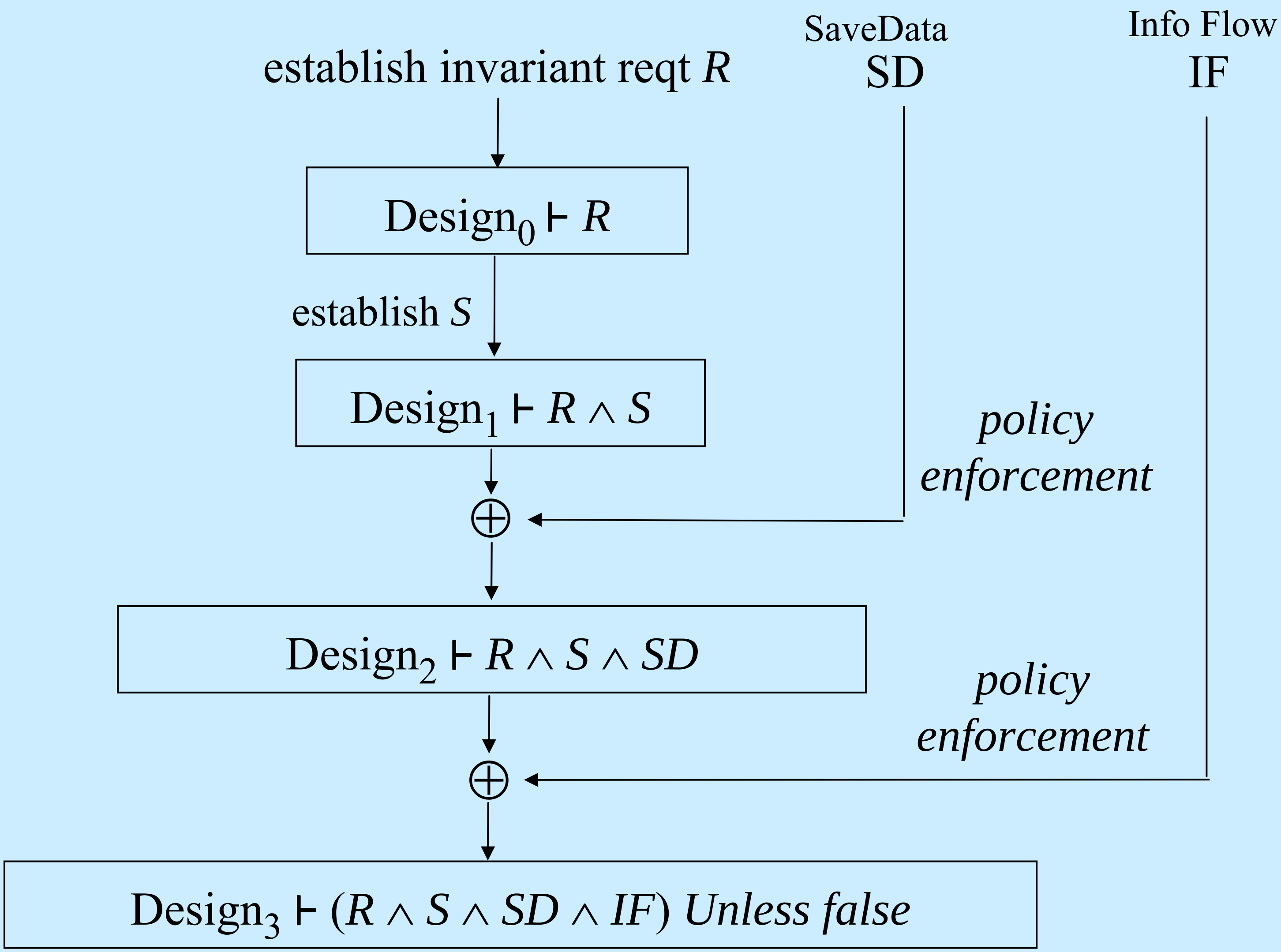
$= \text{if } rf \text{ then post}_{act} \wedge \neg rf$

$\quad \text{else post}_{act} \wedge \neg rf$

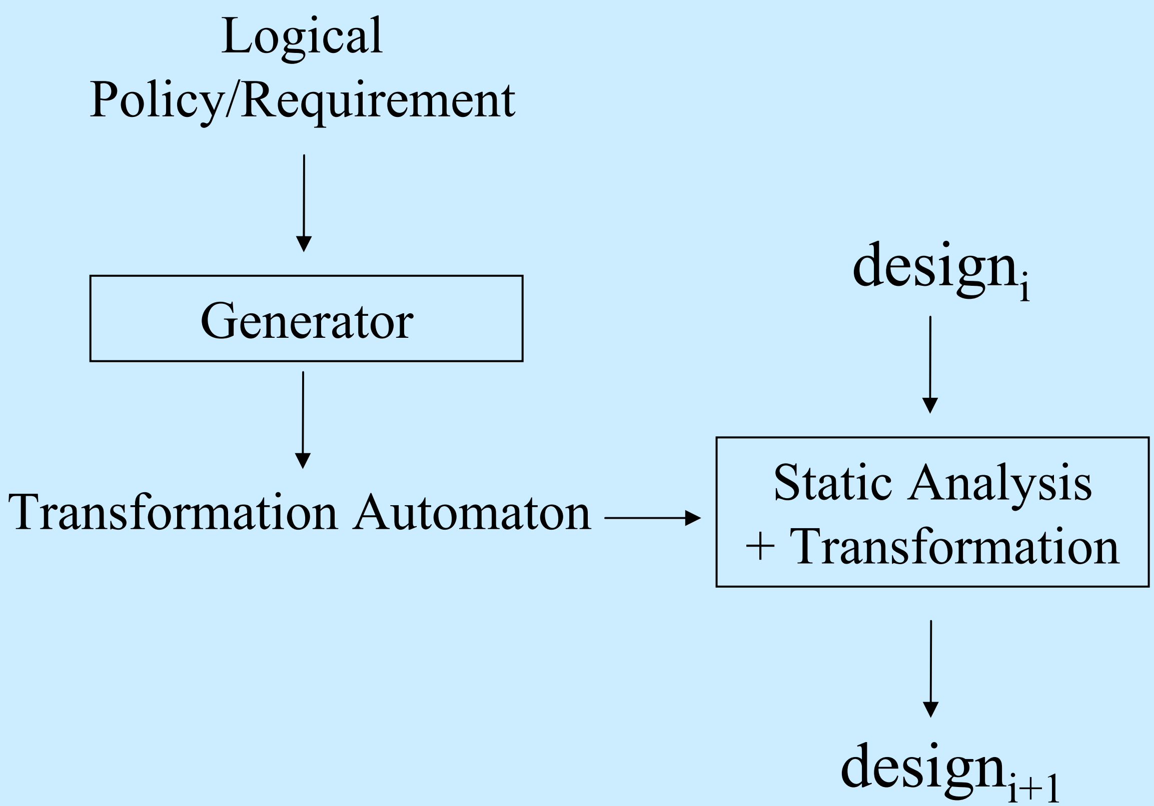
$= \text{if } rf \text{ then false}$

$\quad \text{else post}_{act}$

Generalized Refinement



Enforcing a Policy



Extras

Verification versus Refinement

Given Design_0 and temporal formula Ψ

1. Static Verification: show $\text{Design}_0 \vdash \Psi$

2. Runtime Verification: for input e, show $\text{Design}_0 \| e \models \Psi$

3. Synthesis/Refinement:

$$\text{Design}_0 \vdash \Phi$$

$$\Big\downarrow \textit{transform}$$

$$\text{Design}_1 \vdash \Phi \wedge \Psi$$