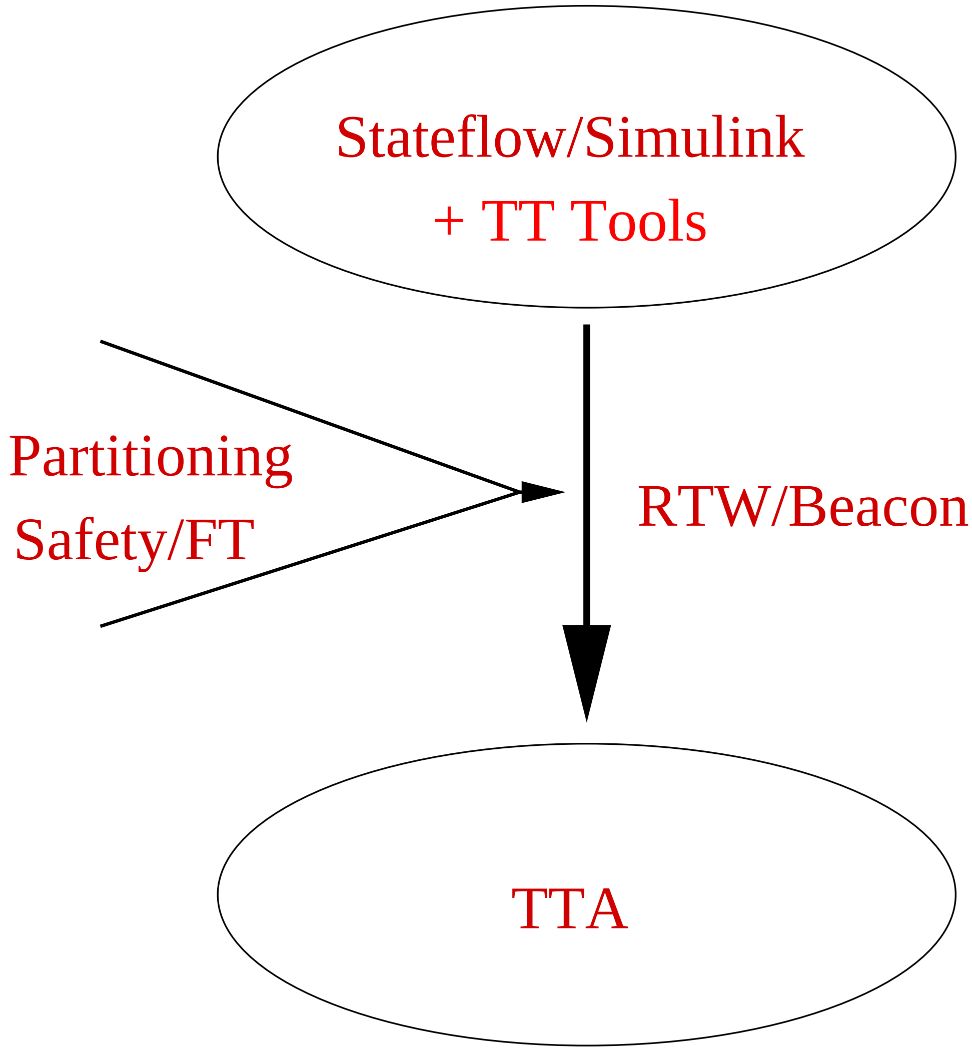
Formal Composition for Time-Triggered Systems

John Rushby and Ashish Tiwari

Rushby,Tiwari@csl.sri.com

Computer Science Laboratory
SRI International
Menlo Park CA 94025

Objective: Specific



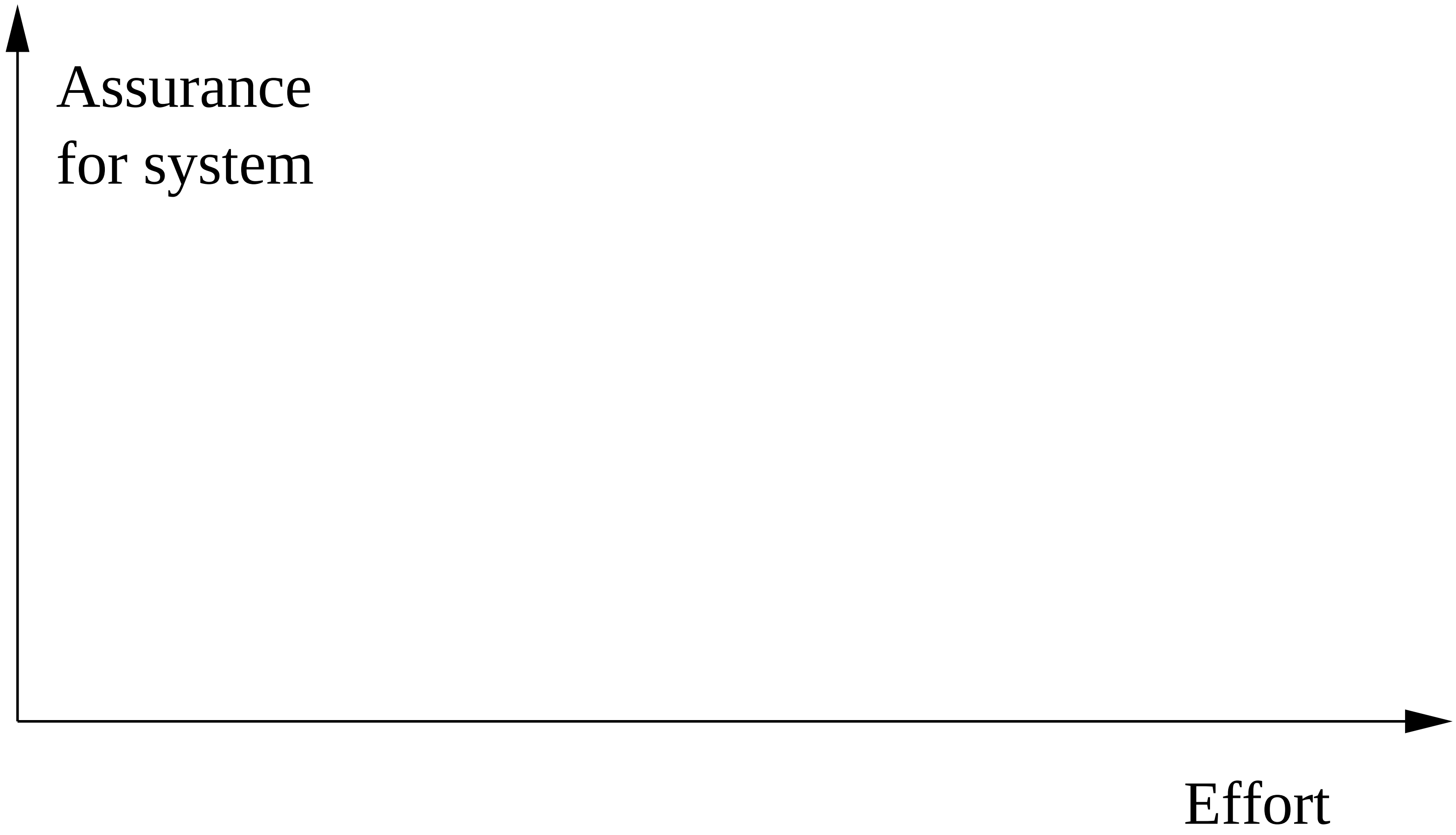
Analyze models

Verify transformations

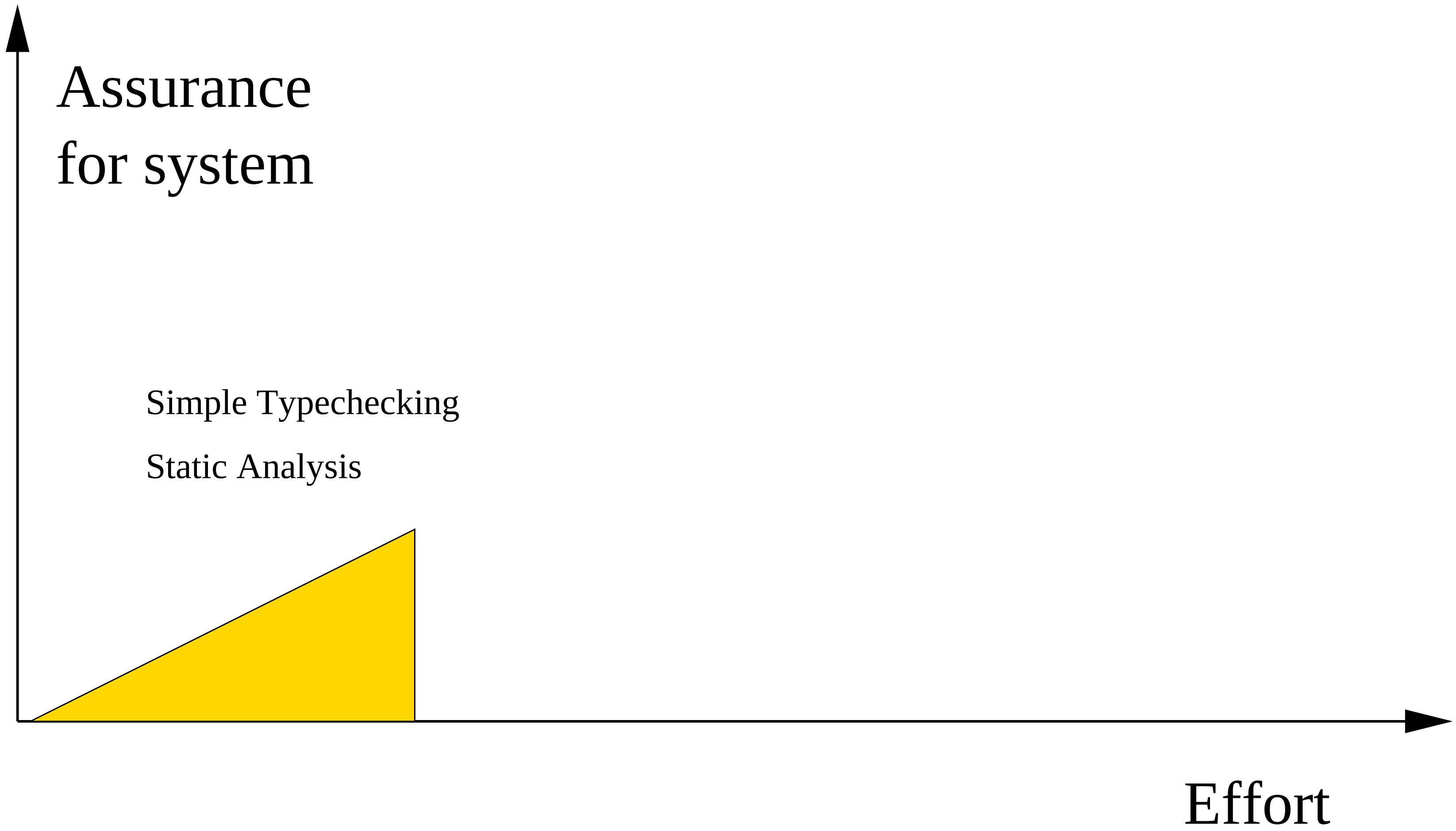
Verify TT services

To yield assurance
for final system

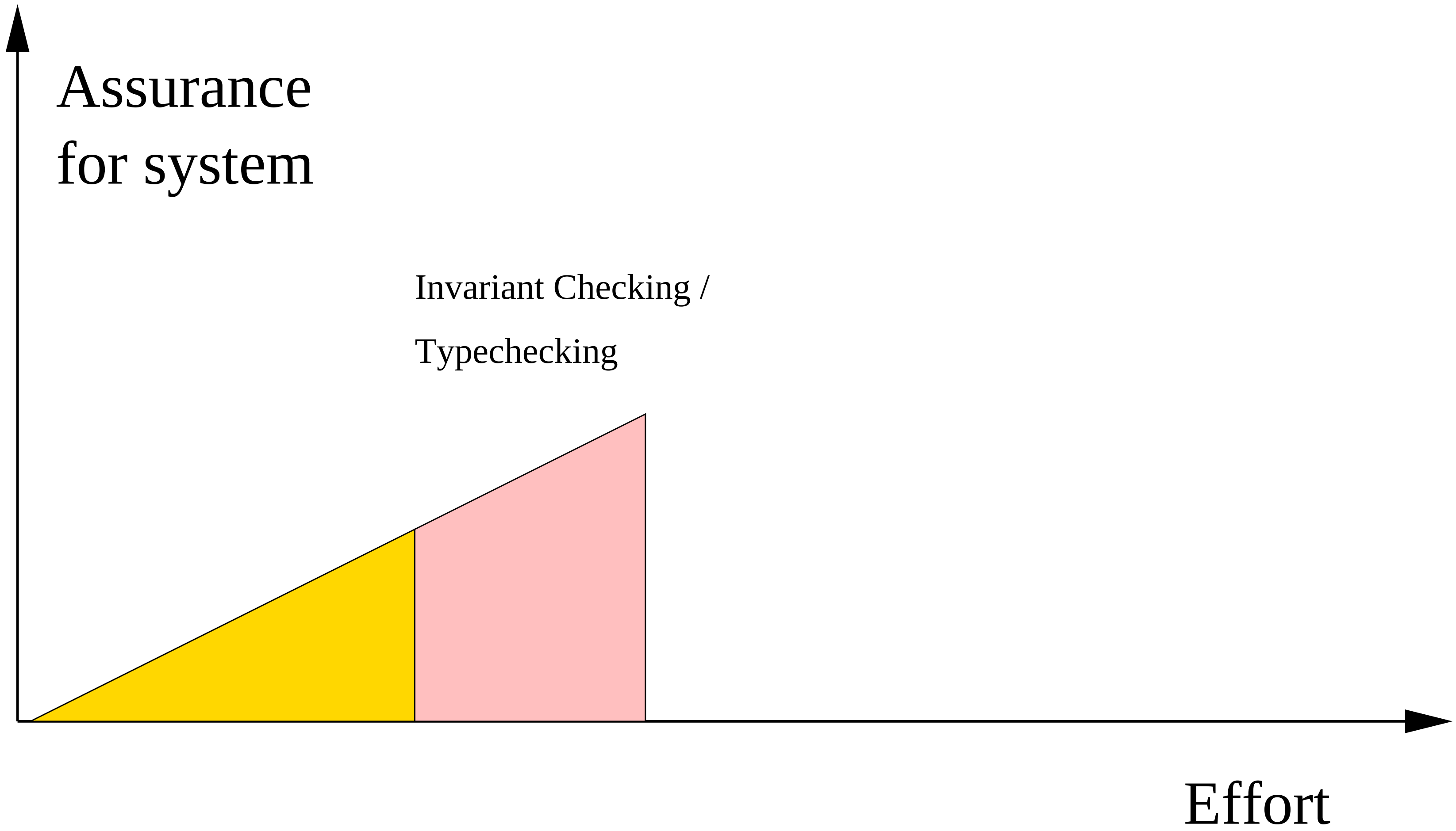
Analysis Techniques



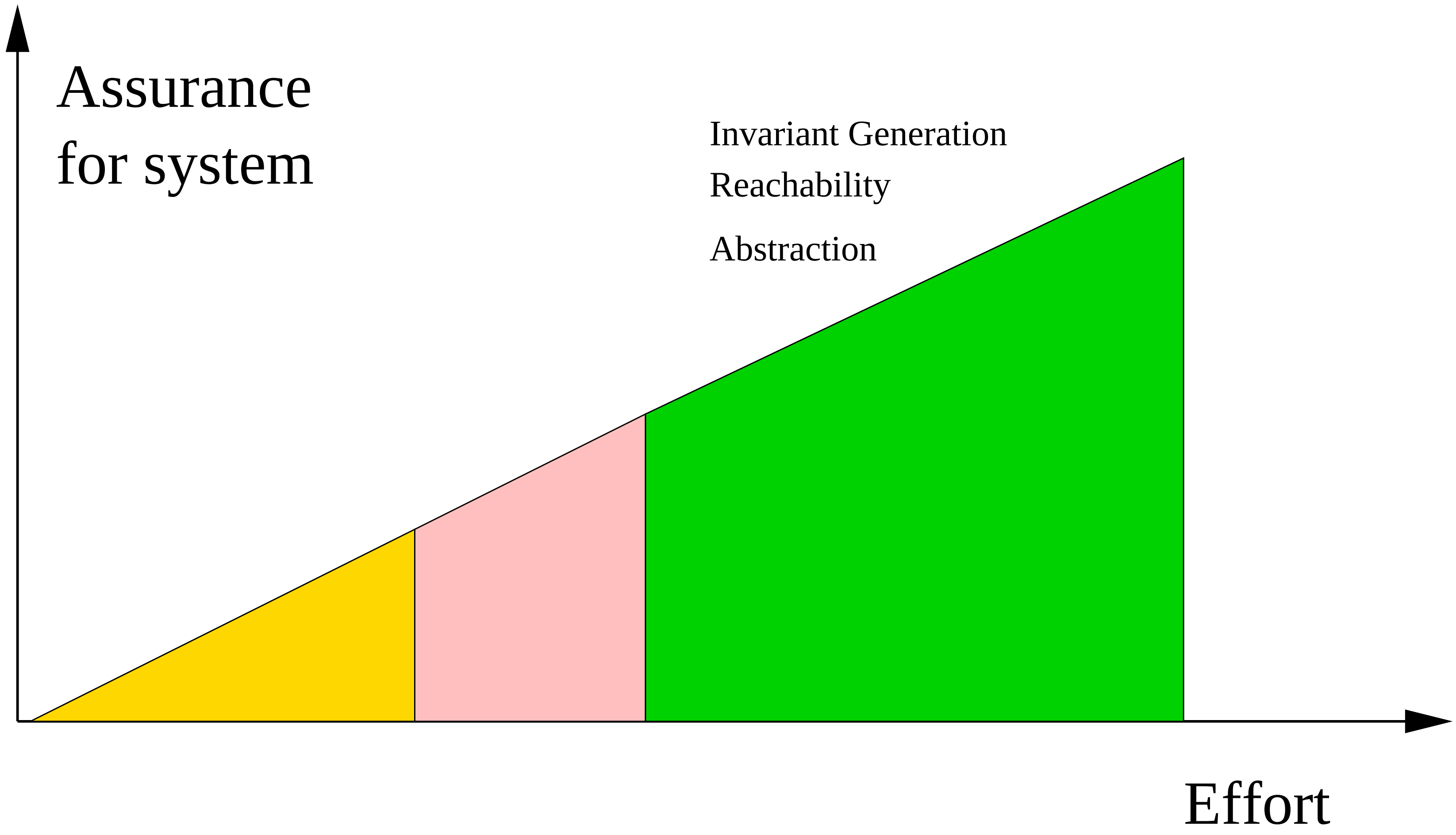
Analysis Techniques



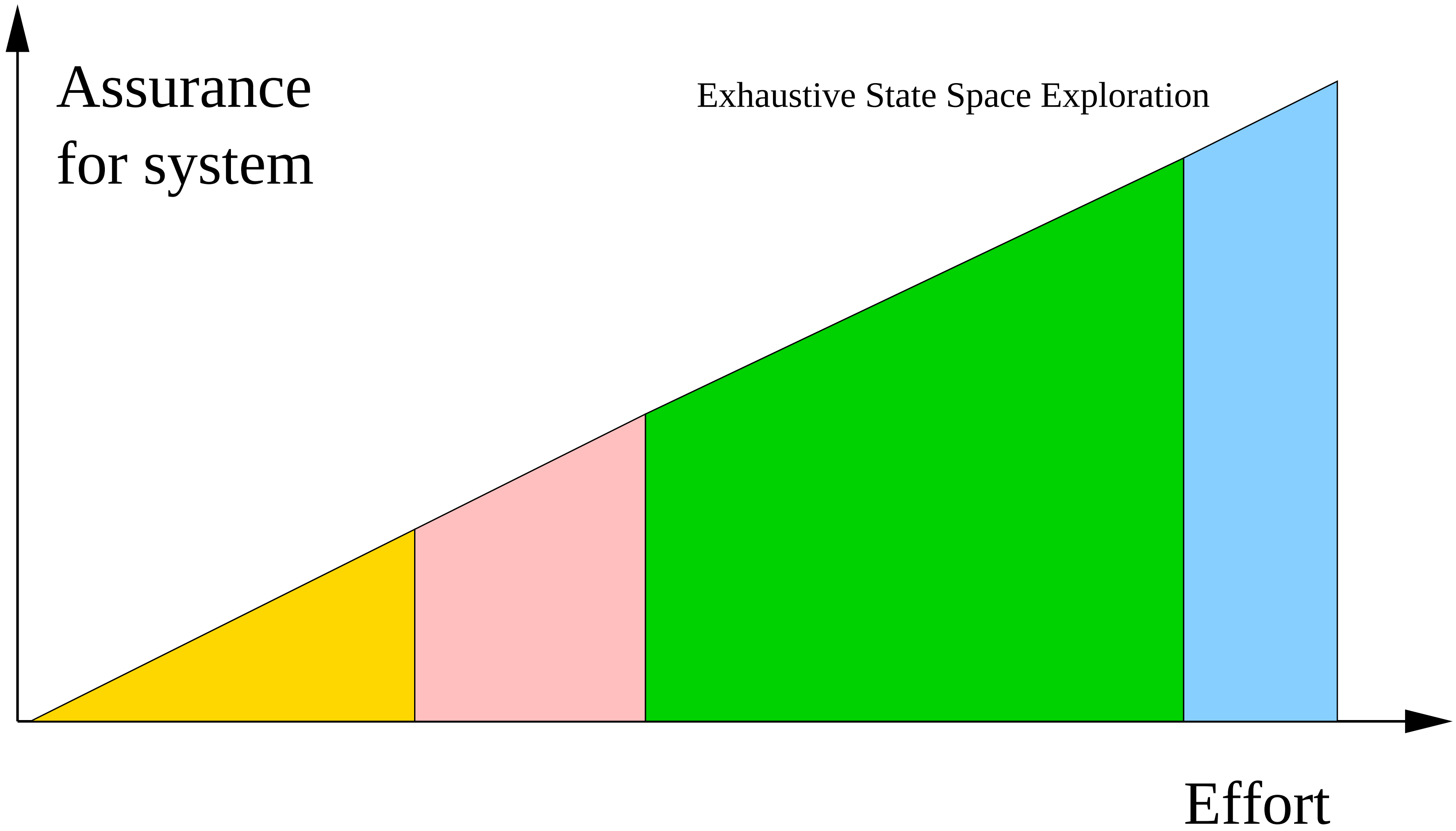
Analysis Techniques



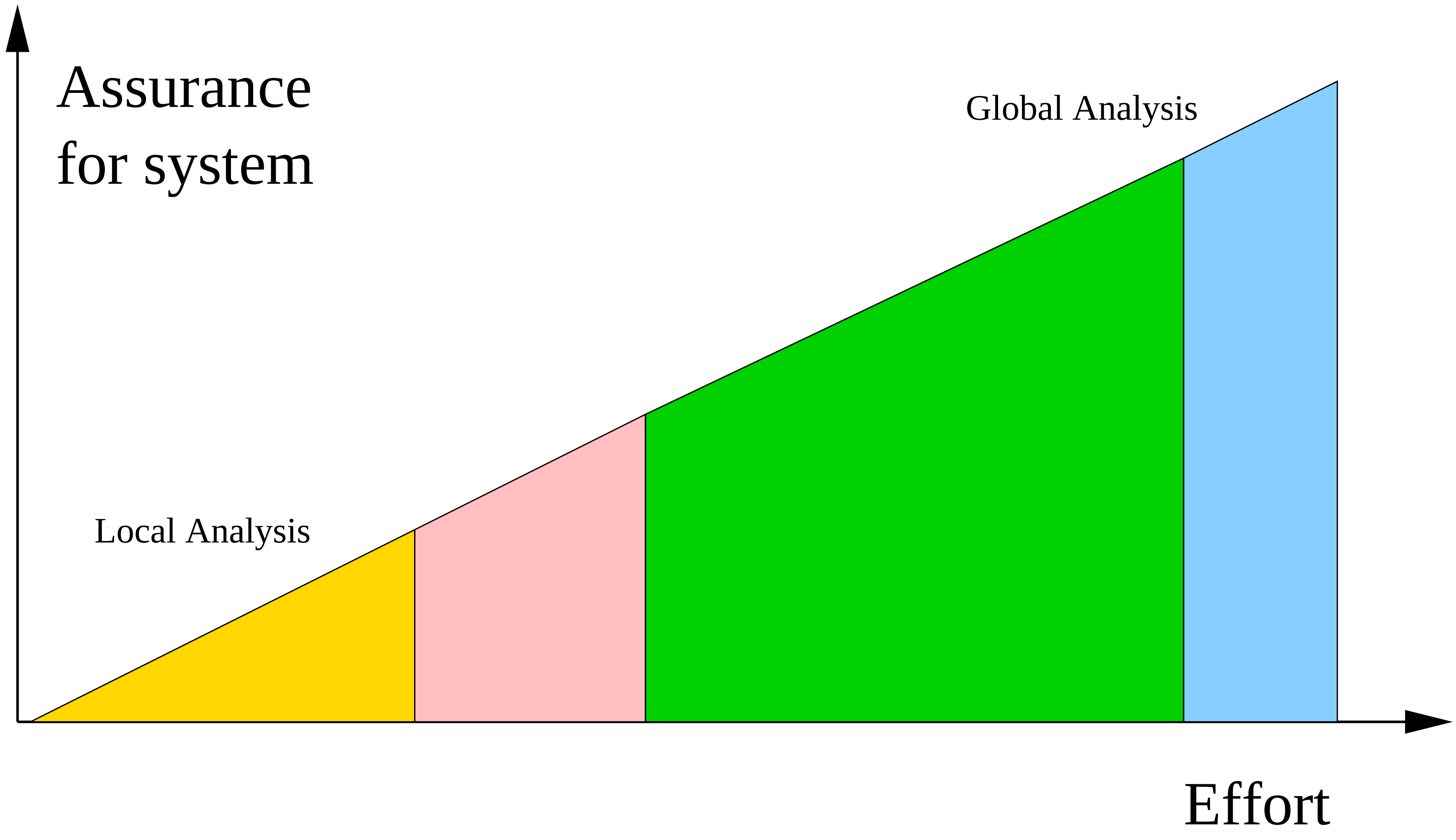
Analysis Techniques



Analysis Techniques



Analysis Techniques



SAL: Language

SAL models transition systems and supports

- Transitions: Definitions and guarded commands
- Modules: input, output, local, global variables
- Composition of modules

Supported by theorem-provers, model-checkers, and program analyzers

SAL: Tool Suite

- Simple typechecking
- Symbolic Simulation
- Invariant Checking
- Invariant Generation
- Abstraction

All of these tools work on modules. Module could represent individual components of the system, or the full system.

Benefit to Development Process

- Early detection of errors: models can be typechecked and verified in the design phase
- Reduction in the development cycle time
- Provably correct transformation and mapping onto target architecture
- Extra information generated in the verification process may be used for efficient code generation

Tool Interfaces

Verification tool—

Input	:	Stateflow-Simulink, or SAL language
Intermediate Representation	:	SAL (XML)
Output	:	SAL Theorems

We have a translator from Stateflow-Simulink abstract (logical) syntax to SAL.

Tool Integration

SAL is designed for easy integration with other verification tools.

- SAL concrete syntax is XML based.
- SAL analysis capabilities comprise of a collection of independent tools.
- Different tools communicate through XML and a tool bus management software is under development.

The ETC Example in SAL

```
ETC : CONTEXT =
BEGIN
    Driver : MODULE = . . .
    Actuator : MODULE = . . .
    Controller : MODULE = . . .
    HumanController : MODULE = . . .
    Plant : MODULE = . . .
END
```

ETC: Driver Spec

```
Driver : MODULE =
   BEGIN
        INPUT duty : REAL
        LOCAL lduty, cnt : REAL
        LOCAL mode : BOOLEAN
        OUTPUT pwm : REAL
        INITIALIZATION . . .
        TRANSITIONS . . .
   END;
```

Given *duty* s.t. $0 < duty < 100$, output a pwm signal.

ETC: Driver Specification

TRANSITIONS

[

$mode = \mathrm{F} \;\wedge\; cnt = 0 \;\wedge\; duty > 0 \;\wedge\; duty < 100 \longrightarrow$

$\quad lduty' = duty; \quad mode' = \mathrm{T};$

$\quad pwm' = 1; \quad cnt' = 100$

[]

$mode = \mathrm{T} \;\wedge\; cnt < lduty \longrightarrow$

$\quad mode' = \mathrm{F}; \quad pwm' = 0$

[]

$(mode = \mathrm{F} \;\wedge\; cnt > 0) \;\vee\; (mode = \mathrm{T} \;\wedge\; cnt \geq \mathrm{lduty}) \longrightarrow$

$\quad cnt' = cnt - 1$

]

Driver: Symbolic Propagation

sal(45): (propagate-up 'ETC 'Driver)

sal(48): (widen 'ETC 'Driver "...")

The widening is correct.

sal(49): (propagate-up 'ETC 'Driver)

sal(52): (widen 'ETC 'Driver "...")

The widening is correct.

sal(53): (propagate-up 'ETC 'Driver)

The formula

$$(\mathit{mode} = \mathrm{T} \wedge \mathit{pwm} = 1 \wedge 0 < \mathrm{lduty} < 100 \wedge \mathit{lduty} - 1 \leq \mathit{cnt} \leq 100) \vee$$

$$(\mathit{mode} = \mathrm{F} \wedge \mathit{pwm} = 0 \wedge 0 < \mathrm{lduty} < 100 \wedge -1 < \mathit{cnt} < \mathit{lduty})$$

is an invariant.

Driver: Assigning Types

Variable *lduty* can be declared to be of type:

$$\{x\text{:INT} \mid 0 < x \ \wedge \ x < 100\}.$$

Similarly, variable cnt is of *type*:

$$\{x\text{:INT} \mid \text{if} \quad mode \quad \text{then} \quad lduty - 1 \leq x \leq 100 \quad \text{else} \quad 0 \leq x < lduty\}$$

Typechecking establishes correctness. Typechecking involves one step of symbolic simulation.

ETC: Actuator

```
Actuator : MODULE =
  BEGIN
      INPUT pwm_state : BOOLEAN
      LOCAL Vc, i : REAL
      OUTPUT Trq_throttle : REAL
      INITIALIZATION . . .
      TRANSITIONS . . .
  END;
```

Actuator outputs *Trq_throttle* based on the input pwm signal.

ETC: Actuator Specification

TRANSITIONS

[

$pwm\_state = \mathrm{T} \longrightarrow$

$Vc' = Vc + 2/9 * (24 - i - 2*Vc);$

$i' = i + (1/15) * (120 - 22*i);$

$Trq\_throttle' = 3/250*i$

[]

$pwm\_state = \mathrm{F} \longrightarrow$

$Vc' = Vc - 2/3 * i;$

$i' = i + 2/15 * (5*Vc - 16*i);$

$Trq\_throttle' = 3/250*i$

]

ETC: Actuator Analysis

Using the same technique, we can show that when *pwm_state* is TRUE

$$Trq\_throttle = 3 / 250 * i \ \wedge \ Vc = 102 / 11 \ \wedge \ i = 60 / 11$$

is a stable solution, and when *pwm_state* is FALSE, it is

$$Trq\_throttle = 3 / 250 * i \ \wedge \ Vc = 0 \ \wedge \ i = 0.$$

ETC: Abstracting the System

Properties of individual components help in getting an abstract system.

Replace the driver and actuator modules by a simplified module: given duty $0 \leq d \leq 1$, *Trq_throttle* is 0.065 for d-fraction of the time, and 0 for $(1-d)$-fraction of the time.

ETC: System

```
System : MODULE =
BEGIN
    INPUT desired : REAL
    LOCAL alpha, omega : REAL
    LOCAL mode : BOOLEAN
```

Discrete transition triggers:

$$|160*(alpha - desired)| - 3$$

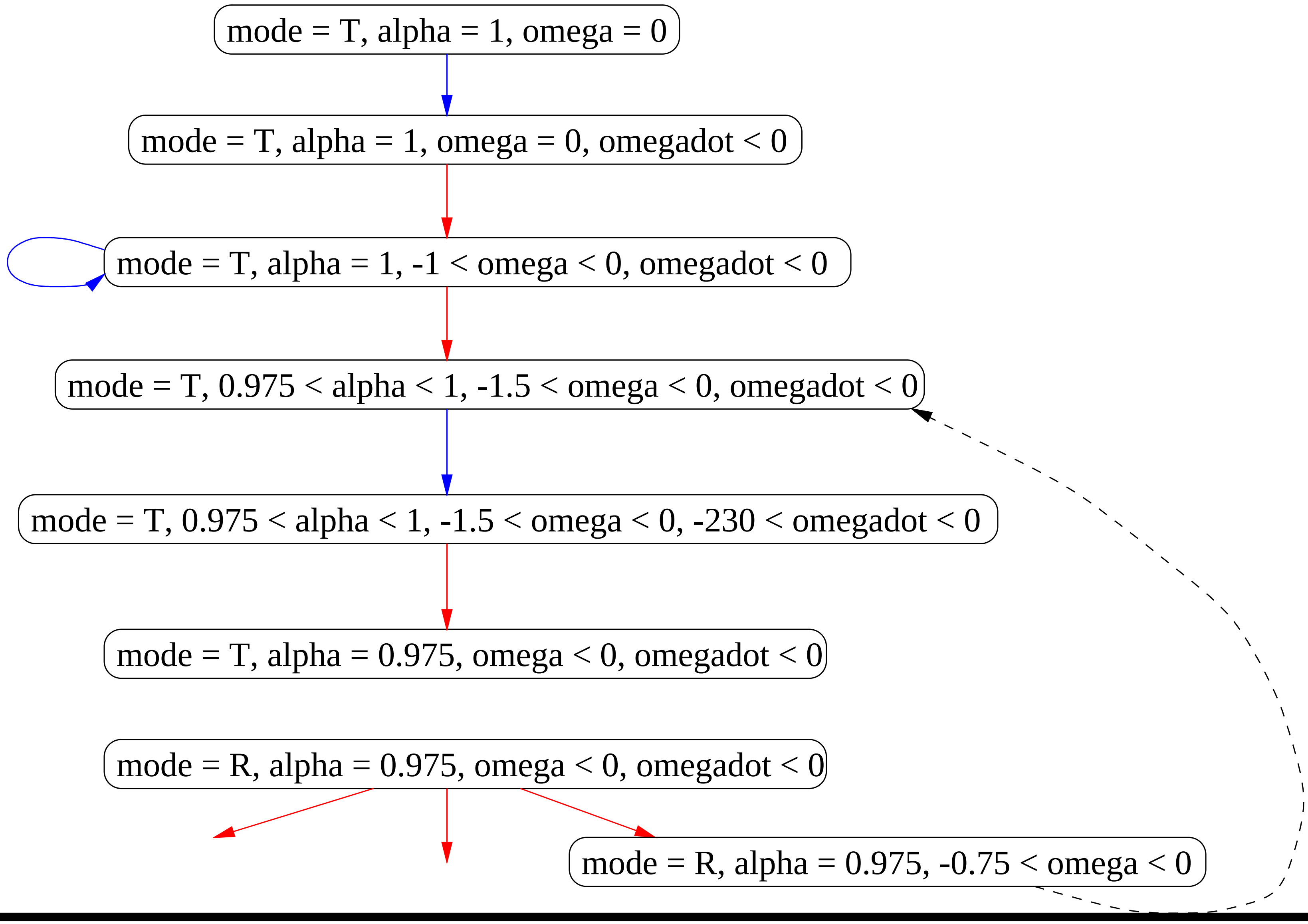
$$|40*(alpha - desired)| - 1$$

omega

$$(alpha - desired)*30 + omega$$

...

ETC: System



mode = T, alpha = 1, omega = 0
mode = T, alpha = 1, omega = 0, omegadot < 0
mode = T, alpha = 1, -1 < omega < 0, omegadot < 0
mode = T, 0.975 < alpha < 1, -1.5 < omega < 0, omegadot < 0
mode = T, 0.975 < alpha < 1, -1.5 < omega < 0, -230 < omegadot < 0
mode = T, alpha = 0.975, omega < 0, omegadot < 0
mode = R, alpha = 0.975, omega < 0, omegadot < 0
mode = R, alpha = 0.975, -0.75 < omega < 0

Building the Abstraction

Each new symbolic state is obtained using

- simulation of current symbolic state
- widening the reached symbolic state

Thus, we have a tool suite for analysis ranging from typechecking to complete verification via invariant generation, abstraction, and model-checking.