

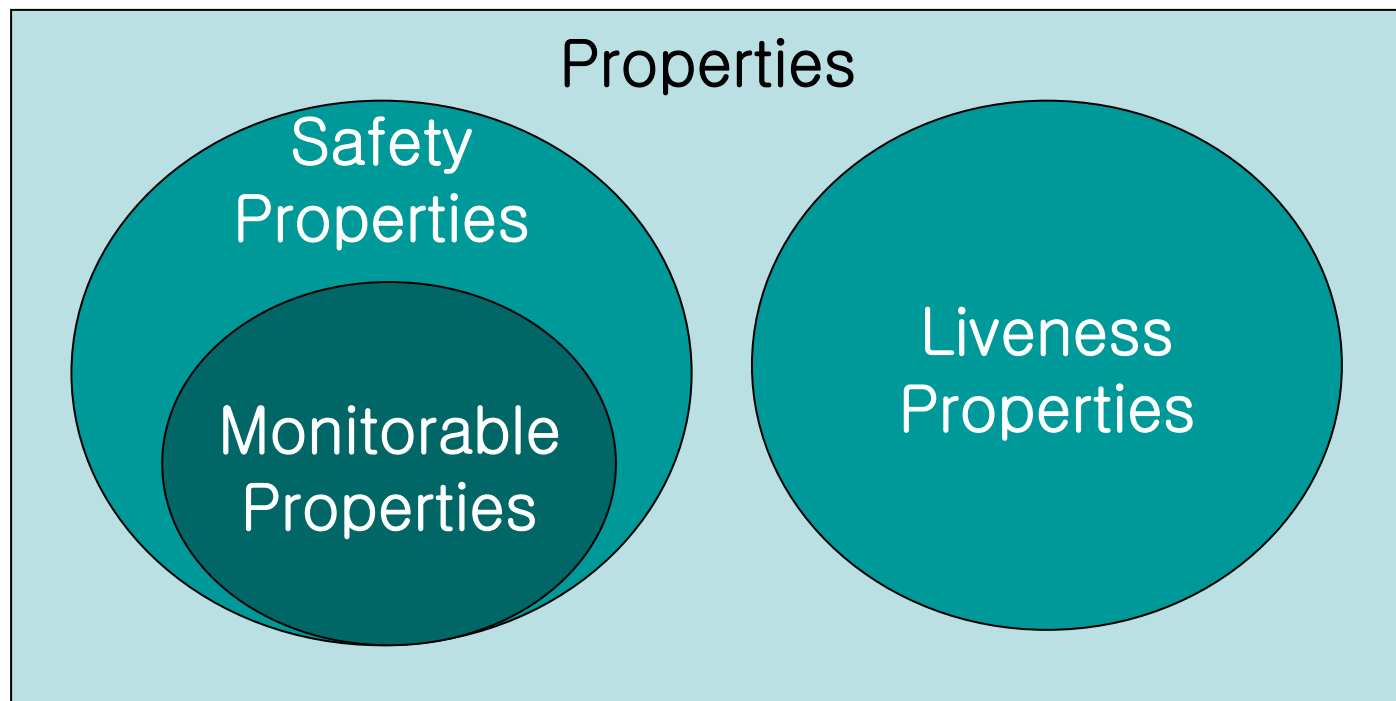
Computational Analysis of Run-time Monitoring

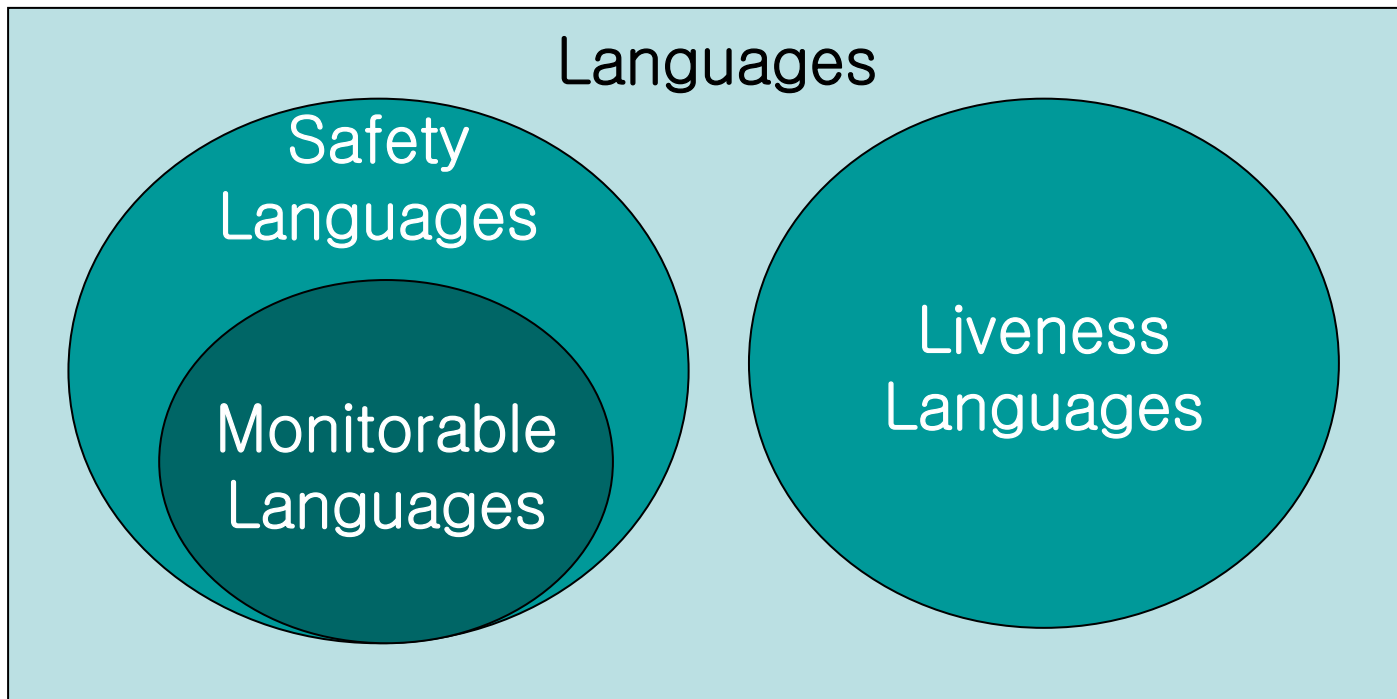
M. Kim, S. Kannan, I. Lee,
O. Sokolsky, M. Viswanathan

Presented by Moonjoo Kim

- Class of Monitorable Properties
- Evaluation of Properties in Process Algebra
- Abstraction of Program Execution
 - Example: the Sieve of Eratosthenes
- Conclusion

- A **property** is a set of program executions
 - An **execution** of a program is an infinite sequence of program states S





- Example

- $\Sigma = \{0, 1, a, b\}$

- $H_* = \{ x \cdot a \cdot y \mid x, y \in \{0, 1\}^* \}$,

the Turing Machine encoded by x halts on input y .

- Membership of H_* (the halting problem) is undecidable

–We can define a safety property

$$H_\omega = H_* \cdot b^\omega \cup \{0, 1\}^* \cdot a \cdot \{0, 1\}^\omega \cup \{0, 1\}^\omega$$

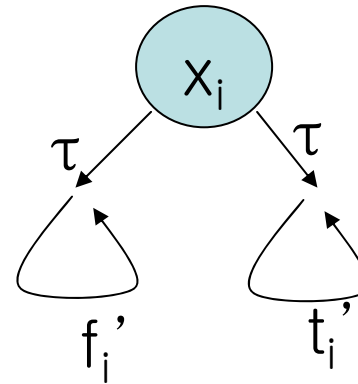
- A property $P \subseteq S^\omega$ is **monitorable** iff
 - P is a safety property
 - $S^* \setminus \text{pref}(P)$ is **recursively enumerable**

- Topic :
 - Complexity of evaluating properties based on a given trace (trace validity problem)
 - When properties are given in process algebra, this problem becomes **NP-complete**
- Trace Validity Problem
 - Input: a process P and a string $s \in L^*$
 - Output: is s a valid trace of P

- NP-completeness proof
 - reduction of 3SAT to the trace validity problem
- Proof sketch
 - A formula ϕ in CNF w/
 - variables $x_1 \dots x_n$
 - Clauses $C_1 \dots C_m$ where $C_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$
 - We construct
 - a process $P(\phi)$
 - A string $s(\phi)$

s.t. $s(\phi)$ is a valid trace of $P(\phi)$ iff ϕ is satisfiable

- For each variable x_i , define processes X_i (assignment of x_i)
 - $X_i = \tau.F_i + \tau.T_i$
 - $F_i = f_i'.F_i$
 - $T_i = t_i'.T_i$
- We define a process P s.t.
 - P runs concurrently with X_i 's
 - P will deadlock iff there is no truth assignment of x_i 's which satisfies ϕ
 - $P(\phi) = (P \parallel X_i \parallel \dots \parallel X_n) \setminus \{t_1, f_1, \dots, t_n, f_n\}$



- $Q_i \rightarrow^* Q_{i+1}$ iff one of the literals in C_i gets true under X_i 's
 - For any $L_{i,j}$,
 $L_{i,j} \rightarrow L'_{i,j}$ iff $L_{i,j}$ gets the truth value true under X_i
- abab...abab is a valid trace of $P(\phi)$ iff ϕ is satisfiable

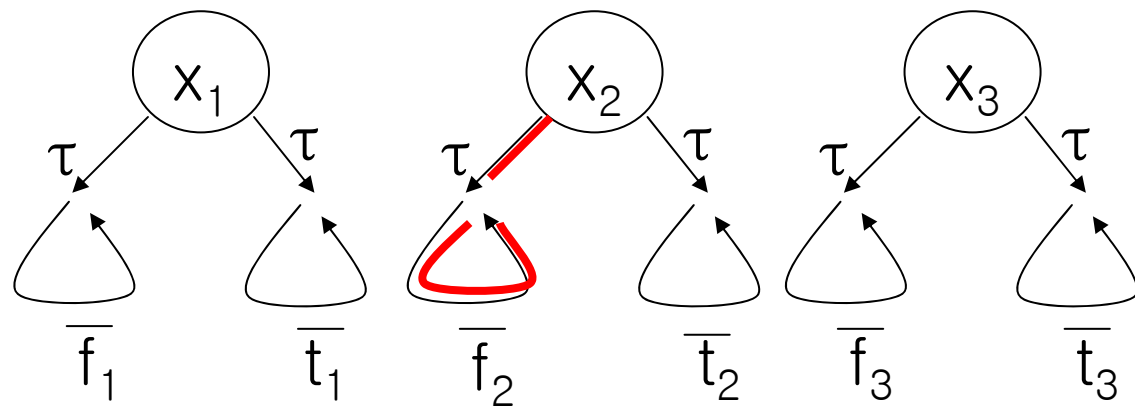
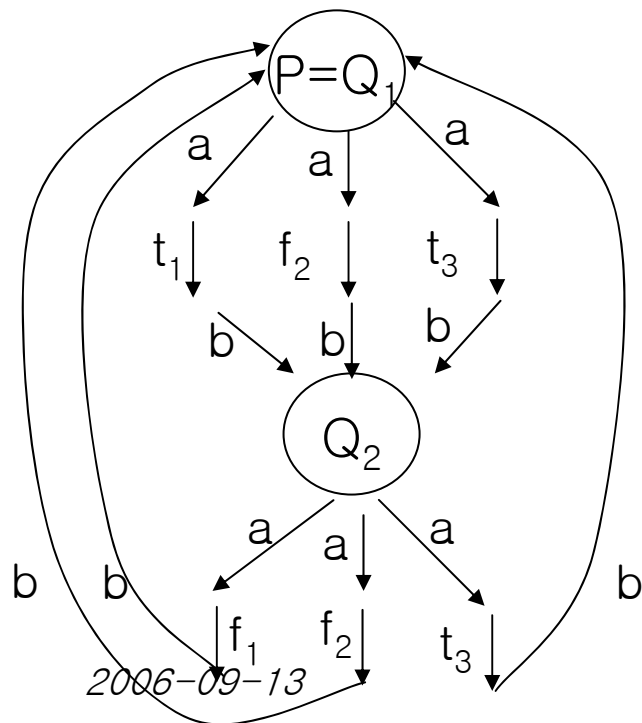
$$\begin{aligned} P &\stackrel{\text{def}}{=} Q_1 \\ Q_1 &\stackrel{\text{def}}{=} a.L_{1,1} + a.L_{1,2} + a.L_{1,3} \\ &\vdots \\ Q_i &\stackrel{\text{def}}{=} a.L_{i,1} + a.L_{i,2} + a.L_{i,3} \\ L_{i,j} &\stackrel{\text{def}}{=} \begin{cases} f_k.L'_{i,j} & \text{if } l_{i,j} \equiv \neg x_k \\ t_k.L'_{i,j} & \text{if } l_{i,j} \equiv x_k \end{cases} \\ L'_{i,j} &\stackrel{\text{def}}{=} b.Q_{i+1} \\ &\vdots \\ L'_{m,j} &\stackrel{\text{def}}{=} b.Q_1 \end{aligned}$$

- Example.

- $C = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$

- Truth assignment:

- x_1 (true), x_2 (false), x_3 (true/false)

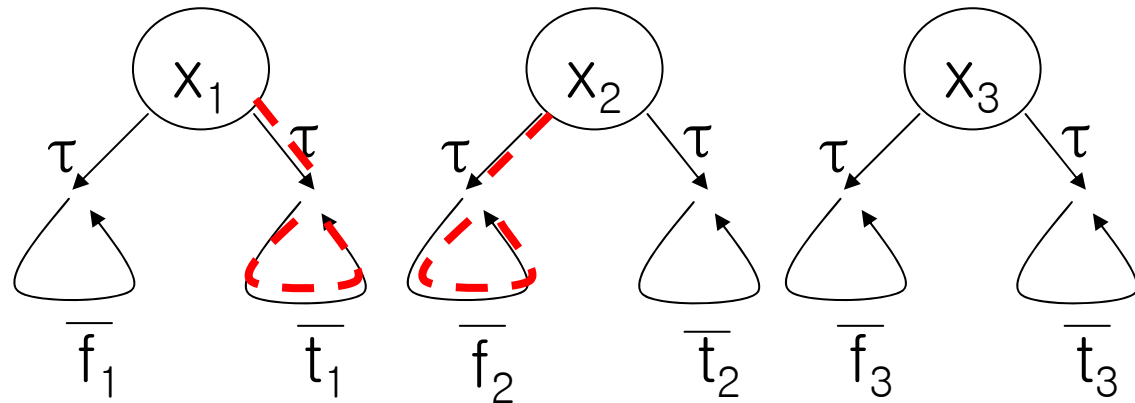
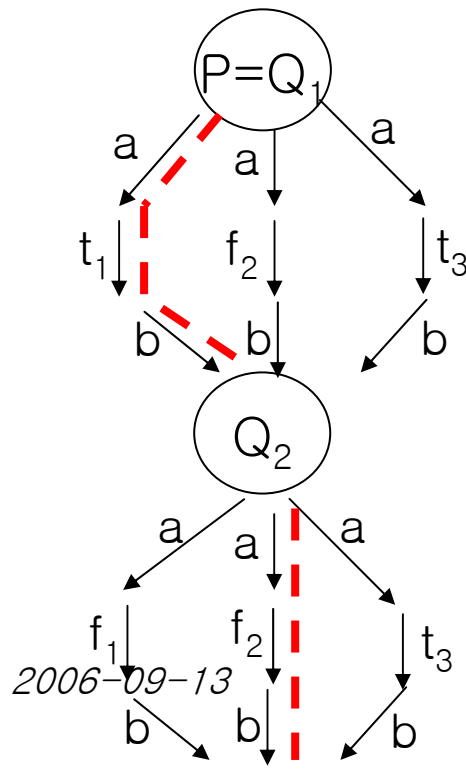


- Example.

- $C = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$

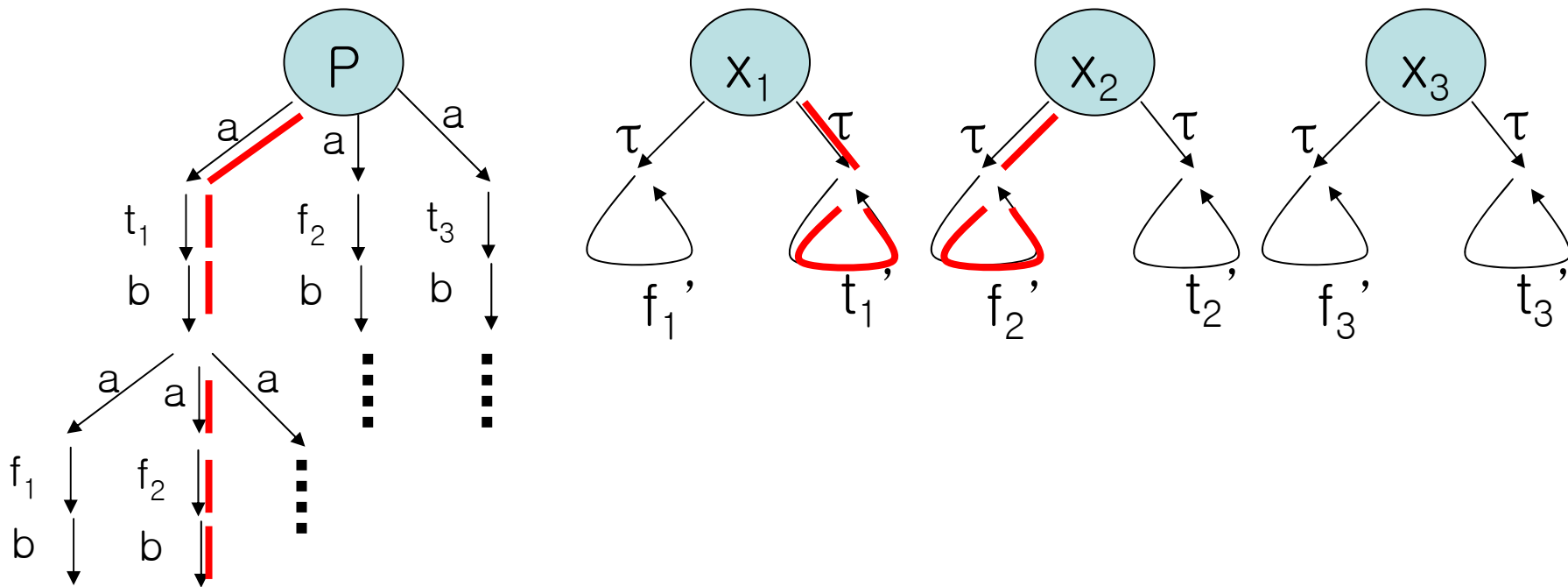
- Truth assignment:

- x_1 (true), x_2 (false), x_3 (true/false)



Proof Sketch (cont.)

- Example.
 - $C = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
 - Truth assignment:
 - x_1 (true), x_2 (false), x_3 (true/false)



- Abstraction of an execution can be thought as reduction of execution trace
- **Value abstraction** γ_{exp} with regard to exp is a function s.t. $\gamma_{exp} : S^\omega \rightarrow (S^\omega \cup S^*)$

$$\gamma_{exp_{V_m}}(s_i s_{i+1} \sigma') = \begin{cases} \gamma_{exp_{V_m}}(s_i \sigma') & \text{if } \forall e \in exp_{V_m} \cdot \llbracket e \rrbracket_{\rho_{s_i}} = \llbracket e \rrbracket_{\rho_{s_{i+1}}} \\ s_i \gamma_{exp_{V_m}}(s_{i+1} \sigma') & \text{if } \exists e \in exp_{V_m} \cdot \llbracket e \rrbracket_{\rho_{s_i}} \neq \llbracket e \rrbracket_{\rho_{s_{i+1}}} \end{cases}$$

Abstraction of Program Execution (cont.)

- Value abstraction γ_{exp} is **valid** with regard to $prop_{req}$ iff

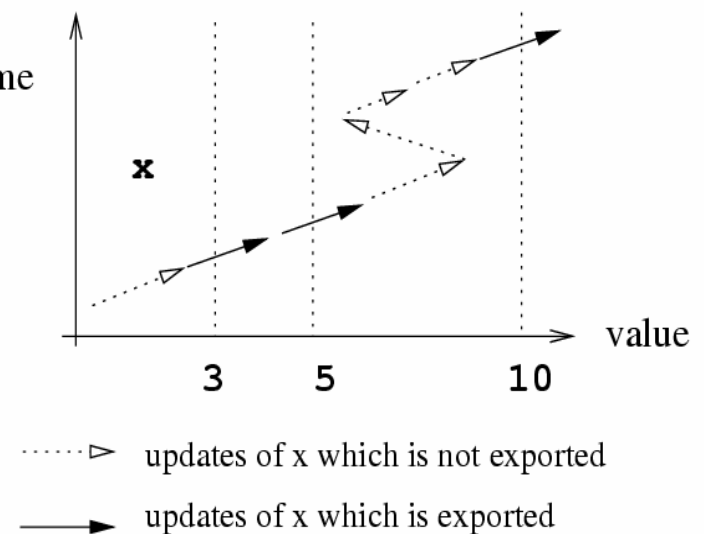
$$\forall j \geq 0. \left(\forall e \in exp_{Vm}. \llbracket e \rrbracket_{\rho_{s_j}} = \llbracket e \rrbracket_{\rho_{s_{j+1}}} \longrightarrow \forall p \in prop_{req}. \llbracket p \rrbracket_{\rho_{s_j}} = \llbracket p \rrbracket_{\rho_{s_{j+1}}} \right)$$

- Example**

- $p_{req\ 1} = (3 < x \wedge x < 10) \vee (z > 10)$ time

- $p_{req\ 2} = (x > 5) \wedge (y > 2z + 3)$

- $exp = \{ 3 < x, x < 10, x > 5 \}$



- Provides primitives to refer to
 - primitive variables
 - beginnings/endings of methods
- Primitive conditions are constructed from
 - boolean-valued expressions over the monitored variables
 - ex> condition IC = (position == 100);
- Primitive events are constructed from
 - update(x)
 - startM(f)/endM(f)
 - ex> event raiseGate= startM(Gate.gu());

```
MonScr <spec_name>
  /* Export section */
  export event <e>;
  export condition <c>;

  /* Monitored entities */
  monobj <var>;
  monmeth <meth>;

  /* Event and condition*/
  event <e> = ...;
  condition <c>= ...;

End
```

- Events can have two attributes - time and value
- $\text{time}(e)$ gives the time of the last occurrence of event e
 - used for expressing temporal properties
- $\text{value}(e,i)$ gives the i th value in the tuple of values of e
 - value of $\text{update}(\text{var})$: a tuple containing a current value of var
 - value of $\text{startM}(f)$: a tuple containing parameters of the method f
 - value of $\text{endM}(f)$: a tuple containing parameters and a return value of the method f

- Sieve of Eratosthenes

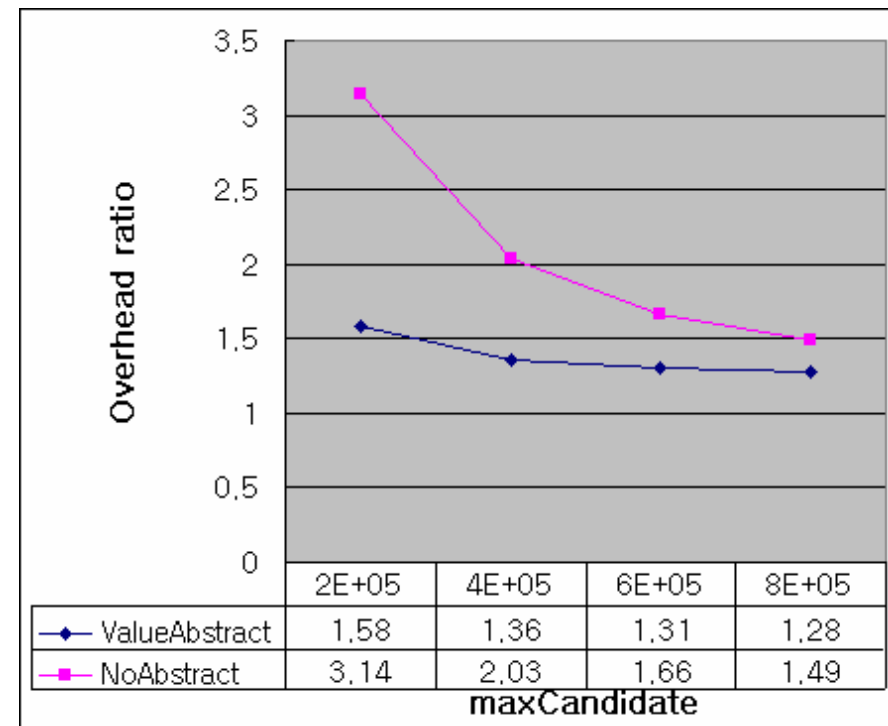
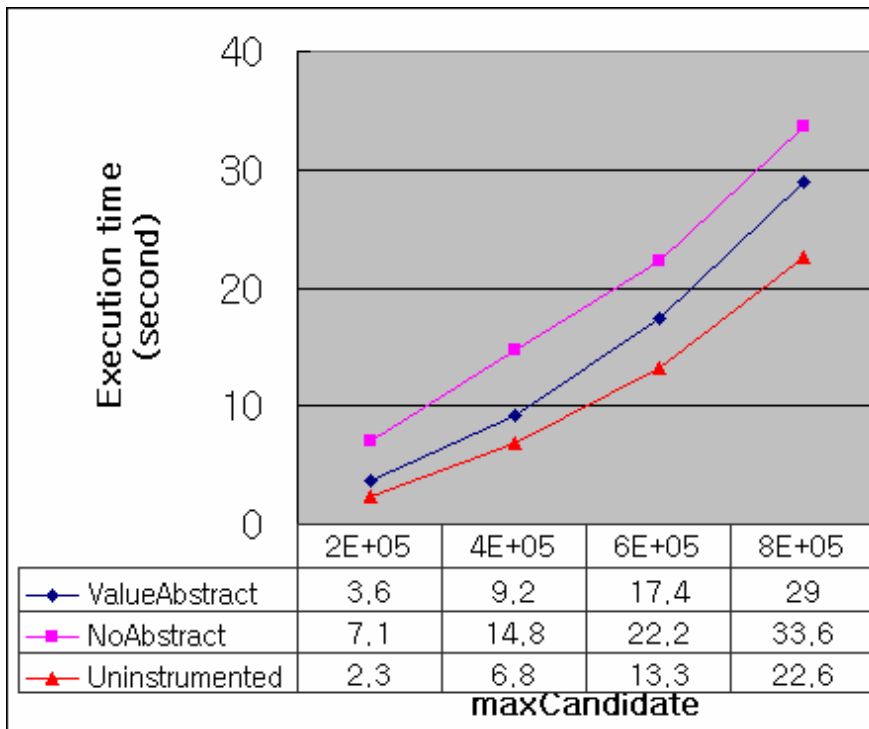
Make a list of all the integers less than or equal to n (and greater than one). Strike out the multiples of all primes less than or equal to \sqrt{n} , then the numbers that are left are the primes.

- We would like to monitor and check existence of a prime number between 99990 and 100000

```
01:MonScr
02:  export event foundPrime;
03:  monobj int SieveMain.sa.numTested;
04:  monobj int SieveMain.sa.numPrimes;
05:  event foundPrime = update(SieveMain.sa.numPrimes) when
06:                        (99990 <= SieveMain.sa.numTested
07:                        && SieveMain.sa.numTested <= 100000);
08:end
```

Experimental Result

- When $n = 200000$,
 - Monitoring overhead slows down the program 3.1 times
 - Value abstraction reduces the overhead 73 %



- A class of monitorable properties is **a strict subset** of safety properties
- Evaluating a requirement written in process algebra is **NP-complete**
- Value abstraction is simple, but effective technique easily applicable to monitoring system
 - Applying value abstraction aggressively with statistical history seems promising