Predicting Space Requirements for a Stream Monitor Specification Language

David M. Cerna, Wolfgang Schreiner, and Temur Kutsia

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

September 28^{th}, 2016

Introduction

- LogicGuard: A coordination language for runtime monitoring of network traffic.
- Stream monitors are written in a fragment of predicate logic.

Introduction

- LogicGuard: A coordination language for runtime monitoring of network traffic.
- Stream monitors are written in a fragment of predicate logic.
- Monitor instances are evaluated using an operational semantics.
- Violations, monitor instances evaluating to false, are flagged.

Introduction

- LogicGuard: A coordination language for runtime monitoring of network traffic.
- Stream monitors are written in a fragment of predicate logic.
- Monitor instances are evaluated using an operational semantics.
- Violations, monitor instances evaluating to false, are flagged.
- Previous work focused on analysis of the stream history [Kutsia and Schreiner 2014], and coarse space complexity results [Cerna et al. 2016].
- We present an algorithm which outperforms the previous results.
- In special cases is computes precisely the space complexity of the instance set, and in general provides acceptable bounds.

The LogicGuard Specification Language

- LogicGuard was developed to monitor boolean event streams from external sources.
- The full language does not have a concept of absolute position and messages can be assigned the same time unit depending on arrival time and coarseness of the system clock.

The LogicGuard Specification Language

- LogicGuard was developed to monitor boolean event streams from external sources.
- The full language does not have a concept of absolute position and messages can be assigned the same time unit depending on arrival time and coarseness of the system clock.
- The core language [Kutsia and Schreiner 2014] on the other hand has absolute positions allowing for easier analysis of a specification's behaviour.
- Also, LogicGuard allows both value computation and internal stream construction.

The LogicGuard Specification Language

- LogicGuard was developed to monitor boolean event streams from external sources.
- The full language does not have a concept of absolute position and messages can be assigned the same time unit depending on arrival time and coarseness of the system clock.
- The core language [Kutsia and Schreiner 2014] on the other hand has absolute positions allowing for easier analysis of a specification's behaviour.
- Also, LogicGuard allows both value computation and internal stream construction.
- These feature were removed from the core language to limit the express power.
- Though the core language is expressive enough to approximate the behaviour of such concepts

Example LogicGuard Specification

```
type tcp; type message; ...
stream<tcp> IP;
stream<message> S = stream<IP> x satisfying start(@x) :
  value[seq,@x,combine]<IP> y
      with x < _ satisfying same(@x,@y) until end(@y) :
    @y ;
monitor<S> M = monitor<S> x satisfying trigger(@x) :
  exists<S> y with x < _ <=# x+5000:
    match(@x,@y);
```

Core Language

$$
\begin{aligned}
M &::= \forall_{0 \le V} : F. \\
F &::= @V \mid \neg F \mid F \wedge F \mid F \,\&\, F \mid \forall_{V \in [B,B]} : F. \\
B &::= 0 \mid \infty \mid V \mid B \pm N. \\
V &::= x \mid y \mid z \mid \ldots \\
N &::= 0 \mid 1 \mid 2 \mid \ldots
\end{aligned}
$$

$$\forall_{0 \le x} : \forall_{y \in [x+1, x+5]} : ((\forall_{z \in [y, x+3]} : \neg @z \,\&\, @z) \,\&\, G(x,y))$$

$$G(x,y) = \forall_{w \in [x+2, y+2]} : (\neg @y \,\&\, (\forall_{m \in [y,w]} : \neg @x \,\&\, @m))$$

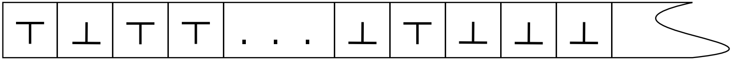
The Operational Semantics

Atomic Formulas		
#	Transition	Constraints
A1	$\mathbf{n}(@V) \rightarrow \mathbf{d}(c.2(V)))$	$V \in dom(c.2)$
...		
Sequential conjunction		
C1	$\mathbf{n}(f_1 \ \& \ f_2) \rightarrow \mathbf{n}(\mathbf{n}(f_1') \ \& \ f_2)$	$f_1 \rightarrow \mathbf{n}(f_1')$
C2	$\mathbf{n}(f_1 \ \& \ f_2) \rightarrow \mathbf{d}(\bot)$	$f_1 \rightarrow \mathbf{d}(\bot)$
C3	$\mathbf{n}(f_1 \ \& \ f_2) \rightarrow \mathbf{n}(f_2')$	$f_1 \rightarrow \mathbf{d}(\top)$, $f_2 \rightarrow \mathbf{n}(f_2')$
Quantification		
Q1	$\forall_{V \in [b_1, b_2]} : f \rightarrow \mathbf{d}(\top)$	$p_1 = b_1(c)$, $p_2 = b_2(c)$, $p_1 = \infty \vee p_1 > p_2$
Q2	$\forall_{V \in [b_1, b_2]} : f \rightarrow f'$	$p_1 = b_1(c)$, $p_2 = b_2(c)$, $p_1 \neq \infty$, $p_1 \leq p_2$, $\mathbf{n}(\forall_{V \in [p_1, p_2]} : f) \rightarrow f'$
Q3	$\mathbf{n}(\forall_{V \in [p_1, p_2]} : f) \rightarrow \mathbf{n}(\forall_{V \in [p_1, p_2]} : f)$	$p < p_1$
Q4	$\mathbf{n}(\forall_{V \in [p_1, p_2]} : f) \rightarrow f'$	$p_1 \leq p$, $\mathbf{n}(\forall^{I_0}_{V \leq p_2} : f) \rightarrow f'$
Q5	$\mathbf{n}(\forall^{I}_{V \leq p_2} : f) \rightarrow \mathbf{d}(\bot)$	DF
Q6	$\mathbf{n}(\forall^{I}_{V \leq p_2} : f) \rightarrow \mathbf{d}(\top)$	$\neg DF$, $I'' = \emptyset$, $p_2 < p$
Q7	$\mathbf{n}(\forall^{I}_{V \leq p_2} : f) \rightarrow \mathbf{n}(\forall^{I''}_{V \leq p_2} : f)$	$\neg DF$, $(I'' \neq \emptyset \vee p \leq p_2)$

$$DF \equiv \exists t \in \mathbb{N}, f \in \mathcal{F}, c \in \mathcal{C} : (t, f, c) \in I' \wedge \vdash f \rightarrow \mathbf{d}(\bot)$$

The Problem

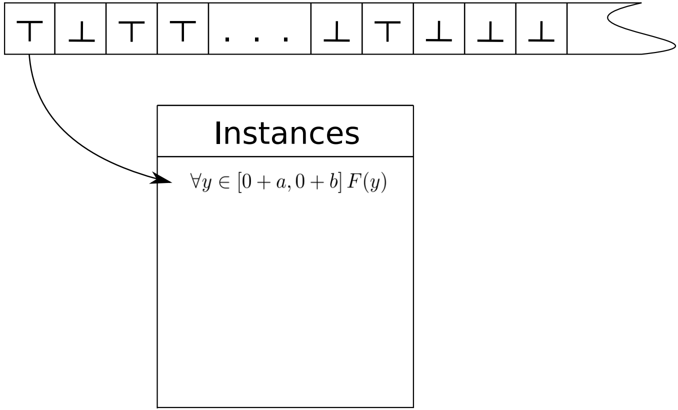
$x = \; 0 \quad 1 \quad 2 \quad 3 \quad . \; . \; .$



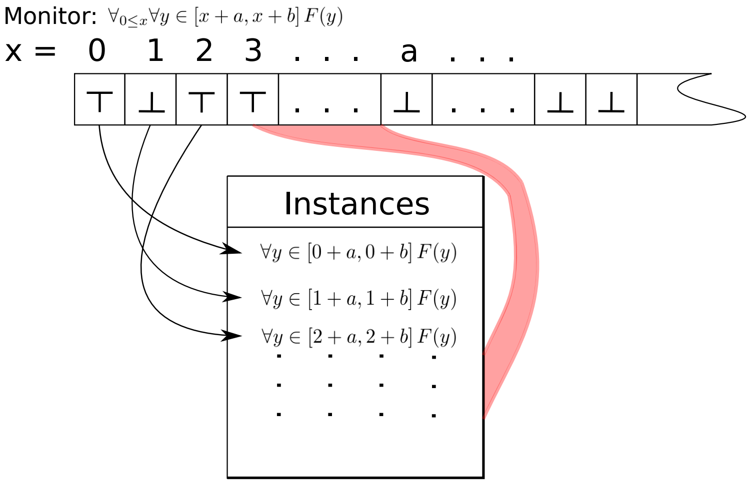
The Problem

Monitor: $\forall_{0 \leq x} \forall y \in [x + a, x + b]\, F(y)$

x = 0 1 2 3 . . .

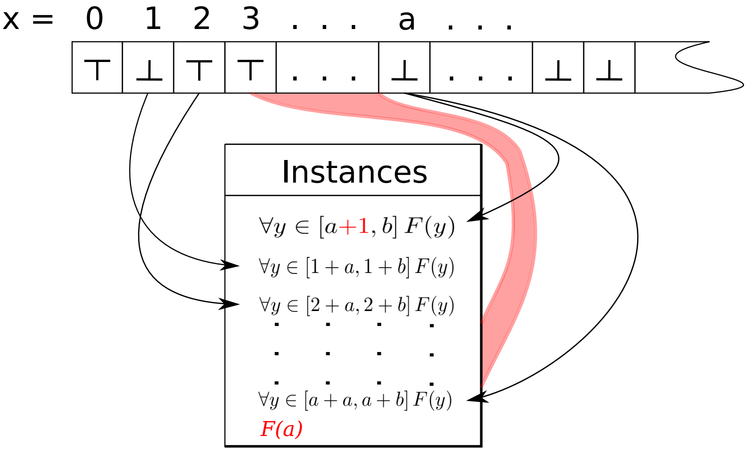


The Problem


Monitor: $\forall_{0\leq x} \forall y \in [x+a, x+b]\, F(y)$
x = 0 1 2 3 . . . a . . .
⊤ ⊥ ⊤ ⊤ . . . ⊥ . . . ⊥ ⊥
Instances
$\forall y \in [0+a, 0+b]\, F(y)$
$\forall y \in [1+a, 1+b]\, F(y)$
$\forall y \in [2+a, 2+b]\, F(y)$

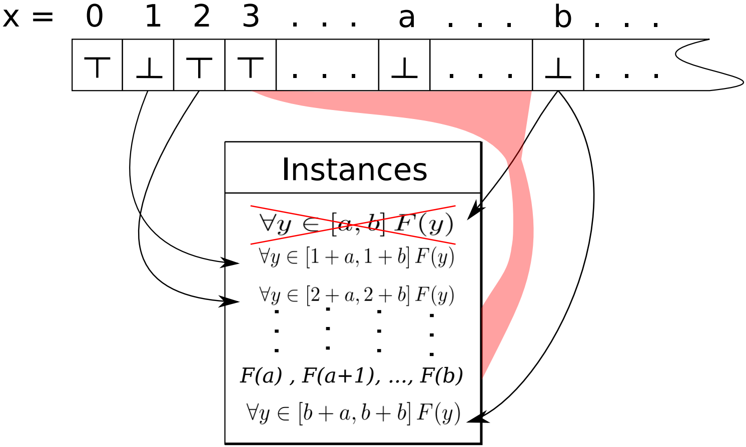
The Problem

Monitor: $\forall_{0\leq x} \forall y \in [x+a, x+b]\, F(y)$



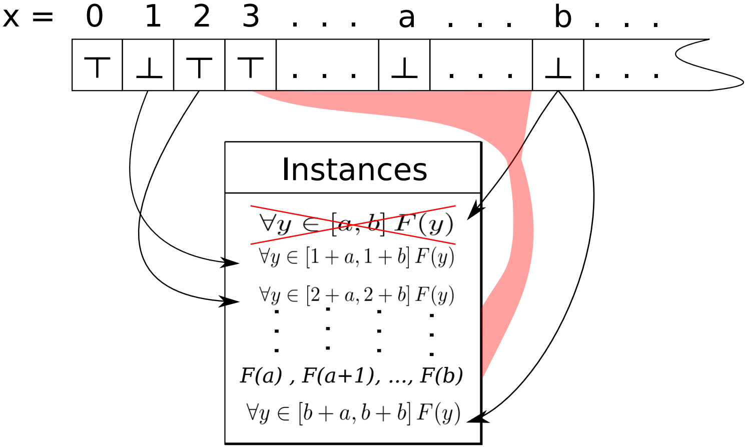
The Problem

Monitor: $\forall_{0\leq x} \forall y \in [x+a, x+b]\, F(y)$



The Problem

Monitor: $\forall_{0\leq x}\forall y \in [x+a, x+b]\, F(y)$



– How large does the instance set gets during evaluation.

Nested Variables

- Getting back to the example monitor, Notice the nested variables:

Nested Variables

- Getting back to the example monitor, Notice the nested variables:

$$\forall_{0 \leq x} : \forall_{y \in [x+1, x+5]} : ((\forall_{z \in [y, x+3]} : \neg @z \; \& \; @z) \; \& \; G(x, y))$$

$$G(x, y) = \forall_{w \in [x+2, y+2]} : (\neg @y \; \& \; (\forall_{m \in [y, w]} : \neg @x \; \& \; @m))$$

Nested Variables

- Getting back to the example monitor, Notice the nested variables:

$$\forall_{0\leq x} : \forall_{y\in[x+1,x+5]} : ((\forall_{z\in[y,x+3]} : \neg @z \;\&\; @z) \;\&\; G(x,y))$$
$$G(x,y) = \forall_{w\in[x+2,y+2]} : (\neg @y \;\&\; (\forall_{m\in[y,w]} : \neg @x \;\&\; @m))$$

- In [Cerna et al. 2016] we developed the concept of dominating monitor tranformation to remove nested variables.
- The following relationship holds between monitors and there dominated counterparts:

Nested Variables

- Getting back to the example monitor, Notice the nested variables:

$$\forall_{0\leq x} : \forall_{y\in[x+1,x+5]} : ((\forall_{z\in[y,x+3]} : \neg @z \ \& \ @z) \ \& \ G(x,y))$$
$$G(x,y) = \forall_{w\in[x+2,y+2]} : (\neg @y \ \& \ (\forall_{m\in[y,w]} : \neg @x \ \& \ @m))$$

- In [Cerna et al. 2016] we developed the concept of dominating monitor tranformation to remove nested variables.
- The following relationship holds between monitors and there dominated counterparts:

Theorem

Let $M \in \mathbb{M}$. Then for all $p, n, S, S' \in \mathbb{N}$ and $s \in \{\top, \bot\}^{\omega}$ such that $T(M) \multimap_{p,s,n} S$ and $T(D(M)) \multimap_{p,s,n} S'$, we have $S \leq S'$.

Dominating Monitor Example

— The dominating monitor of

$$\forall_{0 \leq x} : \forall_{y \in [x+1, x+5]} : ((\forall_{z \in [y, x+3]} : \neg @z \;\&\; @z) \;\&\; G(x, y))$$
$$G(x, y) = \forall_{w \in [x+2, y+2]} : (\neg @y \;\&\; (\forall_{m \in [y, w]} : \neg @x \;\&\; @m))$$

is the following monitor

$$\forall_{0 \leq x} : \forall_{y \in [x+1, x+5]} : ((\forall_{z \in [x+1, x+3]} : \neg @z \;\&\; @z) \;\&\; G(x, y))$$
$$G(x, y) = \forall_{w \in [x+2, x+7]} : (\neg @y \;\&\; (\forall_{m \in [x+1, x+7]} : \neg @x \;\&\; @m))$$

Dominating Monitor Example

— The dominating monitor of

$$\forall_{0\leq x} : \forall_{y\in[x+1,x+5]} : ((\forall_{z\in[y,x+3]} : \neg @z \;\&\; @z) \;\&\; G(x,y))$$
$$G(x,y) = \forall_{w\in[x+2,y+2]} : (\neg @y \;\&\; (\forall_{m\in[y,w]} : \neg @x \;\&\; @m))$$

is the following monitor

$$\forall_{0\leq x} : \forall_{y\in[x+1,x+5]} : ((\forall_{z\in[x+1,x+3]} : \neg @z \;\&\; @z) \;\&\; G(x,y))$$
$$G(x,y) = \forall_{w\in[x+2,x+7]} : (\neg @y \;\&\; (\forall_{m\in[x+1,x+7]} : \neg @x \;\&\; @m))$$

— We retrain from going into the details of the transformation and will only use dominating monitors for the rest of this talk.

Dealing with the Runtime Representation Size

- Runtime representation size equals instances kept in memory while evaluating a monitor.

Dealing with the Runtime Representation Size

- Runtime representation size equals instances kept in memory while evaluating a monitor.
- Consider the following simple monitor:

$$\forall_{0 \leq x} : \forall_{y \in [x, x+4]} : \forall_{z \in [x, x+4]} : \forall_{r \in [x, x+4]} : @r$$

Dealing with the Runtime Representation Size

- Runtime representation size equals instances kept in memory while evaluating a monitor.
- Consider the following simple monitor:

$$\forall_{0 \leq x} : \forall_{y \in [x, x+4]} : \forall_{z \in [x, x+4]} : \forall_{r \in [x, x+4]} : @r$$

- We can simplify its representation:

$$[0, 4]\ [0, 4]\ [0, 4]$$

Dealing with the Runtime Representation Size

- Runtime representation size equals instances kept in memory while evaluating a monitor.
- Consider the following simple monitor:

$$\forall_{0 \leq x} : \forall_{y \in [x, x+4]} : \forall_{z \in [x, x+4]} : \forall_{r \in [x, x+4]} : @r$$

- We can simplify its representation:

$$[0, 4]\ [0, 4]\ [0, 4]$$

- Now let us consider its behaviour as it is evaluated.

Initial State

<u>Initial state</u>

[0,4][0,4][0,4]:1

[0,4][0,4]:0

[0,4]:0

Evaluation

Initial state

[0,4][0,4][0,4]:1

[0,4][0,4]:0

[0,4]:0

Step 0

[1,4][0,4][0,4]:1

[1,4][0,4]:1

[1,4]:1

Step 1

[2,4][0,4][0,4]:1

[2,4][0,4]:2

[2,4]:4

Step 2

[3,4][0,4][0,4]:1

[3,4][0,4]:3

[3,4]:9

Step 3

[4,4][0,4][0,4]:1

[4,4][0,4]:4

[4,4]:16

Step 4

[4,4][0,4][0,4]:0 (1)

[3,4][0,4]:0 (5)

[3,4]:0 (25)

Evaluation

Initial state

[0,4][0,4][0,4]:1
[0,4][0,4]:0
[0,4]:0

→

Step 0

[1,4][0,4][0,4]:1
[1,4][0,4]:1
[1,4]:1

←

Step 1

[2,4][0,4][0,4]:1
[2,4][0,4]:2
[2,4]:4

→

Step 2

[3,4][0,4][0,4]:1
[3,4][0,4]:3
[3,4]:9

←

Step 3

[4,4][0,4][0,4]:1
[4,4][0,4]:4
[4,4]:16

→

Step 4

[4,4][0,4][0,4]:0 (1)
[3,4][0,4]:0 (5)
[3,4]:0 (25)

- Notice that we did not add new instances.
- How does this relate to true evaluation?

Instance to Position Mapping

— It turns out that there is a mapping from the evaluation of a single instance at various positions to the evaluation of multiple instances at a single position.

Instance 4 at position 4

[5,8][4,8][4,8]:1

[5,8][4,8]:1

[5,8]:1

→

Instance 0 at position 0

[1,4][0,4][0,4]:1

[1,4][0,4]:1

[1,4]:1

Instance to Position Mapping

Instance 5 at position 4

[5,9][5,9][5,9]:1

[5,9][5,9]:0

[5,9]:0

Instance 4 at position 4

[5,8][4,8][4,8]:1

[5,8][4,8]:1

[5,8]:1

Instance 3 at position 4

[5,7][3,7][3,7]:1

[5,7][3,7]:2

[5,7]:4

Instance 2 at position 4

[5,6][2,6][2,6]:1

[5,6][2,6]:3

[5,6]:9

Instance 1 at position 4

[5,5][1,5][1,5]:1

[5,5][1,5]:4

[5,5]:16

Instance 0 at position 4

[4,4][0,4][0,4]:0 (1)

[4,4][0,4]:0 (5)

[4,4]:0 (25)

Instance to Position Mapping

Instance 5 at position 4

[5,9][5,9][5,9]:1

[5,9][5,9]:0

[5,9]:0

Instance 4 at position 4

[5,8][4,8][4,8]:1

[5,8][4,8]:1

[5,8]:1

Instance 3 at position 4

[5,7][3,7][3,7]:1

[5,7][3,7]:2

[5,7]:4

Instance 2 at position 4

[5,6][2,6][2,6]:1

[5,6][2,6]:3

[5,6]:9

Instance 1 at position 4

[5,5][1,5][1,5]:1

[5,5][1,5]:4

[5,5]:16

Instance 0 at position 4

[4,4][0,4][0,4]:0 (1)

[4,4][0,4]:0 (5)

[4,4]:0 (25)

— Notice that going to the next position does not change anything

Instance to Position Mapping, Next Position

Instance 6 at position 5

[6,10][6,10][6,10]:1

[6,10][6,10]:0

[6,10]:0

Instance 5 at position 5

[6,9][5,9][5,9]:1

[6,9][5,9]:1

[6,9]:1

Instance 4 at position 5

[6,8][4,8][4,8]:1

[6,8][4,8]:2

[6,8]:4

Instance 3 at position 5

[6,7][3,7][3,7]:1

[6,7][3,7]:3

[6,7]:9

Instance 2 at position 5

[6,6][2,6][2,6]:1

[6,6][2,6]:4

[6,6]:16

Instance 1 at position 5

[5,5][1,5][1,5]:0 (1)

[5,5][1,5]:0 (5)

[5,5]:0 (25)

Instance to Position Mapping, Next Position

Instance 6 at position 5

[6,10][6,10][6,10]:1

[6,10][6,10]:0

[6,10]:0

Instance 5 at position 5

[6,9][5,9][5,9]:1

[6,9][5,9]:1

[6,9]:1

Instance 4 at position 5

[6,8][4,8][4,8]:1

[6,8][4,8]:2

[6,8]:4

Instance 3 at position 5

[6,7][3,7][3,7]:1

[6,7][3,7]:3

[6,7]:9

Instance 2 at position 5

[6,6][2,6][2,6]:1

[6,6][2,6]:4

[6,6]:16

Instance 1 at position 5

[5,5][1,5][1,5]:0 (1)

[5,5][1,5]:0 (5)

[5,5]:0 (25)

— Essentially, we only need to look at the behaviour of one instance up to the largest upper bound. This is the key to the algorithm.

First step towards Algorithmic Space Complexity

– The above concept translates to the following algorithm (assuming no variable nesting).

function $\mathrm{SR}(\langle A, a, b, Q\rangle)$
    if $A = \infty$ **then**
        return ∞
    else
        return $\sum_{i=0}^{A-1} \mathrm{SR}(\langle A, a, b, Q\rangle, i)$
    end if
end function

First step towards Algorithmic Space Complexity

– The highlighted object is a representation of a monitor specification.

function $\mathrm{SR}(\ \langle A, a, b, Q\rangle)$
    if $A = \infty$ **then**
        return ∞
    else
        return $\sum_{i=0}^{A-1}$ **SR**$(\langle A, a, b, Q\rangle\ , i)$
    end if
end function

First step towards Algorithmic Space Complexity

- The highlighted object is the largest upper bound.

function $\mathrm{SR}(\ \langle A, a, b, Q\rangle)$
  if $A = \infty$ **then**
    return ∞
  else
    return $\sum_{i=0}^{A-1}$ **SR**$(\langle A, a, b, Q\rangle\ , i)$
  end if
end function

Constructing a Quantifier Tree

- Consider the following monitor specification:

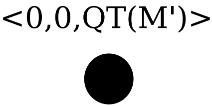
$$M = \forall_{0 \leq x} : \forall_{y \in [x+1, x+5]} : ((\forall_{z \in [x+1, x+3]} : \neg @z \ \& \ @z) \ \& \ G(x, y))$$

$$G(x, y) = \forall_{w \in [x+2, x+7]} : (\neg @y \ \& \ (\forall_{m \in [x+1, x+7]} : \neg @x \ \& \ @m))$$

- A quantifier tree of M is constructed as follows:

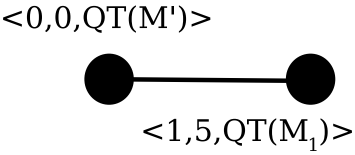
Constructing a Quantifier Tree

$QT(\forall_{0 \leq x} : M') = (0, 0, QT(M'))$ where $M' = \forall_{y \in [x+1, x+5]} : ((\forall_{z \in [x+1, x+3]} : \neg @z \; \& \; @z) \; \& \; G(x, y))$



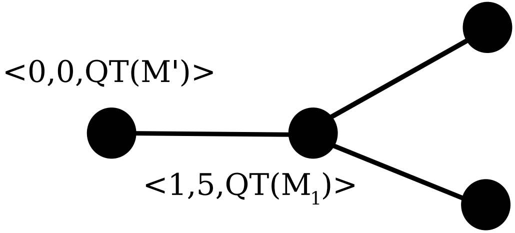
Constructing a Quantifier Tree

$QT(M') = (1, 5, QT(M_1))$ where $M_1 = (\forall_{z \in [x+1, x+3]} : \neg @z \ \& \ @z) \ \& \ G(x, y)$



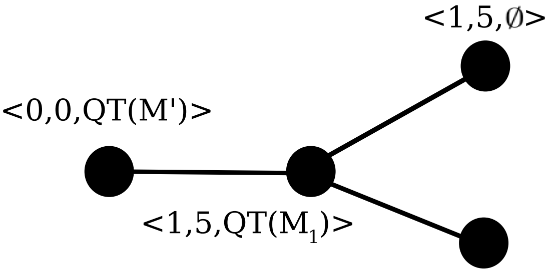
Constructing a Quantifier Tree

$QT(M_1) = QT(M_l) \cup QT(M_r)$ where $M_l{=}(\forall_{z\in[x+1,x+3]}{:}\neg @z \ \&\ @z)$ and $M_r{=}G(x,y)$



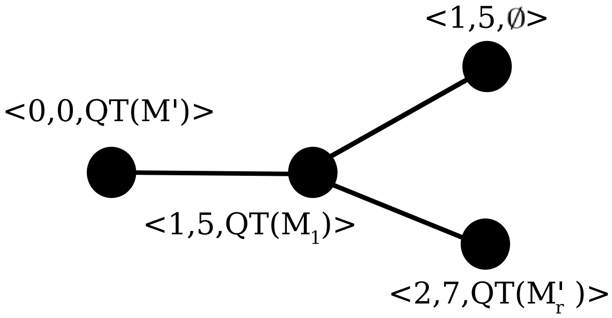
Constructing a Quantifier Tree

$$QT(M_I) = (1, 5, \emptyset)$$



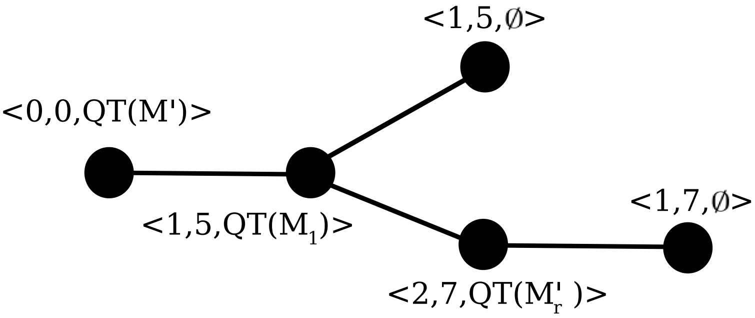
Constructing a Quantifier Tree

$QT(M_r) = (2, 7, QT(M'_r))$ where
$M'_r = \neg @y \;\&\; (\forall_{m \in [x+1, x+7]} : \neg @x \;\&\; @m)$



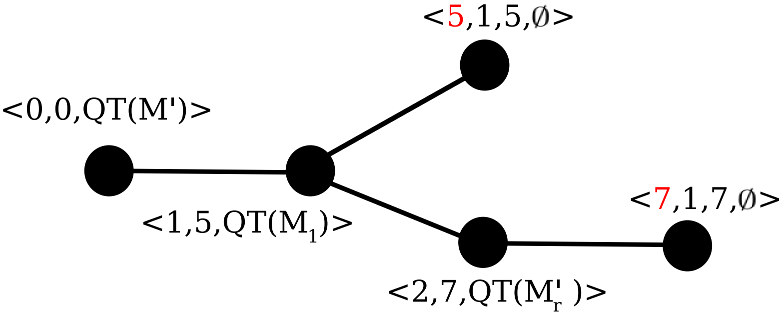
Constructing a Quantifier Tree

$$QT(M'_r) = (1, 7, \emptyset)$$

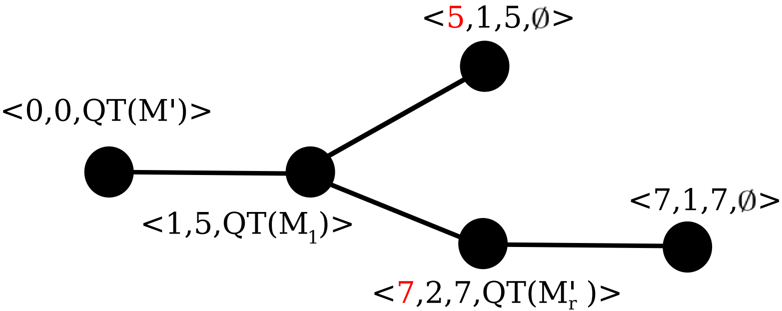


– This is the quantifier tree of monitor M.

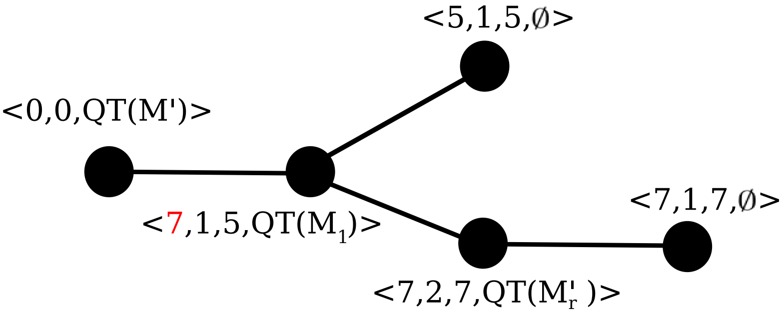
Annotating the Trees



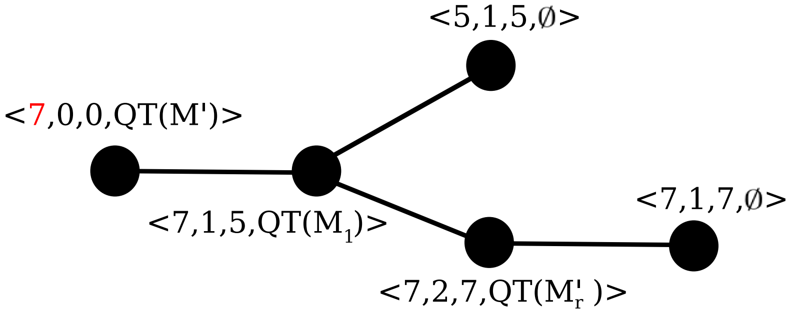
Annotating the Trees



Annotating the Trees



Annotating the Trees



– We will refer to this quantifier tree as $QT(M)$. Now we compute

$$\sum_{i=0}^{6} \mathrm{SR}(QT(M), i).$$

Computing $\mathrm{SR}(QT(M), i)$

– Rather then computing the entire sum

$$\sum_{i=0}^{6} \mathrm{SR}(QT(M), i).$$

We will look into a specific example.

– $\mathrm{SR}(QT(M), 5)$
We will also ignore the first node $\langle 7, 0, 0, QT(M') \rangle$

Computing SR($\langle 7, 1, 5, QT(M_1) \rangle$, 5)

– At position 5 the whole interval will unroll.

$$\langle 7, 1, 5, QT(M_1) \rangle = \begin{cases} \langle 7, 1, 1, QT(M_1) \rangle \\ \langle 7, 2, 2, QT(M_1) \rangle \\ \langle 7, 3, 3, QT(M_1) \rangle \\ \langle 7, 4, 4, QT(M_1) \rangle \\ \langle 7, 5, 5, QT(M_1) \rangle \end{cases}$$

– The number of generated instances is computed using the following formula:

$$1 + \min\{i, b\} - a$$

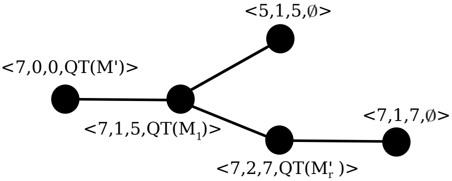
which in our case is $1 + \min\{5, 5\} - 1 = 5$.

Dealing with instances

- The specific instances don't really matter.
- We can just write the following

$$5 \cdot SR(QT(M_1), 5)$$

- But notice that $QT(M_1)$ branches.



The Optimization and Branching

- Normally $5 \cdot SR(QT(M_1), 5) = 5 \cdot (SR(QT(M_l), 5) + SR(QT(M_r), 5))$.
- However $QT(M_l) = \langle 5, 1, 5, \emptyset \rangle$, the upper bound is equal to the position.
- This means $SR(QT(M_l), 5) = 0$, and we optimize the computation by ignoring it. Thus,

$$5 \cdot SR(QT(M_1), 5) = 5 \cdot (SR(QT(M_l), 5) + SR(QT(M_r), 5)) =$$

$$5 \cdot SR(QT(M_r), 5)$$

Computing the Right Branch

- Moving on to $\langle 7, 2, 7, QT(M_r') \rangle$ we compute the interval size as

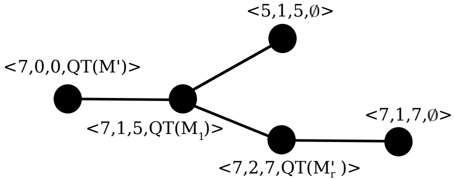
$$1 + \min\{5, 7\} - 2 = 4$$

- Thus, we get

$$5 \cdot SR(QT(M_r), 5) = 20 \cdot SR(QT(M_r'), 5)$$

- As the last step we get

$$SR(QT(M), 5) = 20 \cdot SR(QT(M_r'), 5) = 20 \cdot 5 = 100$$



Algorithm

– The algorithm is as follows:

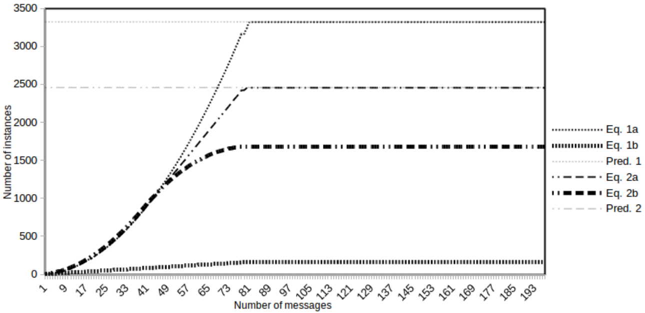
```
function SR((A, a, b, Q),i)
    cil ← 1 + min {i, b} − a
    if cil ≤ 0 & b ≥ a then
        return 1
    else
        return 0
    end if
    if  i ≥ b  then
        inst ← 0
    else
        inst ← 1
    end if
    for all  aqt′ = (A′, a′, b′, Q′) ∈ Q do
        if i < A′ then
            inst ← inst + cil · SR(aqt′, i)
        end if
    end for
    return inst
end function
```

– It has a running time of $O(n)$ in terms of formula size.

Experimental Results: Artificial

– We ran the algorithm on the following monitor specifications:

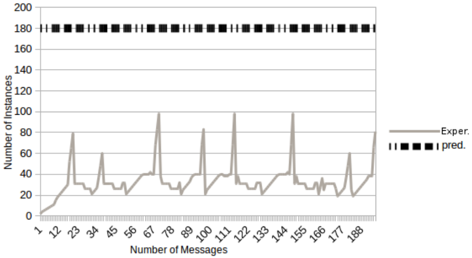
$$\forall_{0\leq x} : \forall_{y\in[x,x+80]} : \forall_{z\in[x,x+80]} : @z \quad (1a)$$
$$\forall_{0\leq x} : \forall_{y\in[x,x+80]} : \forall_{z\in[x,y]} : @z \quad (1b)$$
$$\forall_{0\leq x} : \forall_{y\in[x,x+40]} : \forall_{z\in[x,x+80]} : @z \quad (2a)$$
$$\forall_{0\leq x} : \forall_{y\in[x,x+40]} : \forall_{z\in[x,y+40]} : @z \quad (2b)$$



Experimental Results: Realistic

– We ran the algorithm on the following monitor written in the full specification language:

```
type int; type message; stream<int> IP;
stream<int> S = stream<IP> x satisfying @x>=0 :
      value[seq,@x,plus]<IP> y with x < _ <=# x+10000: @y;
monitor<S> M = monitor<S> x :
      forall<S> y with x < _ <=# x+15000:
        exists<S> z with y < _ <=# y+4000: IsEven(#z);
```



What is Next?

- The Run time representation size for general monitor specifications is bounded by our algorithm.
  - Dealing with nested variables would provide precise results for all monitor specifications
  - Currently we are investigating the implications of these results for writing monitor specifications.
  - Looking for more optimal ways of writing monitor specifications.
- The next measure we are going to tackle concerning logical guard is the number of stream accesses per message.

Thank you for your time.