Space Analysis of a Predicate Logic Fragment for the Specification of Stream Monitors

David M. Cerna, Wolfgang Schreiner, and Temur Kutsia

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

March 29^{th}, 2016

JOHANNES KEPLER
UNIVERSITY LINZ

Introduction

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

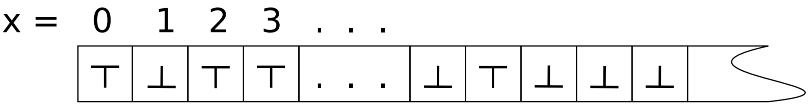
Introduction

- 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]
- In this work, we focus on the space complexity of the instance set.

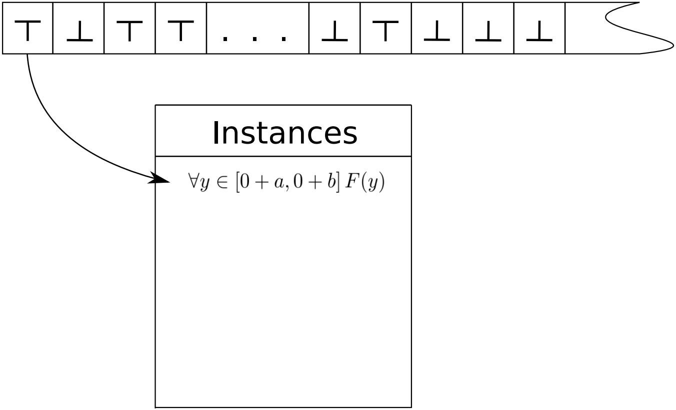
The Problem



The Problem

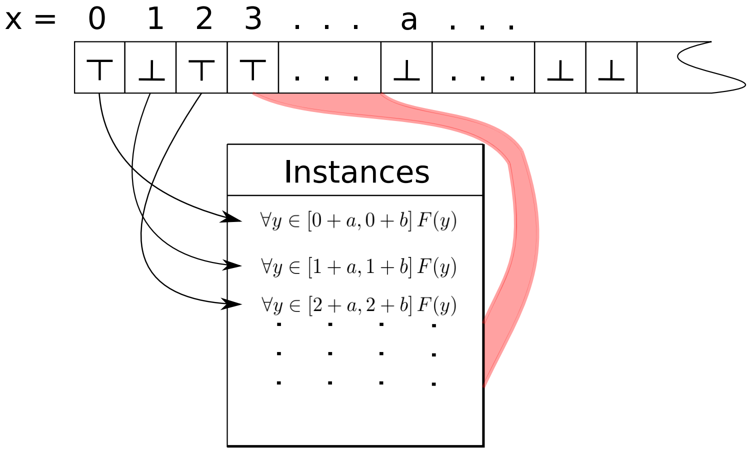
Monitor: $\forall_{0 \le x} \forall y \in [x+a, x+b]\, F(y)$

x = 0 1 2 3 . . .



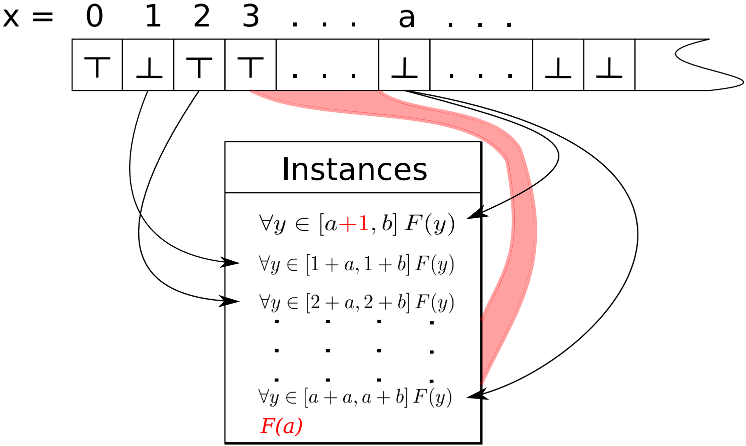
The Problem

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



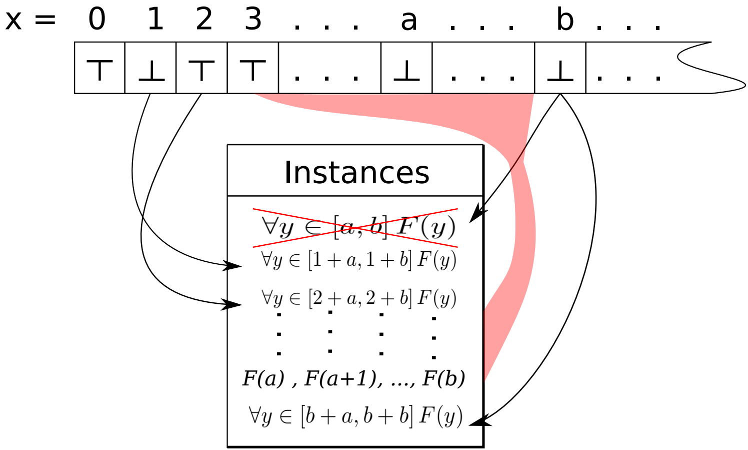
The Problem

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



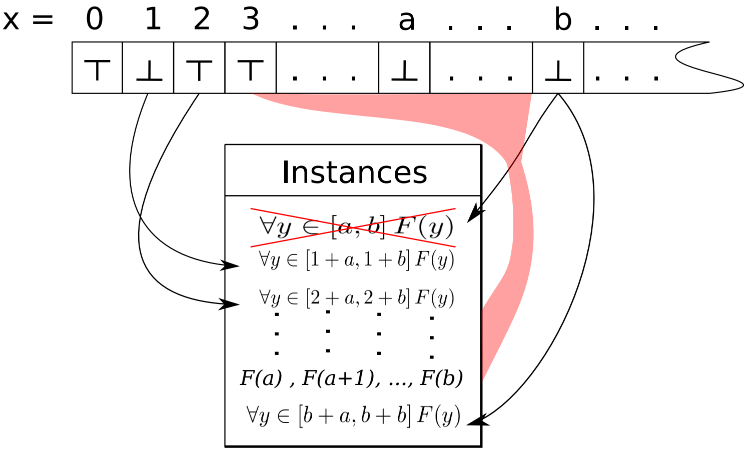
The Problem

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



The Problem

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



– We want to know how large the instance set gets during evaluation.

Outline

- Abstraction of the specification language.
- Background required to understand the results.
- Simplified operational semantics for the our abstraction.
- Results concerning our abstraction and evaluation method.
- Use the results to produce a bounding function for a much larger fragment of the LogicGuard specification language.
- Conclusion and future work.

Basic Idea

- Precise space analysis of the entire core language is quite difficult.
- Many individual cases to consider.
- Large formulas can allow for complex variable interaction.

Basic Idea

- Precise space analysis of the entire core language is quite difficult.
- Many individual cases to consider.
- Large formulas can allow for complex variable interaction.

$$
\begin{array}{lcl}
M & ::= & \forall_{0 \leq V} : F. \\
F & ::= & @V \Big| \neg F \Big| F \mathbin{\&} F \Big| F \wedge F \Big| \forall_{V \in [B,B]} : F. \\
B & ::= & \infty | 0 | V | B + N | B - N \ . \\
V & ::= & x | y | z | \ldots \ . \\
N & ::= & 0 | 1 | 2 | \ldots \ .
\end{array}
$$

Basic Idea

- Precise space analysis of the entire core language is quite difficult.
- Many individual cases to consider.
- Large formulas can allow for complex variable interaction.

M	$::=$	$\forall_{0 \leq V} : F.$	M	$::=$	$\forall_{0 \leq V} : F.$
F	$::=$	$@V \big| \neg F \big| F \,\&\, F \big| F \wedge F \big| \forall_{V \in [B,B]} : F.$	F	$::=$	$@V \big| \neg F \big| F \,\&\, F \big| F \wedge F \big| \forall_{V \in [B,B]} : F.$
B	$::=$	$\infty \| 0 \| V \| B + N \| B - N$.	B	$::=$	$V \| B + N$.
V	$::=$	$x \| y \| z \| \ldots$.	V	$::=$	$x \| y \| z \| \ldots$.
N	$::=$	$0 \| 1 \| 2 \| \ldots$.	N	$::=$	$0 \| 1 \| 2 \| \ldots$.

- Removal of constants provides uniformity.
- Variable definition nesting can still result in complex structure.

Basic Idea:1

- We focus on a very simple class of monitors.

Basic Idea:1

- We focus on a very simple class of monitors.
  - Deriving precise bounds for this simple class is easier.

Basic Idea:1

- We focus on a very simple class of monitors.
  - Deriving precise bounds for this simple class is easier.
- We can use the solutions for the simple cases as an invariant for a recursive function.
- The function will be inductively defined over the formula structure.

Basic Idea:1

- We focus on a very simple class of monitors.
  - Deriving precise bounds for this simple class is easier.
- We can use the solutions for the simple cases as an invariant for a recursive function.
- The function will be inductively defined over the formula structure.
- The following assumptions are made.

Basic Idea:2

- We assume that the given formula is a sentence, i.e. no free variables.
- Propositional formulas evaluate instantly when the needed positions of the stream are available.
- Quantifier bounds only contain the stream variable.

Basic Idea:2

- We assume that the given formula is a sentence, i.e. no free variables.
- Propositional formulas evaluate instantly when the needed positions of the stream are available.
- Quantifier bounds only contain the stream variable.
- The last assumption is easier to give by example.

Basic Idea:3

- Consider the following:

$$\forall_{0\leq x}:\Big(\forall_{y\in[x,x+5]}:\Big(@x \ \& \ \Big(\forall_{z\in[x+1,x+2]}:(@z \ \& \ @y)\Big)\Big)\Big)$$

Basic Idea:3

- Consider the following:

$$\forall_{0\leq x}:\left(\forall_{y\in[x,x+5]}:\left(@x \ \& \ \left(\forall_{z\in[x+1,x+2]}:(@z \ \& \ @y)\right)\right)\right)$$

- We also “drop” nested quantifiers using the following method.

Basic Idea:3

- Consider the following:

$$\forall_{0\le x}:\left(\forall_{y\in[x,x+5]}:\left(@x \ \& \ \left(\forall_{z\in[x+1,x+2]}:(@z \ \& \ @y)\right)\right)\right)$$

- We also "drop" nested quantifiers using the following method.

$$\forall_{0\le x}:\left(@x \ \& \ \left(\forall_{z\in[x+1,x+2]}:\left(@z \ \& \ \left(\forall_{w\in[x+5,x+5]}:@w\right)\right)\right)\right)$$

Basic Idea:3

— Consider the following:

$$\forall_{0\le x}:\Big(\forall_{y\in[x,x+5]}:\Big(@x \ \& \ \Big(\forall_{z\in[x+1,x+2]}:(@z \ \& \ @y)\Big)\Big)\Big)$$

— We also "drop" nested quantifiers using the following method.

$$\forall_{0\le x}:\Big(@x \ \& \ \Big(\forall_{z\in[x+1,x+2]}:\Big(@z \ \& \ \Big(\forall_{w\in[x+5,x+5]}:@w\Big)\Big)\Big)\Big)$$
$$\forall_{0\le x}:\Big(@x \ \& \ \Big(\forall_{z\in[x+1,x+2]}:\Big(@z \ \& \ F'[(\forall_{w\in[x+5,x+5]}:@w),x,z]\Big)\Big)\Big)$$

Basic Idea:3

— Consider the following:

$$\forall_{0\leq x}:\Big(\forall_{y\in[x,x+5]}:\Big(@x \ \& \ \Big(\forall_{z\in[x+1,x+2]}:(@z \ \& \ @y)\Big)\Big)\Big)$$

— We also “drop” nested quantifiers using the following method.

$$\forall_{0\leq x}:\Big(@x \ \& \ \Big(\forall_{z\in[x+1,x+2]}:\Big(@z \ \& \ \Big(\forall_{w\in[x+5,x+5]}:@w\Big)\Big)\Big)\Big)$$

$$\forall_{0\leq x}:\Big(@x \ \& \ \Big(\forall_{z\in[x+1,x+2]}:\Big(@z \ \& \ F'[(\forall_{w\in[x+5,x+5]}:@w),x,z]\Big)\Big)\Big)$$

$$\forall_{0\leq x}:F[(\forall_{z\in[x+1,x+2]}:@z \ \& \ F'[(\forall_{w\in[x+5,x+5]}:@w),x,z]),x]$$

Basic Idea:3

- Consider the following:

$$\forall_{0\le x}:\Big(\forall_{y\in[x,x+5]}:\big(@x \ \& \ \big(\forall_{z\in[x+1,x+2]}:(@z \ \& \ @y)\big)\big)\Big)$$

- We also "drop" nested quantifiers using the following method.

$$\forall_{0\le x}:\Big(@x \ \& \ \big(\forall_{z\in[x+1,x+2]}:\big(@z \ \& \ \big(\forall_{w\in[x+5,x+5]}:@w\big)\big)\big)\Big)$$
$$\forall_{0\le x}:\Big(@x \ \& \ \big(\forall_{z\in[x+1,x+2]}:\big(@z \ \& \ F'[(\forall_{w\in[x+5,x+5]}:@w),x,z]\big)\big)\Big)$$
$$\forall_{0\le x}:F[(\forall_{z\in[x+1,x+2]}:@z \ \& \ F'[(\forall_{w\in[x+5,x+5]}:@w),x,z]),x]$$

$$\langle 1, 2, 5\rangle_{\mathbb{N}}$$

Basic Idea:3

- Consider the following:

$$\forall_{0\leq x}:\big(\forall_{y\in[x,x+5]}:\big(@x \ \& \ \big(\forall_{z\in[x+1,x+2]}:(@z \ \& \ @y)\big)\big)\big)$$

- We also "drop" nested quantifiers using the following method.

$$\forall_{0\leq x}:\big(@x \ \& \ \big(\forall_{z\in[x+1,x+2]}:\big(@z \ \& \ \big(\forall_{w\in[x+5,x+5]}:@w\big)\big)\big)\big)$$
$$\forall_{0\leq x}:\big(@x \ \& \ \big(\forall_{z\in[x+1,x+2]}:\big(@z \ \& \ F'[(\forall_{w\in[x+5,x+5]}:@w),x,z]\big)\big)\big)$$
$$\forall_{0\leq x}:F[(\forall_{z\in[x+1,x+2]}:@z \ \& \ F'[(\forall_{w\in[x+5,x+5]}:@w),x,z]),x]$$

$$\langle 1,2,5\rangle_{\mathbb{N}}$$

- We refer to the last object as an $\mathbb{N}$-triple.
- The formulas represented by $\mathbb{N}$-triples are referred to as the restricted fragment.

Basic Idea:4

— $\mathbb{N}$-triples essentially represent sets of formulas.

Basic Idea:4

- $\mathbb{N}$-triples essentially represent sets of formulas.
- Also, $\mathbb{N}$-triples are independent of the variables used.
  - This is a side effect of bounds containing the stream variable only
- The set of formulas an $\mathbb{N}$-triple $\langle a, b, c\rangle_{\mathbb{N}}$ represents can be written as follows:

$$\forall_{0\leq x}{:}F[(\forall_{z\in[x+a,x+b]}{:}@z \ \& \ F'[(\forall_{w\in[x+c,x+c]}{:}@w),x,z]),x]$$

Basic Idea:4

- $\mathbb{N}$-triples essentially represent sets of formulas.
- Also, $\mathbb{N}$-triples are independent of the variables used.
  - This is a side effect of bounds containing the stream variable only
- The set of formulas an $\mathbb{N}$-triple $\langle a, b, c\rangle_{\mathbb{N}}$ represents can be written as follows:

$$\forall_{0\leq x}{:}F[(\forall_{z\in[x+a,x+b]}{:}@z \;\&\; F'[(\forall_{w\in[x+c,x+c]}{:}@w),x,z]),x]$$

- An instance of an $\mathbb{N}$-triple given the position n for the stream variable is $\langle a, b, c\rangle_{\mathbb{N}}(n)$ and the set can be written as follows:

$$F[(\forall_{z\in[n+a,n+b]}{:}@z \;\&\; F'[(\forall_{w\in[n+c,n+c]}{:}@w),n,z]),n]$$

Basic Idea:5

Today we will address the following question: Given an $\mathbb{N}$-triple $\langle a, b, c\rangle_{\mathbb{N}}$ and an external stream S starting at some value α, how many instances do we need to keep in memory when evaluating $\langle a, b, c\rangle_{\mathbb{N}}$ on S starting at α?

Core Language Evaluation

- $\mathbb{N}$-triples are extremely simple compared to sentences of the core language.
- Evaluation of $\mathbb{N}$-triples only requires a fragment of the evaluation rules used for sentences of the core language.
- Most of the reduction in the number of rules concerns the removal of propositional structure from $\mathbb{N}$-triples.

Core Language Evaluation

- $\mathbb{N}$-triples are extremely simple compared to sentences of the core language.
- Evaluation of $\mathbb{N}$-triples only requires a fragment of the evaluation rules used for sentences of the core language.
- Most of the reduction in the number of rules concerns the removal of propositional structure from $\mathbb{N}$-triples.
- Evaluation of monitors written using the core language is done by a small step operational semantics.

$$\forall^{IS}_{0 \le x} : f \rightarrow_{p,MS,m,RS} \forall^{IS'}_{0 \le x} : f$$

- The transition from IS to IS' is defined as a formula transition relation:

$$f \rightarrow_{p,MS,m,c} f'$$

Formula Transition Relation

#	Transition	Constraints
Atomic Formulas		
A1	$\mathbf{n}(@y) \to \mathbf{d}(c.2(y)))$	$y \in dom(c.2)$
A2	$\mathbf{n}(@y) \to \mathbf{d}(\bot)$	$y \notin dom(c.2)$
Negation		
N1	$\mathbf{n}(\neg f) \to \mathbf{n}(\neg \mathbf{n}(f'))$	$f \to \mathbf{n}(f')$
N2	$\mathbf{n}(\neg f) \to \mathbf{d}(\bot)$	$f \to \mathbf{d}(\top)$
N3	$\mathbf{n}(\neg f) \to \mathbf{d}(\top)$	$f \to \mathbf{d}(\bot)$
Sequential conjunction		
C1	$\mathbf{n}(f_1 \ \& \ f_2) \to \mathbf{n}(\mathbf{n}(f_1') \ \& \ f_2)$	$f_1 \to \mathbf{n}(f_1')$
C2	$\mathbf{n}(f_1 \ \& \ f_2) \to \mathbf{d}(\bot)$	$f_1 \to \mathbf{d}(\bot)$
C3	$\mathbf{n}(f_1 \ \& \ f_2) \to \mathbf{n}(f_2')$	$f_1 \to \mathbf{d}(\top)$, $f_2 \to \mathbf{n}(f_2')$
Quantification		
Q1	$\forall_{y \in [b_1, b_2]} : f \to \mathbf{d}(\top)$	$p_1 = b_1(c)$, $p_2 = b_2(c)$, $p_1 > p_2 \lor p_1 = \infty$
Q2	$\forall_{y \in [b_1, b_2]} : f \to F'$	$p_1 = b_1(c)$, $p_2 = b_2(c)$, $p_1 \neq \infty$, $p_1 \leq p_2$, $\mathbf{n}(\forall_{y \in [p_1, p_2]} : f) \to F'$
Q3	$\mathbf{n}(\forall_{y \in [p_1, p_2]} : f) \to \mathbf{n}(\forall_{y \in [p_1, p_2]} : f)$	$p < p_1$
Q4	$\mathbf{n}(\forall_{y \in [p_1, p_2]} : f) \to F'$	$p_1 \leq p$, $\mathbf{n}(\forall^{IS^f}_{y \leq p_2} : f) \to F'$
Q5	$\mathbf{n}(\forall^{IS^f}_{y \leq p_2} : f) \to \mathbf{d}(\bot)$	DF
Q6	$\mathbf{n}(\forall^{IS^f}_{y \leq p_2} : f) \to \mathbf{d}(\top)$	$\neg DF$, $IS^f_1 = \emptyset$, $p_2 < p$
Q7	$\mathbf{n}(\forall^{IS^f}_{y \leq p_2} : f) \to \mathbf{n}(\forall^{IS^f_1}_{y \leq p_2} : f)$	$\neg DF$, $(IS^f_1 \neq \emptyset \lor p \leq p_2)$

Defining $\mathbb{N}$-triple Evaluation

– For $\mathbb{N}$-triple evaluation we only need to consider the quantifier rules of core language formula evaluation.

Quantification		
Q1	$\forall_{y\in[b_1,b_2]} : f \to \mathbf{d}(\top)$	$p_1 = b_1(c)$, $p_2 = b_2(c)$, $p_1 > p_2 \vee p_1 = \infty$
Q2	$\forall_{y\in[b_1,b_2]} : f \to F'$	$p_1 = b_1(c)$, $p_2 = b_2(c)$, $p_1 \neq \infty$, $p_1 \leq p_2$, $\mathbf{n}(\forall_{y\in[p_1,p_2]} : f) \to F'$
Q3	$\mathbf{n}(\forall_{y\in[p_1,p_2]} : f) \to \mathbf{n}(\forall_{y\in[p_1,p_2]} : f)$	$p < p_1$
Q4	$\mathbf{n}(\forall_{y\in[p_1,p_2]} : f) \to F'$	$p_1 \leq p$, $\mathbf{n}(\forall^{IS^f}_{y\leq p_2} : f) \to F'$
Q5	$\mathbf{n}(\forall^{IS^f}_{y\leq p_2} : f) \to \mathbf{d}(\bot)$	DF
Q6	$\mathbf{n}(\forall^{IS^f}_{y\leq p_2} : f) \to \mathbf{d}(\top)$	$\neg DF$, $IS^f_1 = \emptyset$, $p_2 < p$
Q7	$\mathbf{n}(\forall^{IS^f}_{y\leq p_2} : f) \to \mathbf{n}(\forall^{IS^f_1}_{y\leq p_2} : f)$	$\neg DF$, $(IS^f_1 \neq \emptyset \vee p \leq p_2)$

Defining $\mathbb{N}$-triple Evaluation

- Though, not all the quantifier rules are needed.

Quantification		
Q1	$\forall_{y\in[b_1,b_2]} : f \rightarrow \mathbf{d}(\top)$	$p_1 = b_1(c)$, $p_2 = b_2(c)$, $p_1 > p_2 \vee p_1 = \infty$
Q3	$\mathbf{n}(\forall_{y\in[p_1,p_2]} : f) \rightarrow \mathbf{n}(\forall_{y\in[p_1,p_2]} : f)$	$p < p_1$
Q5	$\mathbf{n}(\forall^{IS^f}_{y\leq p_2} : f) \rightarrow \mathbf{d}(\perp)$	DF
Q6	$\mathbf{n}(\forall^{IS^f}_{y\leq p_2} : f) \rightarrow \mathbf{d}(\top)$	$\neg DF$, $IS^f_1 = \emptyset$, $p_2 < p$
Q7	$\mathbf{n}(\forall^{IS^f}_{y\leq p_2} : f) \rightarrow \mathbf{n}(\forall^{IS^f_1}_{y\leq p_2} : f)$	$\neg DF$, $(IS^f_1 \neq \emptyset \vee p \leq p_2)$

Evaluation of $\mathbb{N}$-triples

- To apply the above transition rules to $\mathbb{N}$-triples we need the notion of atomic $\mathbb{N}$-triples.
- Also, splitting of $\mathbb{N}$-triples.

Evaluation of $\mathbb{N}$-triples

- To apply the above transition rules to $\mathbb{N}$-triples we need the notion of atomic $\mathbb{N}$-triples.
- Also, splitting of $\mathbb{N}$-triples.

$$\begin{tikzcd} & \langle a+1, b, c\rangle_{\mathbb{N}} \\ \langle a, b, c\rangle_{\mathbb{N}} \arrow[ur, bend left] \arrow[dr, bend right] & \\ & \langle a, a, c\rangle_{\mathbb{N}} \end{tikzcd}$$

Evaluation of $\mathbb{N}$-triples

- To apply the above transition rules to $\mathbb{N}$-triples we need the notion of atomic $\mathbb{N}$-triples.
- Also, splitting of $\mathbb{N}$-triples.

$$\begin{tikzcd} & \langle a+1, b, c\rangle_{\mathbb{N}} \\ \langle a, b, c\rangle_{\mathbb{N}} \arrow[ur, bend left] \arrow[dr, bend right] & \\ & \langle a, a, c\rangle_{\mathbb{N}} \end{tikzcd}$$

$$\langle a, b, c\rangle_{\mathbb{N}} \equiv \langle a+1, b, c\rangle_{\mathbb{N}} \wedge \langle a, a, c\rangle_{\mathbb{N}}$$

Evaluation of $\mathbb{N}$-triples

- To apply the above transition rules to $\mathbb{N}$-triples we need the notion of atomic $\mathbb{N}$-triples.
- Also, splitting of $\mathbb{N}$-triples.

$$\begin{tikzcd} & \langle a+1, b, c\rangle_{\mathbb{N}} \\ \langle a, b, c\rangle_{\mathbb{N}} \arrow[ur, bend left] \arrow[dr, bend right] & \\ & \langle a, a, c\rangle_{\mathbb{N}} \end{tikzcd}$$

$$\langle a, b, c\rangle_{\mathbb{N}} \equiv \langle a+1, b, c\rangle_{\mathbb{N}} \wedge \langle a, a, c\rangle_{\mathbb{N}}$$

- Though, $\langle a, a, c\rangle_{\mathbb{N}}(\alpha)$ is "propositional", it cannot be evaluated till $\alpha + c$ is available.

Evaluation of $\mathbb{N}$-triples

- Essentially as soon as a needed stream position is available we split the triple instances.
- Also as soon as the third component of a triple instance $\langle a, a, c\rangle_{\mathbb{N}}(\alpha)$ is available we remove it from memory.
- We encapsulate these ideas in the following structure:

$$\ldots \xrightarrow{[\alpha,\infty]} \left[n, \langle a, b, c\rangle_{\mathbb{N}}, \mathbf{I}\right] \xrightarrow{[\alpha,\infty]} \left[n+1, \langle a, b, c\rangle_{\mathbb{N}}, \mathbf{I}'\right] \xrightarrow{[\alpha,\infty]} \ldots$$

- n is the initial stream variable position and $\mathbf{I}$ is the initial memory.
- $n+1$ is the new stream variable position and $\mathbf{I}'$ is the new memory state.

Evaluation structure

- An evaluation structure is an object:

$$\left[n, \langle a, b, c\rangle_{\mathbb{N}}, \mathbf{I}\right]$$

- We will start from an initial evaluation structure:

$$\left[\triangleright, \langle a, b, c\rangle_{\mathbb{N}}, \emptyset\right]$$

where $\triangleright$ is the position to left of the interval we are evaluating over.

Evaluation example

– Let us consider the evaluation of $t = \langle 0, 2, 2 \rangle_{\mathbb{N}}$ over the interval $[0, \infty)$ starting at 0.

Evaluation example

- Let us consider the evaluation of $t = \langle 0,2,2\rangle_{\mathbb{N}}$ over the interval $[0,\infty)$ starting at 0.

$$[\rhd, t, \emptyset] \xrightarrow{[0,\infty)} \left[0, t, \left\{ \begin{array}{l} \langle 1,2,2\rangle_{\mathbb{N}}\,(0) \\ \langle 0,0,2\rangle_{\mathbb{N}}\,(0) \end{array} \right\}\right] \xrightarrow{[\alpha,\infty)} \left[1, t, \left\{ \begin{array}{l} \langle 2,2,2\rangle_{\mathbb{N}}\,(0) \\ \langle 1,1,2\rangle_{\mathbb{N}}\,(0) \\ \langle 0,0,2\rangle_{\mathbb{N}}\,(0) \\ \langle 1,2,2\rangle_{\mathbb{N}}\,(1) \\ \langle 0,0,2\rangle_{\mathbb{N}}\,(1) \end{array} \right\}\right]$$

Evaluation example

- Let us consider the evaluation of $t = \langle 0,2,2\rangle_{\mathbb{N}}$ over the interval $[0,\infty)$ starting at 0.

$$[\triangleright, t, \emptyset] \xrightarrow{[0,\infty)} \left[0, t, \left\{ \begin{array}{l} \langle 1,2,2\rangle_{\mathbb{N}}(0) \\ \langle 0,0,2\rangle_{\mathbb{N}}(0) \end{array} \right\}\right] \xrightarrow{[\alpha,\infty)} \left[1, t, \left\{ \begin{array}{l} \langle 2,2,2\rangle_{\mathbb{N}}(0) \\ \langle 1,1,2\rangle_{\mathbb{N}}(0) \\ \langle 0,0,2\rangle_{\mathbb{N}}(0) \\ \langle 1,2,2\rangle_{\mathbb{N}}(1) \\ \langle 0,0,2\rangle_{\mathbb{N}}(1) \end{array} \right\}\right]$$

$$\left[2, t, \left\{ \begin{array}{l} \langle 2,2,2\rangle_{\mathbb{N}}(1) \\ \langle 1,1,2\rangle_{\mathbb{N}}(1) \\ \langle 0,0,2\rangle_{\mathbb{N}}(1) \\ \langle 1,2,2\rangle_{\mathbb{N}}(2) \\ \langle 0,0,2\rangle_{\mathbb{N}}(2) \end{array} \right\}\right] \xrightarrow{[\alpha,\infty)} \left[3, t, \left\{ \begin{array}{l} \langle 2,2,2\rangle_{\mathbb{N}}(2) \\ \langle 1,1,2\rangle_{\mathbb{N}}(2) \\ \langle 0,0,2\rangle_{\mathbb{N}}(2) \\ \langle 1,2,2\rangle_{\mathbb{N}}(3) \\ \langle 0,0,2\rangle_{\mathbb{N}}(3) \end{array} \right\}\right] \xrightarrow{[\alpha,\infty)} \cdots$$

Base Case

- We start by considering $\mathbb{N}$-triples of the form $\langle 0, b, b \rangle_{\mathbb{N}}$.
- These are the simplest because all atomic instances are removed at the same time.
- Also, there is no shift in position

Base Case

- We start by considering $\mathbb{N}$-triples of the form $\langle 0, b, b\rangle_{\mathbb{N}}$.
- These are the simplest because all atomic instances are removed at the same time.
- Also, there is no shift in position
- The following theorem concerns bounding of the instance set:

Theorem

Given an $\mathbb{N}$-triple t of the form $\langle 0, b, b\rangle_{\mathbb{N}}$ and an interval $[\alpha, \beta]$ such that $\alpha \leq b < \beta$, then there exists a value $x \in [\alpha, \beta]$ such that given the complete proper evaluation chain:

$$[\rhd, t, \emptyset] \xrightarrow{[\alpha,\beta]} \cdots \xrightarrow{[\alpha,\beta]} [x-1, t, \mathbf{I}_0] \xrightarrow{[\alpha,\beta]} [x, t, \mathbf{I}_1] \xrightarrow{[\alpha,\beta]} \cdots \xrightarrow{[\alpha,\beta]} [\beta, t, \mathbf{I}_{\beta-x}]$$

the following holds $|\mathbf{I}_0| \neq |\mathbf{I}_1| = \cdots = |\mathbf{I}_{\beta-x}|$.

Base Case

- The instance $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha)$ is the first instance and any sub-instance will not be removed from memory till the position $\alpha + b$. There are b+1 sub-instances of $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha)$ at $\alpha + b - 1$.

Base Case

- The instance $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha)$ is the first instance and any sub-instance will not be removed from memory till the position $\alpha + b$. There are b+1 sub-instances of $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha)$ at $\alpha + b - 1$.
- We also need to count the instances $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha + 1), \cdots,$ $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha + b - 1)$ of which there are

$$\sum_{i=2}^{b} i = \frac{b \cdot (b+1)}{2} - 1.$$

Base Case

- The instance $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha)$ is the first instance and any sub-instance will not be removed from memory till the position $\alpha + b$. There are b+1 sub-instances of $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha)$ at $\alpha + b - 1$.
- We also need to count the instances $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha + 1), \cdots,$ $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha + b - 1)$ of which there are

$$\sum_{i=2}^{b} i = \frac{b \cdot (b+1)}{2} - 1.$$

- At $\alpha + b$ we remove $b + 1$ instances from memory, unroll $b - 1$ new ones, and add two instances for $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha + b)$. Thus the used portion of memory stays the same.

Base Case

- The instance $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha)$ is the first instance and any sub-instance will not be removed from memory till the position $\alpha + b$. There are b+1 sub-instances of $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha)$ at $\alpha + b - 1$.
- We also need to count the instances $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha + 1), \cdots,$ $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha + b - 1)$ of which there are

$$\sum_{i=2}^{b} i = \frac{b \cdot (b+1)}{2} - 1.$$

- At $\alpha + b$ we remove $b + 1$ instances from memory, unroll $b - 1$ new ones, and add two instances for $\langle 0, b, b\rangle_{\mathbb{N}}(\alpha + b)$. Thus the used portion of memory stays the same.
- It can be shown inductively that the same pattern holds for every position $\alpha + b \leq$, thus, $x = \alpha + b - 1$.

Base Case

– The following two corollaries follow from the result:

Corollary

Given an $\mathbb{N}$-triple t of the form $\langle 0, b, b\rangle_{\mathbb{N}}$ and an interval $[\alpha, \infty]$, then there exists a value $x \in [\alpha, \infty]$ such that given the proper evaluation chain:

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\infty]} \cdots [x-1, t, \mathbf{I}_0] \xrightarrow{[\alpha,\infty]} [x, t, \mathbf{I}_1] \xrightarrow{[\alpha,\infty]} \cdots$$

the following holds $|\mathbf{I}_0| \neq |\mathbf{I}_1| = |\mathbf{I}_2| = \cdots$.

Corollary

Given an $\mathbb{N}$-triple t of the form $\langle 0, b, b\rangle_{\mathbb{N}}$, an interval $[\alpha, \infty]$, and the proper evaluation chain:

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\infty]} [\alpha, t, \mathbf{I}_\alpha] \xrightarrow{[\alpha,\infty]} [\alpha+1, t, \mathbf{I}_{\alpha+1}] \xrightarrow{[\alpha,\infty]} \cdots$$

then,

$$|\mathbf{I}_n| \leq \frac{(b+1)*(b+2)}{2} - 1$$

for all $n \in [\alpha, \infty]$.

All $\mathbb{N}$-Triples

– The space complexity of any $\mathbb{N}$-triple $\langle a, b, c\rangle_{\mathbb{N}}$ is as follows:

Theorem

Given an $\mathbb{N}$-triple t of the form $\langle a, b, c\rangle_{\mathbb{N}}$,where $0 \leq a \leq b$, $0 \leq c$, an interval $[\alpha, \infty]$, and the proper evaluation chain:

$$[\triangleright, t, \emptyset] \xrightarrow{[\alpha,\infty]} [\alpha, t, \mathbf{I}_\alpha] \xrightarrow{[\alpha,\infty]} [\alpha + 1, t, \mathbf{I}_{\alpha+1}] \xrightarrow{[\alpha,\infty]} \cdots$$

then,

$$|\mathbf{I}_n| \leq BT(a, b, c)$$

for all $n \in [\alpha, \infty]$,where

$$BT(a,b,c) = \begin{cases} a + (l-1) & c \leq a \\ a + \dfrac{(l-d)(l-d+1)}{2} + (d-1) & c = b - d \;\&\; a \leq c < b \\ a + \dfrac{l*(l+1)}{2} + d*l - 1 & c = b + d \end{cases}$$

We define $l = (b - a) + 1$.

Bounding Function

- Now we move from bounding $\mathbb{N}$-triples to bounding the reduced core language, i.e. without infinity, constants, and subtraction.
- Formulas of the reduced core language can still have variable nesting.
- Before introducing our transformation removing variable nesting we add one more assumption.
- We assume that each quantifier in a given formula has a unique variable name.

Dominating Formula

Definition (Dominating Formula Transformation)

Given a sentence $f \in \mathbb{M}^{vb}$ we construct the dominating formula f_D of f using the following transformation

$$
\begin{aligned}
D(\forall_{0 \le x} : f_D, \emptyset, \emptyset) &\Longrightarrow D(\forall_{0 \le x} : D(f, \{x \leftarrow x\}, \{x \leftarrow x\})) \\
D(f_1 \ \& \ f_2, \sigma_l, \sigma_h) &\Longrightarrow D(f_1, \sigma_l, \sigma_h) \ \& \ D(f_2, \sigma_l, \sigma_h) \\
D(\neg f, \sigma_l, \sigma_h) &\Longrightarrow \neg D(f, \sigma_l, \sigma_h) \\
D(\forall_{y \in [b_1, b_2]} : f, \sigma_l, \sigma_h) &\Longrightarrow \forall_{y \in [h_L(b_1), h_H(b_2)]} : D(f, \sigma_l \{y \leftarrow h_L(b_1)\}, \sigma_h \{y \leftarrow h_H(b_2)\}) \\
D(@x, \sigma_l, \sigma_h) &\Longrightarrow @x
\end{aligned}
$$

where $h_L(b_1) = \min \{b_1\sigma_l, b_1\sigma_h\}$ and $h_H(b_2) = \max \{b_2\sigma_l, b_2\sigma_h\}$.

Example Dominating Formula

Definition (Quantifier Tree of a Monitor (Formula))

Given $m \in M$, and $f \in F$, we define the quantifier tree $QT(m)$ of m, respectively $QT(f)$ of f, recursively as follows:

$$
\begin{aligned}
QT(\forall_{0 \leq V} : F) &= (V, 0, 0, QT(F)) \\
QT(F \& G) &= QT(F) \cup QT(G) \\
QT(F \wedge G) &= QT(F) \cup QT(G) \\
QT(\neg F) &= \ QT(F) \\
QT(\forall_{V \in [B_1, B_2]} : F) &= (V, B_1, B_2, QT(F)) \\
QT(@V) &= \emptyset
\end{aligned}
$$

The Bounding Function

— Now we introduce the bounding function:

Definition (Bounding Function)

Given a sentence $f \in \mathbb{M}^{vb}$, let f_D be its dominating formula. We construct the bounding function $b(f_D)$ as follows:

$$
\begin{array}{ccc}
b(\forall_{0 \le x} : f) & \Longrightarrow & b(f, \{x \leftarrow 0\}) \\
b(@f \ \& \ @g, \sigma) & \Longrightarrow & b(f, \sigma) + b(g, \sigma) \\
b(\neg f, \sigma) & \Longrightarrow & b(f, \sigma) \\
b(\forall_{y \in [x+a, x+b]} : f, \sigma) & \Longrightarrow & \color{red}{BT(a, b, w(f, \sigma\,\{y \leftarrow (x+b)\sigma\}))} * b(f, \sigma\,\{y \leftarrow (x+b)\sigma\}) \\
b(@y, \sigma) & \Longrightarrow & 1 \\
w(@f \ \& \ @g, \sigma) & \Longrightarrow & \max\{w(f, \sigma), w(G, \sigma)\} \\
w(\neg f, \sigma) & \Longrightarrow & w(f, \sigma) \\
w(\forall_{y \in [x+a, x+b]} : f, \sigma) & \Longrightarrow & w(f, \sigma\,\{y \leftarrow (x+b)\sigma\}) \\
w(@y, \sigma) & \Longrightarrow & y\sigma
\end{array}
$$

Bounding Function Example

- We can compute the upper bound of the previous example using this bounding function:

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

Bounding Function Example

— We can compute the upper bound of the previous example using this bounding function:

$$\forall_{0 \le x} : \forall_{y \in [x+1, x+5]} :$$
$$\left(\forall_{z \in [x+1, x+3]} : \neg @z \;\&\; @y\right) \;\&\; \left(\forall_{w \in [x, x+7]} : \neg @y \;\&\; \forall_{m \in [x+1, x+7]} : \neg @z \;\&\; @m\right)$$

$$BT(1,5,7) * (BT(1,3,5) + BT(-1,7,7) * BT(1,7,7)) =$$
$$25 * (12 + 34 * 28) = 24100$$

Bounding Function Example

- We can compute the upper bound of the previous example using this bounding function:

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

$$BT(1,5,7) * (BT(1,3,5) + BT(-1,7,7) * BT(1,7,7)) =$$
$$25 * (12 + 34 * 28) = 24100$$

- This is the resulting bound for the naive method
- With a few simple optimization we are able to get a more accurate result, ≈ 1000.
- Though the true value for the dominating formula is 240 and non-dominating formula 18

Conclusions and Future Work

- We have developed the method completely for the entire core language.
- Though, as one can see the method is quite coarse.

Conclusions and Future Work

- We have developed the method completely for the entire core language.
- Though, as one can see the method is quite coarse.
- Our future work will focus on extending the method to more general structures. For example, quantifier chains.

Conclusions and Future Work

- We have developed the method completely for the entire core language.
- Though, as one can see the method is quite coarse.
- Our future work will focus on extending the method to more general structures. For example, quantifier chains.
- Also, we would like to investigate the dominating formula transformation.
- For example, how coarse is the transformation.

Conclusions and Future Work

- We have developed the method completely for the entire core language.
- Though, as one can see the method is quite coarse.
- Our future work will focus on extending the method to more general structures. For example, quantifier chains.
- Also, we would like to investigate the dominating formula transformation.
- For example, how coarse is the transformation.
- Still open is finding a precise bound for the entire core language.

Thank you for your time.