Comparing XML Path Expressions

Pierre Genevès and Nabil Layaïda

INRIA Rhône-Alpes
pierre.geneves@inria.fr

ACM Doc'Eng, October 2006
Amsterdam, Netherlands

Introduction

Motivation

- XML transformations: the basic processing of structured documents (rendering, repurposing, content adaptation...)
- XML processing code should be safe and efficient
- Document transformations are based on W3C standard languages (XSLT, XQuery,...) which use XPath

Ultimate goal

- **Static** type-checking of XML processing code involving XPath

Approach

- Static analysis of XML transformation languages
- Find effective methods for analyzing XPath queries

Introduction

Motivation

- XML transformations: the basic processing of structured documents (rendering, repurposing, content adaptation...)
- XML processing code should be safe and efficient
- Document transformations are based on W3C standard languages (XSLT, XQuery,...) which use XPath

Ultimate goal

- **Static** type-checking of XML processing code involving XPath

Approach

- Static analysis of XML transformation languages
- Find effective methods for analyzing XPath queries

Introduction

Motivation

- XML transformations: the basic processing of structured documents (rendering, repurposing, content adaptation...)
- XML processing code should be safe and efficient
- Document transformations are based on W3C standard languages (XSLT, XQuery,...) which use XPath

Ultimate goal

- **Static** type-checking of XML processing code involving XPath

Approach

- Static analysis of XML transformation languages
- Find effective methods for analyzing XPath queries

Basic XPath Static Analysis Problems

Query Emptiness

- Does a query always return an empty result when evaluated on a set of XML documents?
- Applications: error-detection and optimization of host languages implementations

Query Containment

- Is the result of q_1 always included in the result of q_2 when evaluated on a set of XML documents?
- Applications: type-checking; control-flow analysis of XSLT; checking integrity constraints; checking access control in XML security

Basic XPath Static Analysis Problems

Query Emptiness

- Does a query always return an empty result when evaluated on a set of XML documents?
- Applications: error-detection and optimization of host languages implementations

Query Containment

- Is the result of q_1 always included in the result of q_2 when evaluated on a set of XML documents?
- Applications: type-checking; control-flow analysis of XSLT; checking integrity constraints; checking access control in XML security

Other XPath Static Analysis Problems

Query Equivalence

- Do q_1 and q_2 always return the same result when evaluated on a set of XML documents?
- Applications: reformulation and optimization of queries

Query Overlap

- Do q_1 and q_2 select common nodes when evaluated on a set of XML documents?
- Application: error-detection and code verification

Query Coverage

- Is the result of q_1 always contained in the union of the results of $q_2, q_3, ..., q_n$?
- Application: error-detection and code verification

Other XPath Static Analysis Problems

Query Equivalence

- Do q_1 and q_2 always return the same result when evaluated on a set of XML documents?
- Applications: reformulation and optimization of queries

Query Overlap

- Do q_1 and q_2 select common nodes when evaluated on a set of XML documents?
- Application: error-detection and code verification

Query Coverage

- Is the result of q_1 always contained in the union of the results of $q_2, q_3, ..., q_n$?
- Application: error-detection and code verification

Other XPath Static Analysis Problems

Query Equivalence

- Do q_1 and q_2 always return the same result when evaluated on a set of XML documents?
- Applications: reformulation and optimization of queries

Query Overlap

- Do q_1 and q_2 select common nodes when evaluated on a set of XML documents?
- Application: error-detection and code verification

Query Coverage

- Is the result of q_1 always contained in the union of the results of $q_2, q_3, ..., q_n$?
- Application: error-detection and code verification

Research Challenges

Fact

- Static analysis of the complete XPath language is undecidable

Open Questions

- What is the largest XPath fragment with decidable static analysis?
- Which fragments can be effectively decided in practice?
- Are there algorithms able to solve XPath decision problems in an efficient way so that they can be used in practice?

Difficulties

- Considered XPath operators and their combination
- Properties on a possibly infinite quantification over a set of XML documents

Research Challenges

Fact

- Static analysis of the complete XPath language is undecidable

Open Questions

- What is the largest XPath fragment with decidable static analysis?
- Which fragments can be effectively decided in practice?
- Are there algorithms able to solve XPath decision problems in an efficient way so that they can be used in practice?

Difficulties

- Considered XPath operators and their combination
- Properties on a possibly infinite quantification over a set of XML documents

Research Challenges

Fact

- Static analysis of the complete XPath language is undecidable

Open Questions

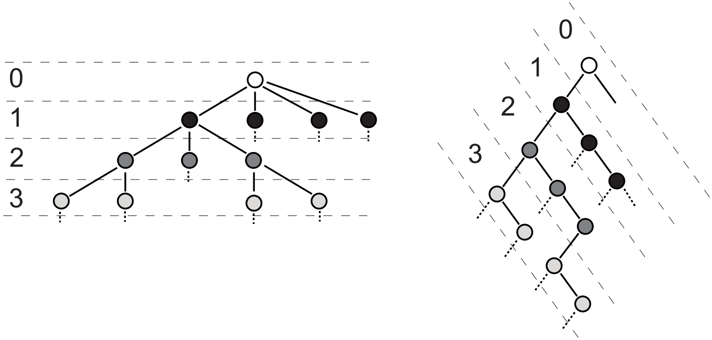
- What is the largest XPath fragment with decidable static analysis?
- Which fragments can be effectively decided in practice?
- Are there algorithms able to solve XPath decision problems in an efficient way so that they can be used in practice?

Difficulties

- Considered XPath operators and their combination
- Properties on a possibly infinite quantification over a set of XML documents

XML Documents

- Finite ordered trees of variable arity, labeled with a unique symbol per node
- Straightforward isomorphism between unranked and binary trees:



- XML documents seen as finite binary trees
- Navigation can be expressed in binary style

The Logical Approach

Method

- Find an appropriate logic for reasoning on finite binary trees
- Embed XPath queries in the logic: $q \longrightarrow \varphi_q$
- Formulate the decision problem to solve
  - Example: containment test between q_1 and q_2
  - $\forall t, \forall x \in t, [\![q_1]\!]_x^t \subseteq [\![q_2]\!]_x^t$
  - Validity of $\varphi_{q_1} \implies \varphi_{q_2}$
  - Unsatisfiability of $\varphi_{q_1} \wedge \neg\varphi_{q_2}$
- Use the decision procedure of the logic: yes/no answer
  - satisfiable/unsatisfiable
  - if satisfiable, gives a satisfying XML document (counterexample for the containment)

Critical Aspects

- The logic must be expressive enough
- The decision procedure must be effective in practice for XPath formulas

The Logical Approach

Method

- Find an appropriate logic for reasoning on finite binary trees
- Embed XPath queries in the logic: $q \longrightarrow \varphi_q$
- Formulate the decision problem to solve
  - Example: containment test between q_1 and q_2
  - $\forall t, \forall x \in t, [\![q_1]\!]_x^t \subseteq [\![q_2]\!]_x^t$
  - Validity of $\varphi_{q_1} \implies \varphi_{q_2}$
  - Unsatisfiability of $\varphi_{q_1} \wedge \neg\varphi_{q_2}$
- Use the decision procedure of the logic: yes/no answer
  - satisfiable/unsatisfiable
  - if satisfiable, gives a satisfying XML document (counterexample for the containment)

Critical Aspects

- The logic must be expressive enough
- The decision procedure must be effective in practice for XPath formulas

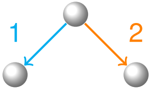
A Candidate Logic

AFMC (Alternation-Free Fragment of the Modal μ-Calculus)

- [Kozen, 1983, Vardi, 1998, Tanabe et al., 2005]
- Expressiveness:
  - AFMC $\equiv$ MSO $\equiv$ Finite Tree Automata (e.g. also captures Relax NG, DTD, XML Schema)
  - Can be extended with converse programs: useful to capture XPath semantics of selection
- Models: Kripke structures (labeled graphs)
- Complexity for decidability: **exponential time** (even when extended with converse)

Syntax of the Logic

- Programs for navigating ordered binary trees: $\alpha \in \{1, 2, \overline{1}, \overline{2}\}$ with $\overline{\overline{\alpha}} = \alpha$:

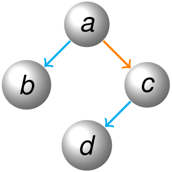


- Formulas:

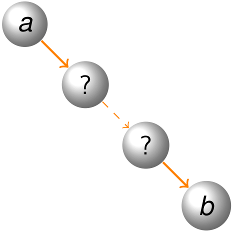
φ, ψ	$::=$		formula
		$\top \mid \bot$	true (false)
	$\mid$	$p \mid \neg p$	atomic proposition (negated)
	$\mid$	$\varphi \vee \psi$	disjunction
	$\mid$	$\varphi \wedge \psi$	conjunction
	$\mid$	$\langle\alpha\rangle\, \varphi$	existential modality
	$\mid$	$[\alpha]\, \varphi$	universal modality
	$\mid$	X	variable
	$\mid$	$\mu X.\varphi$	least fixpoint
	$\mid$	$\nu X.\varphi$	greatest fixpoint

Semantics of the Logic: Examples

- The interpretation of a formula is the set of its satisfying Kripke structures (labeled graphs), for example:
- $a \wedge \langle 1 \rangle \, b \wedge \langle 2 \rangle \langle 1 \rangle \, d \wedge [2] \, c$

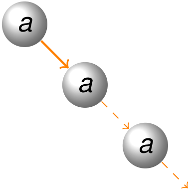


- $a \wedge \mu X.b \vee \langle 2 \rangle \, X$
  shorthand for $a \wedge (b \vee \langle 2 \rangle \, b \vee \langle 2 \rangle \langle 2 \rangle \, b \vee \langle 2 \rangle \langle 2 \rangle \langle 2 \rangle \, b \vee ...)$



Kripke Structures vs. XML Documents

- Due to converse programs, the AFMC does not have the finite tree model property:
  - models are Kripke structures (graphs) that may contain cyclic or infinite paths
  - there exist formulas which are satisfiable on Kripke structures but not on XML documents (finite binary trees)
- Example: $\nu X.a \wedge \langle 2 \rangle X$

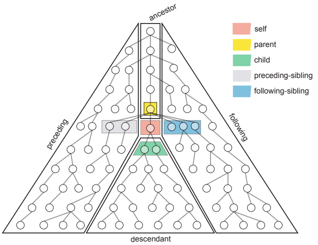


- For XML, we must restrict the models to be finite trees
- This can be done by syntactically rewriting the formula (see the paper)
- Satisfiability over graphs is restricted to satisfiability over finite binary trees

Considered XPath Fragment

Syntax

Expression	e	$::=$	$/p$ | p | $e_1 \cup e_2$ | $e_1 \cap e_2$
Path	p	$::=$	p_1/p_2 | $p[q]$ | a::n
Qualifier	q	$::=$	q and q | q or q | not q | p
Axis	a	$::=$	child | descendant | self | parent | ancestor | following | preceding | descendant-or-self | ancestor-or-self | preceding-sibling | following-sibling
NodeTest	n	$::=$	σ | $*$



Peculiarities

- Multi-directional tree navigation
- Node selection and path existence:
  - a/b: all "b" children of "a" nodes
  - $a[b]$: all "a" nodes which have at least one child "b"
- Almost full XPath (only counting and data values left)

Translating XPath into AFMC

- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::a[child::b]

- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::*a*[child::*b*]

$$\psi = a \wedge (\mu Y. \langle \bar{1} \rangle \chi \vee \langle \bar{2} \rangle Y) \\ \wedge \langle 1 \rangle \mu Z. b \vee \langle 2 \rangle Z$$

- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::*a*[child::*b*]

$$\psi = a \wedge (\mu Y. \langle \bar{1} \rangle \chi \vee \langle \bar{2} \rangle Y) \wedge \langle 1 \rangle \mu Z. b \vee \langle 2 \rangle Z$$

χ

- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::*a*[child::*b*]

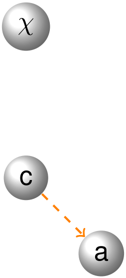
$$\psi = a \wedge (\mu Y. \langle \bar{1} \rangle \chi \vee \langle \bar{2} \rangle Y)$$
$$\wedge \langle 1 \rangle \mu Z.b \vee \langle 2 \rangle Z$$

- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::a[child::b]

$$\psi = a \wedge (\mu Y. \langle \bar{1} \rangle \chi \vee \langle \bar{2} \rangle Y) \wedge \langle 1 \rangle \mu Z.b \vee \langle 2 \rangle Z$$

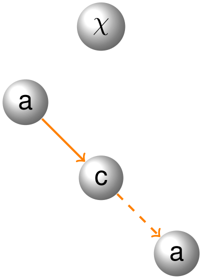


- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::*a*[child::*b*]

$$\psi = a \wedge (\mu Y. \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Y) \wedge \langle 1 \rangle \mu Z.b \vee \langle 2 \rangle Z$$

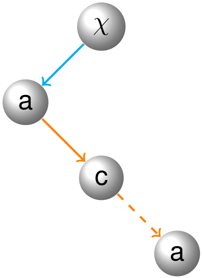


- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::*a*[child::*b*]

$$\psi = a \wedge (\mu Y. \langle \bar{1} \rangle \chi \vee \langle \bar{2} \rangle Y) \wedge \langle 1 \rangle \mu Z.b \vee \langle 2 \rangle Z$$

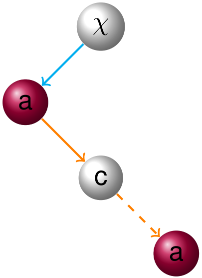


- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::a[child::b]

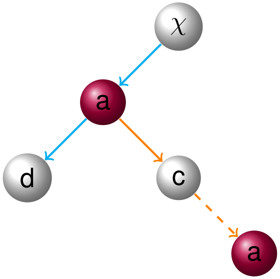
$$\psi = a \wedge (\mu Y. \langle \bar{1} \rangle \, \chi \vee \langle \bar{2} \rangle \, Y) \\ \wedge \langle 1 \rangle \, \mu Z.b \vee \langle 2 \rangle \, Z$$



- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Example: child::*a*[child::*b*]

$$\psi = a \wedge (\mu Y.\, \langle \bar{1} \rangle\, \chi \vee \langle \bar{2} \rangle\, Y) \wedge \langle 1 \rangle\, \mu Z.b \vee \langle 2 \rangle\, Z$$

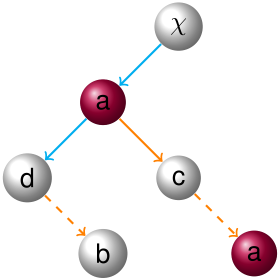


- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::a[child::b]

$$\psi = a \wedge (\mu Y. \langle \bar{1} \rangle \chi \vee \langle \bar{2} \rangle Y) \wedge \langle 1 \rangle \mu Z.b \vee \langle 2 \rangle Z$$

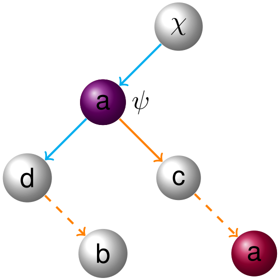


- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Translating XPath into AFMC

Example: child::a[child::b]

$$\psi = a \wedge (\mu Y. \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Y) \wedge \langle 1 \rangle \mu Z. b \vee \langle 2 \rangle Z$$

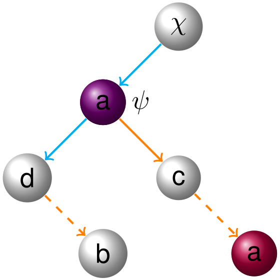


- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_\cdot$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_\cdot$: for stating path existence

Translating XPath into AFMC

Example: child::a[child::b]

$$\psi = a \wedge (\mu Y. \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Y)$$
$$\wedge \langle 1 \rangle \mu Z. b \vee \langle 2 \rangle Z$$



- The complete XPath fragment can be linearly translated into AFMC
- See the paper for formal translations
- Two classes of translating functions:
  - $P^{\rightarrow}[\![\cdot]\!]_{\cdot}$: for selecting nodes through navigation
  - $P^{\leftarrow}[\![\cdot]\!]_{\cdot}$: for stating path existence

Summary

Approach for solving XPath decision problems

- XPath queries are linearly translated into AFMC
- The decision problem (e.g. containment) is formulated in AFMC and transformed
- A Tableau-based method specialized for deciding AFMC [Tanabe et al., 2005] is used
- Time complexity: $2^{O(n \cdot log(n))}$
- Gives acceptable results in practice
- (Demo?)

Conclusion

Results

- The largest XPath fragment treated so far for static analysis
- A new complexity upper bound for XPath decision problems: $2^{O(n \cdot log(n))}$ (the smallest and most precise)
- The approach appears efficient in practice

More Results

- The approach even scales to support DTDs, XSDs: see our forthcoming TOIS'06 article [Genevès and Layaïda, 2006] (longer version)
- A restricted calculus for finite trees yields a still better $2^{O(n)}$ complexity: see http://wam.inrialpes.fr/software/xml-calculus/

Perspectives

- Application to the static type-checking of XSLT, XQuery

Conclusion

Results

- The largest XPath fragment treated so far for static analysis
- A new complexity upper bound for XPath decision problems: $2^{O(n \cdot log(n))}$ (the smallest and most precise)
- The approach appears efficient in practice

More Results

- The approach even scales to support DTDs, XSDs: see our forthcoming TOIS'06 article [Genevès and Layaïda, 2006] (longer version)
- A restricted calculus for finite trees yields a still better $2^{O(n)}$ complexity: see http://wam.inrialpes.fr/software/xml-calculus/

Perspectives

- Application to the static type-checking of XSLT, XQuery

Conclusion

Results

- The largest XPath fragment treated so far for static analysis
- A new complexity upper bound for XPath decision problems: $2^{O(n \cdot log(n))}$ (the smallest and most precise)
- The approach appears efficient in practice

More Results

- The approach even scales to support DTDs, XSDs: see our forthcoming TOIS'06 article [Genevès and Layaïda, 2006] (longer version)
- A restricted calculus for finite trees yields a still better $2^{O(n)}$ complexity: see http://wam.inrialpes.fr/software/xml-calculus/

Perspectives

- Application to the static type-checking of XSLT, XQuery

Genevès, P. and Layaïda, N. (2006).
A system for the static analysis of XPath.
ACM Transactions on Information Systems. To appear.

Kozen, D. (1983).
Results on the propositional μ-calculus.
Theoretical Computer Science, 27:333–354.

Tanabe, Y., Takahashi, K., Yamamoto, M., Tozawa, A., and Hagiya, M. (2005).
A decision procedure for the alternation-free two-way modal μ-calculus.
In *TABLEAUX '05: Proceedings of the 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 3702 of *LNCS*, pages 277–291, London, UK. Springer-Verlag.

Vardi, M. Y. (1998).
Reasoning about the past with two-way automata.
In *ICALP '98: Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 628–641, London, UK. Springer-Verlag.

Semantics of $\mathcal{L}_\mu^{\text{full}}$

A Kripke Structure $K = \langle W, R, L \rangle$ is a Labeled Graph

- W: set of nodes
- nodes are labeled with atomic propositions ($L : Prop \to 2^W$)
- edges are labeled with programs ($R : Prog \to 2^{W \times W}$)

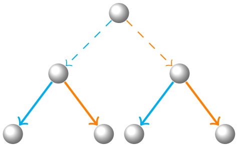
$\mathcal{L}_\mu^{\text{full}}$ Semantics ($V : Var \to 2^W$ is a Valuation for Variables)

$$
\begin{array}{lcl}
[\![\cdot]\!]_V^K & : & \mathcal{L}_\mu^{\text{full}} \longrightarrow 2^W \\
[\![\top]\!]_V^K & = & W \\
[\![\bot]\!]_V^K & = & \emptyset \\
[\![p]\!]_V^K & = & L(p) \\
[\![X]\!]_V^K & = & V(X) \\
[\![\neg\varphi]\!]_V^K & = & W \setminus [\![\varphi]\!]_V^K
\end{array}
$$

$$
\begin{array}{lcl}
[\![\varphi_1 \vee \varphi_2]\!]_V^K & = & [\![\varphi_1]\!]_V^K \cup [\![\varphi_2]\!]_V^K \\
[\![\varphi_1 \wedge \varphi_2]\!]_V^K & = & [\![\varphi_1]\!]_V^K \cap [\![\varphi_2]\!]_V^K \\
[\![[\alpha]\,\varphi]\!]_V^K & = & \{w : \forall w'(w, w') \in R(\alpha) \Rightarrow w' \in [\![\varphi]\!]_V^K\} \\
[\![\langle\alpha\rangle\,\varphi]\!]_V^K & = & \{w : \exists w'(w, w') \in R(\alpha) \wedge w' \in [\![\varphi]\!]_V^K\} \\
[\![\mu X.\varphi]\!]_V^K & = & \bigcap\{W' \subseteq W : [\![\varphi]\!]_{V[X/W']}^K \subseteq W'\} \\
[\![\nu X.\varphi]\!]_V^K & = & \bigcup\{W' \subseteq W : [\![\varphi]\!]_{V[X/W']}^K \supseteq W'\}
\end{array}
$$

Enforcing the Finite Tree Model Property

- The property can be enforced at the AFMC formula level:
  - Trees have at most one node per program: $\langle\alpha\rangle\,\varphi \rightsquigarrow \langle\alpha\rangle\,\top \wedge [\alpha]\,\varphi$
  - Koenig's lemma is used for ensuring finiteness
    - The formula $\mu X.\,[1]\,X \wedge [2]\,X$
    - is vacuously satisfied at the leaves
    - and only at the top of finite branches



- The checked formula becomes $\varphi^{\text{rewritten}} \wedge \mu X.\,[1]\,X \wedge [2]\,X$
- Satisfiability over Kripke structures is reduced to satisfiability over finite binary trees