

XPath, transitive closure logic, and nested tree walking automata

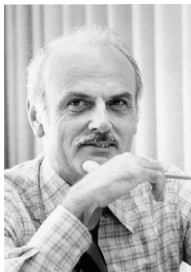
Balder ten Cate

U. of Amsterdam and UC Santa Cruz (visiting IBM Almaden)

Luc Segoufin

INRIA, ENS Cachan

Logic and Algorithms, Edinburgh 2008



987

RELATIONAL COMPLETENESS OF DATA BASE SUBLANGUAGES

by

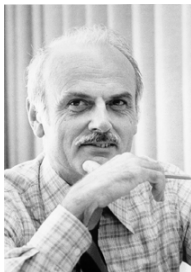
E. F. Codd

IBM Research Laboratory
San Jose, California

Codd (1972)

- proposes **first-order logic (FO)** as yardstick of expressive power.
- shows that his **relational algebra** is expressively complete for FO: it can express every FO-definable query.

Our aim: a theory of Codd completeness for XPath.



987

RELATIONAL COMPLETENESS OF DATA BASE SUBLANGUAGES

by

E. F. Codd

IBM Research Laboratory
San Jose, California

Codd (1972)

- proposes **first-order logic (FO)** as yardstick of expressive power.
- shows that his **relational algebra** is expressively complete for FO: it can express every FO-definable query.

Our aim: a theory of Codd completeness for XPath.

- XPath expressions **navigate through XML documents** from one node to another. In other words, they define **binary relations**.
- The **four basic moves** in the tree:
 - \uparrow (go to the **parent** of the current node)
 - \downarrow (go to a **child** of the current node)
 - \leftarrow (go to the **previous sibling** of the current node)
 - \rightarrow (go to the **next sibling** of the current node)
- These can be combined with **node tests** and various operations such as **composition** ($/$) and **union** (\cup).
(which operations are allowed differs per XPath dialect.)
- **Example:** \uparrow/\downarrow denotes the binary relation $\{(n_{start}, n_{end}) \mid \text{either } n_{end} \text{ is a sibling of } n_{start}, \text{ or } n_{end} = n_{start} \neq \text{root}\}$

- XPath expressions **navigate through XML documents** from one node to another. In other words, they define **binary relations**.
- The **four basic moves** in the tree:
 - \uparrow (go to the **parent** of the current node)
 - \downarrow (go to a **child** of the current node)
 - \leftarrow (go to the **previous sibling** of the current node)
 - \rightarrow (go to the **next sibling** of the current node)
- These can be combined with **node tests** and various operations such as **composition** ($/$) and **union** (\cup).
(which operations are allowed differs per XPath dialect.)
- **Example:** \uparrow/\downarrow denotes the binary relation $\{(n_{start}, n_{end}) \mid \text{either } n_{end} \text{ is a sibling of } n_{start}, \text{ or } n_{end} = n_{start} \neq \text{root}\}$

- XPath expressions **navigate through XML documents** from one node to another. In other words, they define **binary relations**.
- The **four basic moves** in the tree:
 - \uparrow (go to the **parent** of the current node)
 - \downarrow (go to a **child** of the current node)
 - \leftarrow (go to the **previous sibling** of the current node)
 - \rightarrow (go to the **next sibling** of the current node)
- These can be combined with **node tests** and various operations such as **composition** ($/$) and **union** (\cup).
(which operations are allowed differs per XPath dialect.)
- **Example:** \uparrow/\downarrow denotes the binary relation $\{(n_{start}, n_{end}) \mid \text{either } n_{end} \text{ is a sibling of } n_{start}, \text{ or } n_{end} = n_{start} \neq \text{root}\}$

- XPath expressions **navigate through XML documents** from one node to another. In other words, they define **binary relations**.
- The **four basic moves** in the tree:
 - \uparrow (go to the **parent** of the current node)
 - \downarrow (go to a **child** of the current node)
 - \leftarrow (go to the **previous sibling** of the current node)
 - \rightarrow (go to the **next sibling** of the current node)
- These can be combined with **node tests** and various operations such as **composition** ($/$) and **union** (\cup).
(which operations are allowed differs per XPath dialect.)
- **Example:** \uparrow/\downarrow denotes the binary relation $\{(n_{start}, n_{end}) \mid \text{either } n_{end} \text{ is a sibling of } n_{start}, \text{ or } n_{end} = n_{start} \neq \text{root}\}$

- XPath is used for selecting nodes in an XML document tree.
- XPath lies at the core of the XML querying and processing languages XQuery and XSLT.
- Here, we are interested in the **tree navigation power** of XPath. We ignore arithmetical- and string operations etc.
⇒ we study the **“navigational (logical) core”** of XPath

- XPath is used for selecting nodes in an XML document tree.
- XPath lies at the core of the XML querying and processing languages XQuery and XSLT.
- Here, we are interested in the **tree navigation power** of XPath. We ignore arithmetical- and string operations etc.
⇒ we study the **“navigational (logical) core”** of XPath

- XPath is used for selecting nodes in an XML document tree.
- XPath lies at the core of the XML querying and processing languages XQuery and XSLT.
- Here, we are interested in the **tree navigation power** of XPath. We ignore arithmetical- and string operations etc.
⇒ we study the “**navigational (logical) core**” of XPath

- XML documents are finite sibling-ordered trees whose nodes are labeled with atomic information (**tag**, **attribute-value pairs**, **string content**).
- We model atomic information by a set Σ of **node labels**.
- So, an XML document is a structure $T = (dom_T, R_{\downarrow}, R_{\rightarrow}, L)$ where
 - dom_T is the set of nodes,
 - R_{\downarrow} and R_{\rightarrow} are the 'child' and 'next sibling' relations, and
 - $L : N \rightarrow \wp(\Sigma)$.
- One may assume that L assigns a single label to each node, our results are independent of such an assumption.

- XML documents are finite sibling-ordered trees whose nodes are labeled with atomic information (**tag**, **attribute-value pairs**, **string content**).
- We model atomic information by a set Σ of **node labels**.
- So, an XML document is a structure $T = (dom_T, R_{\downarrow}, R_{\rightarrow}, L)$ where
 - dom_T is the set of nodes,
 - R_{\downarrow} and R_{\rightarrow} are the 'child' and 'next sibling' relations, and
 - $L : N \rightarrow \wp(\Sigma)$.
- One may assume that L assigns a single label to each node, our results are independent of such an assumption.

- XML documents are finite sibling-ordered trees whose nodes are labeled with atomic information (**tag**, **attribute-value pairs**, **string content**).
- We model atomic information by a set Σ of **node labels**.
- So, an XML document is a structure $T = (dom_T, R_{\downarrow}, R_{\rightarrow}, L)$ where
 - dom_T is the set of nodes,
 - R_{\downarrow} and R_{\rightarrow} are the 'child' and 'next sibling' relations, and
 - $L : N \rightarrow \wp(\Sigma)$.
- One may assume that L assigns a single label to each node, our results are independent of such an assumption.

- XML documents are finite sibling-ordered trees whose nodes are labeled with atomic information (**tag**, **attribute-value pairs**, **string content**).
- We model atomic information by a set Σ of **node labels**.
- So, an XML document is a structure $T = (dom_T, R_{\downarrow}, R_{\rightarrow}, L)$ where
 - dom_T is the set of nodes,
 - R_{\downarrow} and R_{\rightarrow} are the 'child' and 'next sibling' relations, and
 - $L : N \rightarrow \wp(\Sigma)$.
- One may assume that L assigns a single label to each node, our results are independent of such an assumption.

- XML documents are finite sibling-ordered trees whose nodes are labeled with atomic information (**tag**, **attribute-value pairs**, **string content**).
- We model atomic information by a set Σ of **node labels**.
- So, an XML document is a structure $T = (dom_T, R_{\downarrow}, R_{\rightarrow}, L)$ where
 - dom_T is the set of nodes,
 - R_{\downarrow} and R_{\rightarrow} are the 'child' and 'next sibling' relations, and
 - $L : N \rightarrow \wp(\Sigma)$.
- One may assume that L assigns a single label to each node, our results are independent of such an assumption.

- XML documents are finite sibling-ordered trees whose nodes are labeled with atomic information (**tag**, **attribute-value pairs**, **string content**).
- We model atomic information by a set Σ of **node labels**.
- So, an XML document is a structure $T = (dom_T, R_{\downarrow}, R_{\rightarrow}, L)$ where
 - dom_T is the set of nodes,
 - R_{\downarrow} and R_{\rightarrow} are the 'child' and 'next sibling' relations, and
 - $L : N \rightarrow \wp(\Sigma)$.
- One may assume that L assigns a single label to each node, our results are independent of such an assumption.

- XML documents are finite sibling-ordered trees whose nodes are labeled with atomic information (**tag**, **attribute-value pairs**, **string content**).
- We model atomic information by a set Σ of **node labels**.
- So, an XML document is a structure $T = (dom_T, R_{\downarrow}, R_{\rightarrow}, L)$ where
 - dom_T is the set of nodes,
 - R_{\downarrow} and R_{\rightarrow} are the 'child' and 'next sibling' relations, and
 - $L : N \rightarrow \wp(\Sigma)$.
- One may assume that L assigns a single label to each node, our results are independent of such an assumption.

- Core XPath 1.0 (the navigational core of XPath 1.0) has two types of expressions:

- path expressions

$$\alpha ::= d \mid d^+ \mid . \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\phi] \quad (d \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\})$$

- node expressions

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle \alpha \rangle \quad (p \in \Sigma)$$

- Path expressions define binary relations. When applied to a given “context node”, they yield a set of nodes.

Node expressions define sets of nodes.

- We use $/\alpha$ as shorthand for $\uparrow^* [\neg\langle \uparrow \rangle]/\alpha$.
We use d^* as a shorthand for $(d^+ \cup .)$.

- Core XPath 1.0 (the navigational core of XPath 1.0) has two types of expressions:

- **path expressions**

$$\alpha ::= d \mid d^+ \mid . \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\phi] \quad (d \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\})$$

- **node expressions**

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle \alpha \rangle \quad (p \in \Sigma)$$

- Path expressions define binary relations. When applied to a given “context node”, they yield a set of nodes.

Node expressions define sets of nodes.

- We use $/\alpha$ as shorthand for $\uparrow^* [\neg\langle \uparrow \rangle]/\alpha$.
We use d^* as a shorthand for $(d^+ \cup .)$.

- Core XPath 1.0 (the navigational core of XPath 1.0) has two types of expressions:

- **path expressions**

$$\alpha ::= d \mid d^+ \mid . \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\phi] \quad (d \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\})$$

- **node expressions**

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle \alpha \rangle \quad (p \in \Sigma)$$

- Path expressions define binary relations. When applied to a given “context node”, they yield a set of nodes.

Node expressions define sets of nodes.

- We use $/\alpha$ as shorthand for $\uparrow^* [\neg\langle \uparrow \rangle]/\alpha$.
We use d^* as a shorthand for $(d^+ \cup .)$.

- Core XPath 1.0 (the navigational core of XPath 1.0) has two types of expressions:

- **path expressions**

$$\alpha ::= d \mid d^+ \mid . \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\phi] \quad (d \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\})$$

- **node expressions**

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle \alpha \rangle \quad (p \in \Sigma)$$

- Path expressions define binary relations. When applied to a given “**context node**”, they yield a set of nodes.

Node expressions define sets of nodes.

- We use $/\alpha$ as shorthand for $\uparrow^* [\neg\langle \uparrow \rangle]/\alpha$.
We use d^* as a shorthand for $(d^+ \cup .)$.

- Core XPath 1.0 (the navigational core of XPath 1.0) has two types of expressions:

- **path expressions**

$$\alpha ::= d \mid d^+ \mid . \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\phi] \quad (d \in \{\uparrow, \downarrow, \leftarrow, \rightarrow\})$$

- **node expressions**

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle \alpha \rangle \quad (p \in \Sigma)$$

- Path expressions define binary relations. When applied to a given “**context node**”, they yield a set of nodes.

Node expressions define sets of nodes.

- We use $/\alpha$ as shorthand for $\uparrow^* [\neg\langle \uparrow \rangle]/\alpha$.
We use d^* as a shorthand for $(d^+ \cup .)$.

Contribute to a theory of **Codd completeness for XPath**:

(a) what are natural yardsticks of expressive power for XML?

- **first-order logic (FO)**
By analogy to relational DBs
- **monadic second-order logic (MSO)**
PTime query evaluation (data complexity),
decidable query containment,
characterization in terms of tree automata
- **first-order logic + monadic transitive closure (FO(MTC))**
Lies in-between FO and MSO

(b) identify XPath dialects that are expressively complete (and, in fact, equivalent) for each of these.

Contribute to a theory of **Codd completeness for XPath**:

(a) what are natural yardsticks of expressive power for XML?

- **first-order logic (FO)**
By analogy to relational DBs
- **monadic second-order logic (MSO)**
PTime query evaluation (data complexity),
decidable query containment,
characterization in terms of tree automata
- **first-order logic + monadic transitive closure (FO(MTC))**
Lies in-between FO and MSO

(b) identify XPath dialects that are expressively complete (and, in fact, equivalent) for each of these.

Contribute to a theory of **Codd completeness for XPath**:

(a) what are natural yardsticks of expressive power for XML?

- **first-order logic (FO)**
By analogy to relational DBs
- **monadic second-order logic (MSO)**
PTime query evaluation (data complexity),
decidable query containment,
characterization in terms of tree automata
- **first-order logic + monadic transitive closure (FO(MTC))**
Lies in-between FO and MSO

(b) identify XPath dialects that are expressively complete (and, in fact, equivalent) for each of these.

Contribute to a theory of **Codd completeness for XPath**:

(a) what are natural yardsticks of expressive power for XML?

- **first-order logic (FO)**
By analogy to relational DBs
- **monadic second-order logic (MSO)**
PTime query evaluation (data complexity),
decidable query containment,
characterization in terms of tree automata
- **first-order logic + monadic transitive closure (FO(MTC))**
Lies in-between FO and MSO

(b) identify XPath dialects that are expressively complete (and, in fact, equivalent) for each of these.

Contribute to a theory of **Codd completeness for XPath**:

(a) what are natural yardsticks of expressive power for XML?

- **first-order logic (FO)**
By analogy to relational DBs
- **monadic second-order logic (MSO)**
PTime query evaluation (data complexity),
decidable query containment,
characterization in terms of tree automata
- **first-order logic + monadic transitive closure (FO(MTC))**
Lies in-between FO and MSO

(b) identify XPath dialects that are expressively complete (and, in fact, equivalent) for each of these.

Contribute to a theory of **Codd completeness for XPath**:

- (a) what are natural yardsticks of expressive power for XML?
- **first-order logic (FO)**
By analogy to relational DBs
 - **monadic second-order logic (MSO)**
PTime query evaluation (data complexity),
decidable query containment,
characterization in terms of tree automata
 - **first-order logic + monadic transitive closure (FO(MTC))**
Lies in-between FO and MSO
- (b) identify XPath dialects that are expressively complete (and, in fact, equivalent) for each of these.

The logics

- With FO, we mean **first order logic** (i.e., the relational calculus) in the following signature:
 - The **descendant** relation $<$
 - The **sibling order** relation \prec .
 - A unary relation for each node label.
- MSO extends FO with set quantification $(\forall X, \exists X)$.
- FO(MTC) extends FO with **transitive closure for binary relations**, denoted by $[TC_{xy} \phi]$.

On trees, $FO \subsetneq FO(MTC) \subseteq? MSO$ (cf. Pothoff 1994)

The logics

- With FO, we mean **first order logic** (i.e., the relational calculus) in the following signature:
 - The **descendant** relation $<$
 - The **sibling order** relation \prec .
 - A unary relation for each node label.
- MSO extends FO with set quantification ($\forall X, \exists X$).
- FO(MTC) extends FO with **transitive closure for binary relations**, denoted by $[TC_{xy} \phi]$.

On trees, $FO \subsetneq FO(MTC) \subseteq? MSO$ (cf. Pothoff 1994)

The logics

- With FO, we mean **first order logic** (i.e., the relational calculus) in the following signature:
 - The **descendant** relation $<$
 - The **sibling order** relation \prec .
 - A unary relation for each node label.
- MSO extends FO with set quantification ($\forall X, \exists X$).
- FO(MTC) extends FO with **transitive closure for binary relations**, denoted by $[TC_{xy} \phi]$.

On trees, $FO \subsetneq FO(MTC) \subseteq? MSO$ (cf. Pothoff 1994)

- With FO, we mean **first order logic** (i.e., the relational calculus) in the following signature:
 - The **descendant** relation $<$
 - The **sibling order** relation \prec .
 - A unary relation for each node label.
- MSO extends FO with set quantification ($\forall X, \exists X$).
- FO(MTC) extends FO with **transitive closure for binary relations**, denoted by $[TC_{xy} \phi]$.

On trees, $FO \subsetneq FO(MTC) \subseteq? MSO$ (cf. Pothoff 1994)

Example

$R(x, y, z)$

x	y	z
San Francisco	Amsterdam	KLM
Amsterdam	Budapest	KLM
Amsterdam	Budapest	Malev
Budapest	Chişinău	Malev

$[TC_{xy} R(x, y, z)]$

x	y	z
San Francisco	Amsterdam	KLM
Amsterdam	Budapest	KLM
San Francisco	Budapest	KLM
Amsterdam	Budapest	Malev
Budapest	Chişinău	Malev
Amsterdam	Chişinău	Malev

$[TC_{xy} \exists z. R(x, y, z)] \dots$

Example

$R(x, y, z)$

x	y	z
San Francisco	Amsterdam	KLM
Amsterdam	Budapest	KLM
Amsterdam	Budapest	Malev
Budapest	Chişinău	Malev

$[TC_{xy} R(x, y, z)]$

x	y	z
San Francisco	Amsterdam	KLM
Amsterdam	Budapest	KLM
San Francisco	Budapest	KLM
Amsterdam	Budapest	Malev
Budapest	Chişinău	Malev
Amsterdam	Chişinău	Malev

$[TC_{xy} \exists z. R(x, y, z)] \dots$

Example

$R(x, y, z)$

x	y	z
San Francisco	Amsterdam	KLM
Amsterdam	Budapest	KLM
Amsterdam	Budapest	Malev
Budapest	Chişinău	Malev

$[TC_{xy} R(x, y, z)]$

x	y	z
San Francisco	Amsterdam	KLM
Amsterdam	Budapest	KLM
San Francisco	Budapest	KLM
Amsterdam	Budapest	Malev
Budapest	Chişinău	Malev
Amsterdam	Chişinău	Malev

$[TC_{xy} \exists z. R(x, y, z)] \dots$

What is known

- Core XPath 1.0 is **not expressively complete for FO** (it corresponds to the two-variable fragment). Adding **conditional axes** makes it FO-complete (Marx'04).
- Core XPath 2.0 is expressively complete for FO (by design)
- Some XPath extensions were proposed that are expressively complete for MSO (e.g., with **fixed point operators**). None is very attractive as a practical language.
- Regular XPath, with the **Kleene star** for transitive closure, is natural (proposed independently by a number of people).

Is Regular XPath expressively complete for MSO?

(Asked for instance by Marx 2004)

What is known

- Core XPath 1.0 is **not expressively complete for FO** (it corresponds to the two-variable fragment). Adding **conditional axes** makes it FO-complete (Marx'04).
- Core XPath 2.0 is expressively complete for FO (by design)
- Some XPath extensions were proposed that are expressively complete for MSO (e.g., with **fixed point operators**). None is very attractive as a practical language.
- Regular XPath, with the **Kleene star** for transitive closure, is natural (proposed independently by a number of people).

Is Regular XPath expressively complete for MSO?

(Asked for instance by Marx 2004)

What is known

- Core XPath 1.0 is **not expressively complete for FO** (it corresponds to the two-variable fragment). Adding **conditional axes** makes it FO-complete (Marx'04).
- Core XPath 2.0 is expressively complete for FO (by design)
- Some XPath extensions were proposed that are expressively complete for MSO (e.g., with **fixed point operators**). None is very attractive as a practical language.
- Regular XPath, with the **Kleene star** for transitive closure, is natural (proposed independently by a number of people).

Is Regular XPath expressively complete for MSO?

(Asked for instance by Marx 2004)

What is known

- Core XPath 1.0 is **not expressively complete for FO** (it corresponds to the two-variable fragment). Adding **conditional axes** makes it FO-complete (Marx'04).
- Core XPath 2.0 is expressively complete for FO (by design)
- Some XPath extensions were proposed that are expressively complete for MSO (e.g., with **fixed point operators**). None is very attractive as a practical language.
- Regular XPath, with the **Kleene star** for transitive closure, **is** natural (proposed independently by a number of people).

Is Regular XPath expressively complete for MSO?

(Asked for instance by Marx 2004)

What is known

- Core XPath 1.0 is **not expressively complete for FO** (it corresponds to the two-variable fragment). Adding **conditional axes** makes it FO-complete (Marx'04).
- Core XPath 2.0 is expressively complete for FO (by design)
- Some XPath extensions were proposed that are expressively complete for MSO (e.g., with **fixed point operators**). None is very attractive as a practical language.
- Regular XPath, with the **Kleene star** for transitive closure, **is** natural (proposed independently by a number of people).

Is Regular XPath expressively complete for MSO?

(Asked for instance by Marx 2004)

Examples of queries using transitive closure

- “Retrieve all nodes at **even distance from the root**”:
 $/(↓/↓)^*$. Can't be expressed without transitive closure.
- (In a directory structure) “Retrieve all files reachable from the current folder by **repeatedly selecting non-hidden subfolders**”.
 - Cannot be expressed in XPath 1.0.
 - Can be expressed in XPath 2.0 but not very efficiently. E.g.,

$↓^+[file]$ **except** $↓^+[folder \wedge @hidden = 'true']/↓^+[file]$

- More elegantly using transitive closure:

$(↓[folder \wedge @hidden = 'false'])^*/↓[file]$

Examples of queries using transitive closure

- “Retrieve all nodes at **even distance from the root**”:
 $/(↓/↓)^*$. Can't be expressed without transitive closure.
- (In a directory structure) “Retrieve all files reachable from the current folder by **repeatedly selecting non-hidden subfolders**”.

- Cannot be expressed in XPath 1.0.

- Can be expressed in XPath 2.0 but not very efficiently. E.g.,

$↓^+[file]$ **except** $↓^+[folder \wedge @hidden = 'true']/↓^+[file]$

- More elegantly using transitive closure:

$(↓[folder \wedge @hidden = 'false'])^*/↓[file]$

Examples of queries using transitive closure

- “Retrieve all nodes at **even distance from the root**”:
 $/(↓/↓)^*$. Can't be expressed without transitive closure.
- (In a directory structure) “Retrieve all files reachable from the current folder by **repeatedly selecting non-hidden subfolders**”.
 - Cannot be expressed in XPath 1.0.
 - Can be expressed in XPath 2.0 but not very efficiently. E.g.,

$↓^+[file]$ **except** $↓^+[folder \wedge @hidden = 'true']/↓^+[file]$

- More elegantly using transitive closure:

$(↓[folder \wedge @hidden = 'false'])^*/↓[file]$

Examples of queries using transitive closure

- “Retrieve all nodes at **even distance from the root**”:
 $/(↓/↓)^*$. Can't be expressed without transitive closure.
- (In a directory structure) “Retrieve all files reachable from the current folder by **repeatedly selecting non-hidden subfolders**”.
 - Cannot be expressed in XPath 1.0.
 - Can be expressed in XPath 2.0 but not very efficiently. E.g.,

$↓^+[file]$ **except** $↓^+[folder \wedge @hidden = 'true']/↓^+[file]$

- More elegantly using transitive closure:

$(↓[folder \wedge @hidden = 'false'])^*/↓[file]$

Examples of queries using transitive closure

- “Retrieve all nodes at **even distance from the root**”:
 $/(↓/↓)^*$. Can't be expressed without transitive closure.
- (In a directory structure) “Retrieve all files reachable from the current folder by **repeatedly selecting non-hidden subfolders**”.
 - Cannot be expressed in XPath 1.0.
 - Can be expressed in XPath 2.0 but not very efficiently. E.g.,

$↓^+[file]$ **except** $↓^+[folder \wedge @hidden = 'true']/↓^+[file]$

- More elegantly using transitive closure:

$(↓[folder \wedge @hidden = 'false'])^*/↓[file]$

Reasons for adding TC to XPath

- DTDs can be expressed inside XPath using TC, which makes **query containment relative to a DTD** a special case of **query containment**. (Marx 2004; Fan et al. 2005)
- TC enables query rewriting for recursive XML views (Fan et al. 2006).
- Useful in practice, e.g., to compute course prerequisites in an XML course catalogue (Nentwich et al., 2002).
- Incidentally TC belongs to many of the earlier query languages for semi-structured data (cf. “path regular expressions” in WebSQL, Lorel, UnQL, ...).

Reasons for adding TC to XPath

- DTDs can be expressed inside XPath using TC, which makes **query containment relative to a DTD** a special case of **query containment**. (Marx 2004; Fan et al. 2005)
- TC enables query rewriting for recursive XML views (Fan et al. 2006).
- Useful in practice, e.g., to compute course prerequisites in an XML course catalogue (Nentwich et al., 2002).
- Incidentally TC belongs to many of the earlier query languages for semi-structured data (cf. “path regular expressions” in WebSQL, Lorel, UnQL, ...).

Reasons for adding TC to XPath

- DTDs can be expressed inside XPath using TC, which makes **query containment relative to a DTD** a special case of **query containment**. (Marx 2004; Fan et al. 2005)
- TC enables query rewriting for recursive XML views (Fan et al. 2006).
- Useful in practice, e.g., to compute course prerequisites in an XML course catalogue (Nentwich et al., 2002).
- Incidentally TC belongs to many of the earlier query languages for semi-structured data (cf. “path regular expressions” in WebSQL, Lorel, UnQL, ...).

Reasons for adding TC to XPath

- DTDs can be expressed inside XPath using TC, which makes **query containment relative to a DTD** a special case of **query containment**. (Marx 2004; Fan et al. 2005)
- TC enables query rewriting for recursive XML views (Fan et al. 2006).
- Useful in practice, e.g., to compute course prerequisites in an XML course catalogue (Nentwich et al., 2002).
- Incidentally TC belongs to many of the earlier query languages for semi-structured data (cf. “path regular expressions” in WebSQL, Lorel, UnQL, ...).

Our contributions

Let **Regular XPath(W)** be the extension of Core XPath 1.0 with **transitive closure (*)** and **subtree relativisation (W)**.

$$\llbracket W\phi \rrbracket^T = \{n \in \text{dom}_T \mid n \in \llbracket \phi \rrbracket^{\text{subtree}(T,n)}\}$$

Theorem

Regular XPath(W) is equally expressive as FO(MTC).

It also admits an automata theoretic characterization, in terms of nested tree walking automata.

Theorem

FO(MTC) is strictly less expressive than MSO (on XML trees)

In particular, Regular XPath is **not** expressively complete for MSO.

Our contributions

Let **Regular XPath(W)** be the extension of Core XPath 1.0 with **transitive closure (*)** and **subtree relativisation (W)**.

$$\llbracket W\phi \rrbracket^T = \{n \in \text{dom}_T \mid n \in \llbracket \phi \rrbracket^{\text{subtree}(T,n)}\}$$

Theorem

Regular XPath(W) is equally expressive as FO(MTC).

It also admits an automata theoretic characterization, in terms of nested tree walking automata.

Theorem

FO(MTC) is strictly less expressive than MSO (on XML trees)

In particular, Regular XPath is **not** expressively complete for MSO.

Our contributions

Let **Regular XPath(W)** be the extension of Core XPath 1.0 with **transitive closure (*)** and **subtree relativisation (W)**.

$$\llbracket W\phi \rrbracket^T = \{n \in \text{dom}_T \mid n \in \llbracket \phi \rrbracket^{\text{subtree}(T,n)}\}$$

Theorem

Regular XPath(W) is equally expressive as FO(MTC).

*It also admits an automata theoretic characterization, in terms of **nested tree walking automata**.*

Theorem

FO(MTC) is strictly less expressive than MSO (on XML trees)

In particular, Regular XPath is **not** expressively complete for MSO.

Our contributions

Let **Regular XPath(W)** be the extension of Core XPath 1.0 with **transitive closure (*)** and **subtree relativisation (W)**.

$$\llbracket W\phi \rrbracket^T = \{n \in \text{dom}_T \mid n \in \llbracket \phi \rrbracket^{\text{subtree}(T,n)}\}$$

Theorem

Regular XPath(W) is equally expressive as FO(MTC).

*It also admits an automata theoretic characterization, in terms of **nested tree walking automata**.*

Theorem

FO(MTC) is strictly less expressive than MSO (on XML trees)

In particular, Regular XPath is **not** expressively complete for MSO.

Our contributions

Let **Regular XPath(W)** be the extension of Core XPath 1.0 with **transitive closure (*)** and **subtree relativisation (W)**.

$$\llbracket W\phi \rrbracket^T = \{n \in \text{dom}_T \mid n \in \llbracket \phi \rrbracket^{\text{subtree}(T,n)}\}$$

Theorem

Regular XPath(W) is equally expressive as FO(MTC).

*It also admits an automata theoretic characterization, in terms of **nested tree walking automata**.*

Theorem

FO(MTC) is strictly less expressive than MSO (on XML trees)

In particular, Regular XPath is **not** expressively complete for MSO.

Example of the use of W

“the current node has an even number of descendants”

can nicely be expressed using $*$ and W :

- Let $(\alpha \text{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$
- Let suc be shorthand for

$$\downarrow[\neg\langle\leftarrow\rangle] \cup .[\neg\langle\downarrow\rangle]/(\uparrow \text{ while } \neg\langle\rightarrow\rangle)/\rightarrow$$

(“go to the successor in depth first left-to-right ordering”).

- Then $W\langle(\text{suc}/\text{suc})^*[\neg\langle\text{suc}\rangle]\rangle$ expresses the intended node test.

Closely related to **subtree query composition** (Benedikt and Fundulaki 2005) and **forgettable past** in temporal logics (Laroussinie et al. 2002; Alur et al. 2007)

Example of the use of W

“the current node has an even number of descendants”

can nicely be expressed using $*$ and W :

- Let $(\alpha \text{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$
- Let suc be shorthand for

$$\downarrow[\neg\langle\leftarrow\rangle] \cup .[\neg\langle\downarrow\rangle]/(\uparrow \text{ while } \neg\langle\rightarrow\rangle)/\rightarrow$$

(“go to the successor in depth first left-to-right ordering”).

- Then $W\langle(\text{suc}/\text{suc})^*[\neg\langle\text{suc}\rangle]\rangle$ expresses the intended node test.

Closely related to **subtree query composition** (Benedikt and Fundulaki 2005) and **forgettable past** in temporal logics (Laroussinie et al. 2002; Alur et al. 2007)

Example of the use of W

“the current node has an even number of descendants”

can nicely be expressed using $*$ and W :

- Let $(\alpha \text{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$
- Let suc be shorthand for

$$\downarrow[\neg\langle\leftarrow\rangle] \cup .[\neg\langle\downarrow\rangle]/(\uparrow \text{ while } \neg\langle\rightarrow\rangle)/\rightarrow$$

(“go to the successor in depth first left-to-right ordering”).

- Then $W\langle(\text{suc}/\text{suc})^*[\neg\langle\text{suc}\rangle]\rangle$ expresses the intended node test.

Closely related to **subtree query composition** (Benedikt and Fundulaki 2005) and **forgettable past** in temporal logics (Laroussinie et al. 2002; Alur et al. 2007)

Example of the use of W

“the current node has an even number of descendants”

can nicely be expressed using $*$ and W :

- Let $(\alpha \text{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$
- Let suc be shorthand for

$$\downarrow[\neg\langle\leftarrow\rangle] \cup .[\neg\langle\downarrow\rangle]/(\uparrow \text{ while } \neg\langle\rightarrow\rangle)/\rightarrow$$

(“go to the successor in depth first left-to-right ordering”).

- Then $W\langle(\text{suc}/\text{suc})^*[\neg\langle\text{suc}\rangle]\rangle$ expresses the intended node test.

Closely related to **subtree query composition** (Benedikt and Fundulaki 2005) and **forgettable past** in temporal logics (Laroussinie et al. 2002; Alur et al. 2007)

Example of the use of W

“the current node has an even number of descendants”

can nicely be expressed using $*$ and W :

- Let $(\alpha \text{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$
- Let suc be shorthand for

$$\downarrow[\neg\langle\leftarrow\rangle] \cup .[\neg\langle\downarrow\rangle]/(\uparrow \text{ while } \neg\langle\rightarrow\rangle)/\rightarrow$$

(“go to the successor in depth first left-to-right ordering”).

- Then $W\langle(\text{suc}/\text{suc})^*[\neg\langle\text{suc}\rangle]\rangle$ expresses the intended node test.

Closely related to **subtree query composition** (Benedikt and Fundulaki 2005) and **forgettable past** in temporal logics (Laroussinie et al. 2002; Alur et al. 2007)

Our contributions (continued)

Computational complexity of Regular XPath(W):

- **Query evaluation:** PTime (combined complexity)
(PTime for Core XPath 1.0; PSpace for Core XPath 2.0)
- **Query containment:** 2ExpTime-c (ExpTime without W)
(ExpTime for Core XPath 1.0; Non-elem for Core XPath 2.0)

Core XPath 2.0 \equiv FO

- Query eval: PSpace
- Query cont: Non-elem

Regular XPath(W) \equiv FO(MTC)

- Query eval: PTime
- Query cont: 2ExpTime

Core XPath 1.0 \equiv FO²

- Query eval: PTime
- Query cont: ExpTime

On the other hand, Regular XPath(W) is non-elementarily less succinct than Core XPath 2.0

Our contributions (continued)

Computational complexity of Regular XPath(W):

- **Query evaluation:** PTime (combined complexity)
(PTime for Core XPath 1.0; PSpace for Core XPath 2.0)
- **Query containment:** 2ExpTime-c (ExpTime without W)
(ExpTime for Core XPath 1.0; Non-elem for Core XPath 2.0)

Core XPath 2.0 \equiv FO

- Query eval: PSpace
- Query cont: Non-elem

Regular XPath(W) \equiv FO(MTC)

- Query eval: PTime
- Query cont: 2ExpTime

Core XPath 1.0 \equiv FO²

- Query eval: PTime
- Query cont: ExpTime

On the other hand, Regular XPath(W) is non-elementarily less succinct than Core XPath 2.0

Our contributions (continued)

Computational complexity of Regular XPath(W):

- **Query evaluation:** PTime (combined complexity)
(PTime for Core XPath 1.0; PSpace for Core XPath 2.0)
- **Query containment:** 2ExpTime-c (ExpTime without W)
(ExpTime for Core XPath 1.0; Non-elem for Core XPath 2.0)

Core XPath 2.0 \equiv FO

- Query eval: PSpace
- Query cont: Non-elem

Regular XPath(W) \equiv FO(MTC)

- Query eval: PTime
- Query cont: 2ExpTime

Core XPath 1.0 \equiv FO²

- Query eval: PTime
- Query cont: ExpTime

On the other hand, Regular XPath(W) is non-elementarily less succinct than Core XPath 2.0

Our contributions (continued)

Computational complexity of Regular XPath(W):

- **Query evaluation:** **PTime** (combined complexity)
(PTime for Core XPath 1.0; PSpace for Core XPath 2.0)
- **Query containment:** **2ExpTime-c** (**ExpTime** without W)
(ExpTime for Core XPath 1.0; Non-elem for Core XPath 2.0)

Core XPath 2.0 \equiv **FO**

- Query eval: **PSpace**
- Query cont: **Non-elem**

Regular XPath(W) \equiv **FO(MTC)**

- Query eval: **PTime**
- Query cont: **2ExpTime**

Core XPath 1.0 \equiv **FO²**

- Query eval: **PTime**
- Query cont: **ExpTime**

On the other hand, Regular XPath(W) is **non-elementarily less succinct** than Core XPath 2.0

Our contributions (continued)

Computational complexity of Regular XPath(W):

- **Query evaluation:** PTime (combined complexity)
(PTime for Core XPath 1.0; PSpace for Core XPath 2.0)
- **Query containment:** 2ExpTime-c (ExpTime without W)
(ExpTime for Core XPath 1.0; Non-elem for Core XPath 2.0)

Core XPath 2.0 \equiv FO

- Query eval: PSpace
- Query cont: Non-elem

Regular XPath(W) \equiv FO(MTC)

- Query eval: PTime
- Query cont: 2ExpTime

\ /
Core XPath 1.0 \equiv FO²

- Query eval: PTime
- Query cont: ExpTime

On the other hand, Regular XPath(W) is **non-elementarily less succinct** than Core XPath 2.0

Some corollaries:

- Regular XPath(W) is closed under path intersection and path complementation.
- FO(MTC) has the 4-variable property: every formula in four free variables can be written using 4 variables in total.

Some corollaries:

- Regular XPath(W) is closed under **path intersection** and **path complementation**.
- FO(MTC) has the **4-variable property**: every formula in four free variables can be written using 4 variables in total.

Some corollaries:

- Regular XPath(W) is closed under **path intersection** and **path complementation**.
- FO(MTC) has the **4-variable property**: every formula in four free variables can be written using 4 variables in total.

Proof that Regular XPath(W) \equiv FO(MTC)

Outline of difficult direction (FO(MTC) \subseteq Regular XPath(W)):

- 1 May assume binary branching trees.
- 2 May assume that FO(MTC) formulas are in a **normal form**: allow only TCs of the form

$$\left[TC_{xy} \phi(x, y, u, v) \wedge u < x, y \wedge v \not< x, y \right] (u, v)$$

(An Immerman-Kozen style argument)

Proof that Regular XPath(W) \equiv FO(MTC)

Outline of difficult direction (FO(MTC) \subseteq Regular XPath(W)):

- 1 May assume binary branching trees.
- 2 May assume that FO(MTC) formulas are in a **normal form**: allow only TCs of the form

$$\left[TC_{xy} \phi(x, y, u, v) \wedge u < x, y \wedge v \not< x, y \right] (u, v)$$

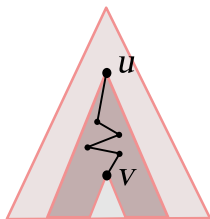
(An Immerman-Kozen style argument)

Proof that Regular XPath(W) \equiv FO(MTC)

Outline of difficult direction (FO(MTC) \subseteq Regular XPath(W)):

- 1 May assume binary branching trees.
- 2 May assume that FO(MTC) formulas are in a **normal form**: allow only TCs of the form

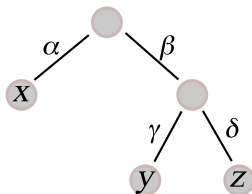
$$\left[TC_{xy} \phi(x, y, u, v) \wedge u < x, y \wedge v \not< x, y \right] (u, v)$$



(An Immerman-Kozen style argument)

Proof that Regular XPath(W) \equiv FO(MTC)

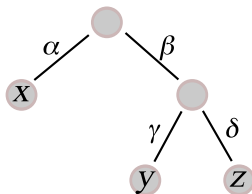
- 3 Defining *n*-ary queries in Regular XPath(W) using tree patterns: tree shaped conjunctive queries containing path expressions as atomic relations.



- 4 Inductive translation from normal form FO(MTC) to unions of tree patterns. The translation is non-elementary. The most difficult inductive step is for the (normalized) TC operator.

Proof that Regular XPath(W) \equiv FO(MTC)

- 3 Defining *n*-ary queries in Regular XPath(W) using **tree patterns**: tree shaped conjunctive queries containing path expressions as atomic relations.



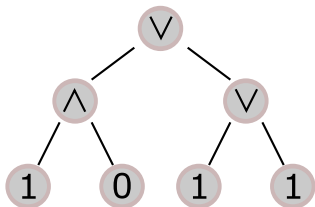
- 4 Inductive translation from **normal form FO(MTC)** to **unions of tree patterns**. The translation is non-elementary. The most difficult inductive step is for the (normalized) TC operator.

Outline of the proof that $\text{FO}(\text{MTC}) \subsetneq \text{MSO}$

- Main ingredient: an automata model that captures exactly $\text{FO}(\text{MTC})$.

Tree automata

A standard (bottom-up) tree automaton in action:



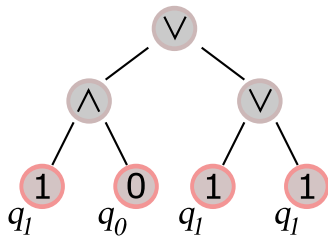
Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, . . .)
- Equivalent to MSO.
- But no longer purely sequential.

Tree automata

A standard (bottom-up) tree automaton in action:



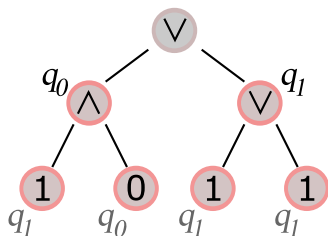
Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, ...)
- Equivalent to MSO.
- But no longer purely sequential.

Tree automata

A standard (bottom-up) tree automaton in action:



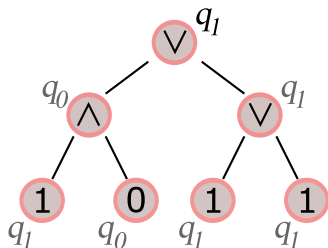
Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, . . .)
- Equivalent to MSO.
- But no longer purely sequential.

Tree automata

A standard (bottom-up) tree automaton in action:



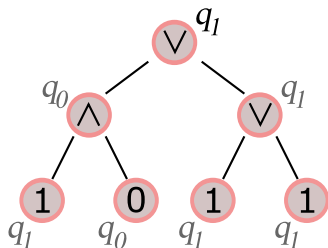
Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, ...)
- Equivalent to MSO.
- But no longer purely sequential.

Tree automata

A standard (bottom-up) tree automaton in action:



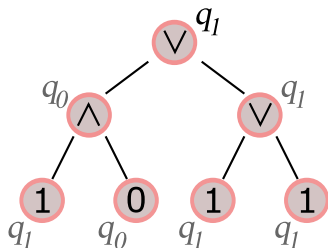
Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, ...)
- Equivalent to MSO.
- But no longer purely sequential.

Tree automata

A standard (bottom-up) tree automaton in action:



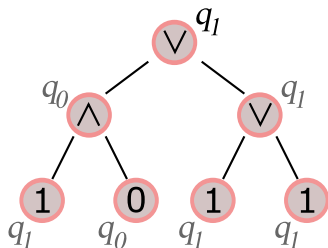
Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, ...)
- Equivalent to MSO.
- But no longer purely sequential.

Tree automata

A standard (bottom-up) tree automaton in action:



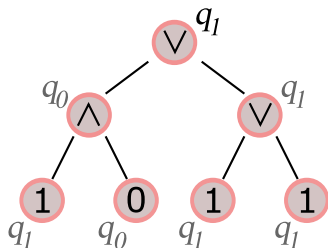
Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, ...)
- Equivalent to MSO.
- But no longer purely sequential.

Tree automata

A standard (bottom-up) tree automaton in action:



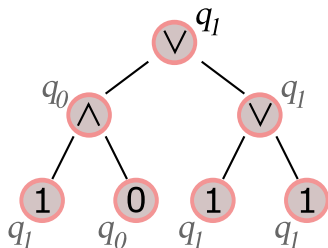
Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, ...)
- Equivalent to MSO.
- But no longer purely sequential.

Tree automata

A standard (bottom-up) tree automaton in action:



Essentially computes a **fixed point**.

Are tree automata a “natural” generalization of string automata?

- The same good properties (determinization, minimization, closure, . . .)
- Equivalent to MSO.
- But no longer purely sequential.

Tree walking automaton

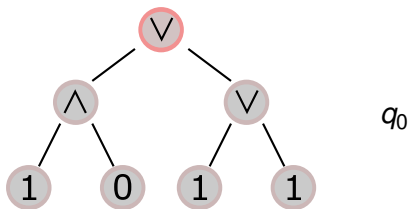
Is there a more **sequential** type of automata for trees?

Aho and Ullman 1971 define **tree walking automata**:

Tree walking automaton

Is there a more **sequential** type of automata for trees?

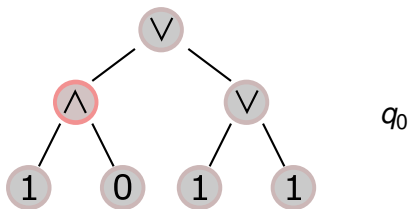
Aho and Ullman 1971 define **tree walking automata**:



Tree walking automaton

Is there a more **sequential** type of automata for trees?

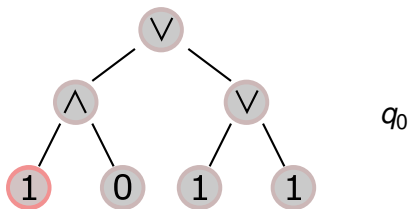
Aho and Ullman 1971 define **tree walking automata**:



Tree walking automaton

Is there a more **sequential** type of automata for trees?

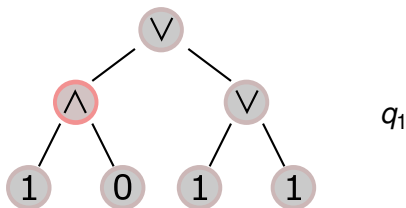
Aho and Ullman 1971 define **tree walking automata**:



Tree walking automaton

Is there a more **sequential** type of automata for trees?

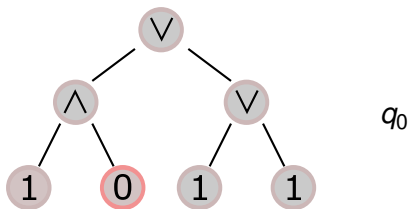
Aho and Ullman 1971 define **tree walking automata**:



Tree walking automaton

Is there a more **sequential** type of automata for trees?

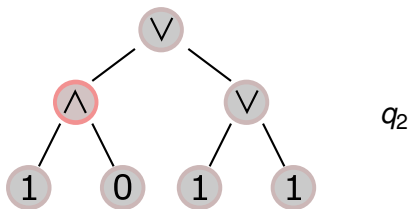
Aho and Ullman 1971 define **tree walking automata**:



Tree walking automaton

Is there a more **sequential** type of automata for trees?

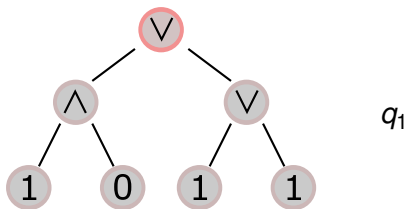
Aho and Ullman 1971 define **tree walking automata**:



Tree walking automaton

Is there a more **sequential** type of automata for trees?

Aho and Ullman 1971 define **tree walking automata**:



- **Theorem (Bojańczyk and Colcombet 2005):**
Twa do not capture all of MSO.
- **Theorem (Bojańczyk, Samuelides, Schwentick and Segoufin 2006; Engelfriet and Hoogeboom 1999):**
Twa with finitely many pebbles do not capture all of MSO (they capture FO(posMTC)).

- **Theorem (Bojańczyk and Colcombet 2005):**
Twa do not capture all of MSO.
- **Theorem (Bojańczyk, Samuelides, Schwentick and Segoufin 2006; Engelfriet and Hoogeboom 1999):**
Twa with finitely many pebbles do not capture all of MSO (they capture FO(posMTC)).

Nested tree walking automata

Definition (Nested tree walking automata)

A **nested twa** (of degree k) is a twa A with finitely many sub-automata (of degree $< k$), that can perform these tests:

- does subautomaton B have an accepting run from the current node or not ?
- does subautomaton B have an accepting run from the current node **inside the current subtree** or not?

We show:

- nested twa capture FO(MTC),
- nested twa do not capture all of MSO (same counterexample as for pebble twa).

Definition (Nested tree walking automata)

A **nested twa** (of degree k) is a twa A with finitely many sub-automata (of degree $< k$), that can perform these tests:

- does subautomaton B have an accepting run from the current node or not ?
- does subautomaton B have an accepting run from the current node **inside the current subtree** or not?

We show:

- nested twa capture FO(MTC),
- nested twa do not capture all of MSO
(same counterexample as for pebble twa).

Definition (Nested tree walking automata)

A **nested twa** (of degree k) is a twa A with finitely many sub-automata (of degree $< k$), that can perform these tests:

- does subautomaton B have an accepting run from the current node or not ?
- does subautomaton B have an accepting run from the current node **inside the current subtree** or not?

We show:

- nested twa capture FO(MTC),
- nested twa do not capture all of MSO (same counterexample as for pebble twa).

Definition (Nested tree walking automata)

A **nested twa** (of degree k) is a twa A with finitely many sub-automata (of degree $< k$), that can perform these tests:

- does subautomaton B have an accepting run from the current node or not ?
- does subautomaton B have an accepting run from the current node **inside the current subtree** or not?

We show:

- nested twa capture FO(MTC),
- nested twa do not capture all of MSO
(same counterexample as for pebble twa).

Definition (Nested tree walking automata)

A **nested twa** (of degree k) is a twa A with finitely many sub-automata (of degree $< k$), that can perform these tests:

- does subautomaton B have an accepting run from the current node or not ?
- does subautomaton B have an accepting run from the current node **inside the current subtree** or not?

We show:

- nested twa capture FO(MTC),
- nested twa do not capture all of MSO (same counterexample as for pebble twa).

How do **nested twa** compare to **pebble twa**?

- Nested twa capture $FO(MTC)$
Pebble twa capture $FO(posMTC)$
(NB: we do not know whether $FO(posMTC) \subseteq FO(MTC)$)
- Translating $FO(MTC)$ to nested twa: **non-elementary blowup**
Translating $FO(posMTC)$ to pebble twa: **PTime or non-elem.**
(depends on type of pebbles)
- Emptiness for nested twa: **2ExpTime-complete**
Emptiness for pebble twa: **non-elementary hard**

How do **nested twa** compare to **pebble twa**?

- Nested twa capture $FO(MTC)$
Pebble twa capture $FO(posMTC)$

(NB: we do not know whether $FO(posMTC) \subseteq FO(MTC)$)

- Translating $FO(MTC)$ to nested twa: non-elementary blowup
Translating $FO(posMTC)$ to pebble twa: PTime or non-elem.
(depends on type of pebbles)
- Emptiness for nested twa: 2ExpTime-complete
Emptiness for pebble twa: non-elementary hard

How do **nested twa** compare to **pebble twa**?

- Nested twa capture $FO(MTC)$
Pebble twa capture $FO(posMTC)$
(NB: we do not know whether $FO(posMTC) \subsetneq FO(MTC)$)
- Translating $FO(MTC)$ to nested twa: non-elementary blowup
Translating $FO(posMTC)$ to pebble twa: PTime or non-elem.
(depends on type of pebbles)
- Emptiness for nested twa: 2ExpTime-complete
Emptiness for pebble twa: non-elementary hard

How do **nested twa** compare to **pebble twa**?

- Nested twa capture $FO(MTC)$
Pebble twa capture $FO(posMTC)$
(NB: we do not know whether $FO(posMTC) \subsetneq FO(MTC)$)
- Translating $FO(MTC)$ to nested twa: **non-elementary blowup**
Translating $FO(posMTC)$ to pebble twa: **PTime or non-elem.**
(depends on type of pebbles)
- Emptiness for nested twa: **2ExpTime-complete**
Emptiness for pebble twa: **non-elementary hard**

How do **nested twa** compare to **pebble twa**?

- Nested twa capture $FO(MTC)$
Pebble twa capture $FO(posMTC)$
(NB: we do not know whether $FO(posMTC) \subsetneq FO(MTC)$)
- Translating $FO(MTC)$ to nested twa: **non-elementary blowup**
Translating $FO(posMTC)$ to pebble twa: **PTime or non-elem.**
(depends on type of pebbles)
- Emptiness for nested twa: **2ExpTime-complete**
Emptiness for pebble twa: **non-elementary hard**

- FO(MTC) is a **natural** yardstick of expressive power on trees, **strictly** in between FO and MSO.
- Regular XPath(W) \equiv FO(MTC)
- All our results generalize to infinite trees.
- **Question:** are there generalized quantifiers Q_1, \dots, Q_n such that $\text{FO}(Q_1, \dots, Q_n) \equiv \text{MSO}$?

- FO(MTC) is a **natural** yardstick of expressive power on trees, **strictly** in between FO and MSO.
- Regular XPath(W) \equiv FO(MTC)
- All our results generalize to infinite trees.
- **Question:** are there generalized quantifiers Q_1, \dots, Q_n such that $\text{FO}(Q_1, \dots, Q_n) \equiv \text{MSO}$?

- FO(MTC) is a **natural** yardstick of expressive power on trees, **strictly** in between FO and MSO.
- Regular XPath(W) \equiv FO(MTC)
- All our results generalize to infinite trees.
- **Question:** are there generalized quantifiers Q_1, \dots, Q_n such that $\text{FO}(Q_1, \dots, Q_n) \equiv \text{MSO}$?

- FO(MTC) is a **natural** yardstick of expressive power on trees, **strictly** in between FO and MSO.
- Regular XPath(W) \equiv FO(MTC)
- All our results generalize to infinite trees.
- **Question:** are there generalized quantifiers Q_1, \dots, Q_n such that $\text{FO}(Q_1, \dots, Q_n) \equiv \text{MSO}$?