

Logic, Databases, and Formal Languages

Sebastian Maneth
NICTA & UNSW

Logic Summer School - Canberra, December 5th, 2006

LOGIC

$$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$$

$$(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$$

$$(\forall x)(\forall y) ((x=y) \vee \neg(x=y))$$

LOGIC

$$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$$

$$(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$$

$$(\forall x)(\forall y) ((x=y) \vee \neg(x=y))$$

over relational structures

$(\exists x)(\exists y) \text{parent}(x, y) \wedge \text{reports_to}(x, y)?$

DATABASES

LOGIC

$$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$$

$$(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$$

$$(\forall x)(\forall y) ((x=y) \vee \neg(x=y))$$

over relational structures

over ordered (acyclic) structures

$(\exists x)(\exists y) \text{parent}(x, y) \wedge \text{reports_to}(x, y)?$

DATABASES

FORMAL LANGUAGES & Automata

10001010

$(\forall x)(\forall y)(\exists z)(\text{one}(x) \wedge \text{one}(y) \wedge x < y \rightarrow (x < z < y \wedge \neg \text{one}(z)))$

$(0^* (\lambda | 10))^* (\lambda | 1)$

COMPUTATION / COMPLEXITY

$(\exists k)(\neg \text{BIT}(k,0) \wedge \neg(\exists i)(\exists j)(\text{SUM}(i, j, k) \wedge \text{PRIME}(i) \wedge \text{PRIME}(j)))?$

k is even

LOGIC

$$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$$

$$(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$$

$$(\forall x)(\forall y) ((x=y) \vee \neg(x=y))$$

over relational structures

over ordered (acyclic) structures

$(\exists x)(\exists y) \text{parent}(x, y) \wedge \text{reports_to}(x, y)?$

DATABASES

FORMAL LANGUAGES & Automata

10001010

$(\forall x)(\forall y)(\exists z)(\text{one}(x) \wedge \text{one}(y) \wedge x < y \rightarrow (x < z < y \wedge \neg \text{one}(z)))$

$(0^* (\lambda | 10))^* (\lambda | 1)$

COMPUTATION / COMPLEXITY

$(\exists k)(\neg \text{BIT}(k,0) \wedge \neg(\exists i)(\exists j)(\text{SUM}(i, j, k) \wedge \text{PRIME}(i) \wedge \text{PRIME}(j)))?$

k is even

LOGIC

$$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$$

$$(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$$

$$(\forall x)(\forall y) ((x=y) \vee \neg(x=y))$$

over relational structures

over ordered (acyclic) structures

$(\exists x)(\exists y) \text{parent}(x, y) \wedge \text{reports_to}(x, y)?$

DATABASES

Automata \rightarrow VERIFICATION!

10001010

$(\forall x)(\forall y)(\exists z)(\text{one}(x) \wedge \text{one}(y) \wedge x < y \rightarrow (x < z < y \wedge \neg \text{one}(z)))$

$(0^* (\lambda | 10))^* (\lambda | 1)$

Outline

(2) Computation / complexity theory
numbers

(1) Databases / finite model theory
relational structures

(3) Automata / language theory
ordered structures



(1) DATABASES

ReportsTo		manager			
employee			•	•	•
•	0	0	1		
•	1	0	0		
•	...				

DATABASES

ReportsTo		manager			
employee			•	•	•
•	0	0	1		
•	1	0	0		
•	...				

DATABASES

ReportsTo		manager			
employee			•	•	•
•	0	0	1		
•	1	0	0		
•	...				

Query "find employees with their manager's manager"

DATABASES

ReportsTo		manager			
employee			•	•	•
•	0	0	1		
•	1	0	0		
•	...				

Query "find employees with their manager's manager"

in SQL: `select R1.employee, R2.manager
from ReportsTo R1, ReportsTo R2
where R1.manager=R2.employee`

DATABASES

ReportsTo		manager			
employee			•	•	•
•	0	0	1		
•	1	0	0		
•	...				

Query "find employees with their manager's manager"

in SQL: `select R1.employee, R2.manager
from ReportsTo R1, ReportsTo R2
where R1.manager=R2.employee`

In First-Order Logic (FO):

$$\phi(x, y) \equiv (\exists z) (\text{ReportsTo}(x, z) \wedge \text{ReportsTo}(z, y))$$

Free variables ϕ defines a **binary relation!**

→ new relational structure is defined when interpreting formulae with free variables, inside a given structure.

DATABASES

Query "find employees with their manager's manager"

in SQL: `select R1. empl oyee, R2. manager
from ReportsTo R1, ReportsTo R2
where R1. manager=R2. empl oyee`

In First-Order Logic (FO):

$$\phi(x, y) \equiv (\exists z) (\text{ReportsTo}(x,z) \wedge \text{ReportsTo}(z,y))$$

$$\psi(x, y) \equiv (\exists z)(\text{ReportsTo}(x,z) \wedge \phi(z,y))$$

DATABASES

Query "find employees with their manager's manager"

in SQL: `select R1. empl oyee, R2. manager
from ReportsTo R1, ReportsTo R2
where R1. manager=R2. empl oyee`

In First-Order Logic (FO):

$$\phi(x, y) \equiv (\exists z) (\text{ReportsTo}(x,z) \wedge \text{ReportsTo}(z,y))$$

$$\psi(x, y) \equiv (\exists z)(\text{ReportsTo}(x,z) \wedge \phi(z,y))$$

$$\vdots$$

"find employees and *all managers above*" ← FO-expressible?
(all nodes reachable via ReportsTo edges)

DATABASES

Fix a **query language / logic L** and a **class of structures C**.

Expressiveness
Which queries can be expressed in L on C, and which not?
How to "prove" that a query canNOT be expressed?
→ e.g. *transitive closure not FO-expressible!*

DATABASES

Fix a **query language / logic L** and a **class of structures C**.

Expressiveness
Which queries can be expressed in L on C, and which not?
How to "prove" that a query canNOT be expressed?
→ e.g. *transitive closure not FO-expressible!*

Decidability
Is there an *algorithm* which can check whether ϕ holds for a given s ?
whether ϕ holds for some s ?
whether ϕ holds for all s ?
whether ϕ is equivalent to ψ ?
whether ϕ is minimal?
etc.

Complexity
If there is an algorithm, what is its worst-case time/space complexity?
Can it be improved? What is the lower bound?

DATABASES

Fix a **query language / logic L** and a **class of structures C**.

Expressiveness, examples:
(a) transitive closure (TC) not FO-expressible.
(b) "x and y have the same number of managers above"
not (FO+TC)-expressible.
Both (a),(b) are MSO-expressible (MSO = FO + set-quantification)

DATABASES

Fix a **query language / logic L** and a **class of structures C**.

Expressiveness, examples:
(a) transitive closure (TC) not FO-expressible.
(b) "x and y have the same number of managers above"
not (FO+TC)-expressible.
Both (a),(b) are MSO-expressible (MSO = FO + set-quantification)

Decidability
→ e.g. There is *no* algorithm which can check whether an arbitrary FO-formula holds for all finite graphs. [Trakhtenbrot1950]
→ e.g. There is an algorithm which can check whether an arbitrary MSO-formula holds for all finite trees. [Thatcher/Wright1968]

DATABASES

Fix a **query language / logic L** and a **class of structures C**.

Expressiveness, examples:

- (a) transitive closure (TC) not **FO**-expressible.
- (b) "x and y have the same number of managers above" not **(FO+TC)**-expressible.

Both (a),(b) are **MSO**-expressible (**MSO = FO + set-quantification**)

Decidability

→ e.g. There is **no** algorithm which can check whether an arbitrary **FO**-formula holds for **all finite graphs**. [Trakhtenbrot1950]

→ e.g. There is an algorithm which can check whether an arbitrary **MSO**-formula holds for **all finite trees**. [Thatcher/Wright1968]

Complexity

→ e.g. Checking whether $g \models \phi$, for a graph g and a **FO** ϕ can be done in time polynomial in the sizes of g and ϕ .

→ e.g. There is an **MSO** formula ϕ such that checking whether it hold for some **string**, takes **at least** time $2^{n^2} \cdot 2^{n^2} \dots 2^{n^2}$ (height=size(ϕ))

DATABASES

Fix a **query language / logic L=MSO** and a **class of structures C**.

MSO = First-Order Logic plus quantification over monadic predicates (=sets)

Lemma

Let S be a structure which has a **binary relation R**.
The **transitive closure** of R is defined by the following **MSO** formula:

DATABASES

Fix a **query language / logic L=MSO** and a **class of structures C**.

MSO = First-Order Logic plus quantification over monadic predicates (=sets)

Lemma

Let S be a structure which has a **binary relation R**.
The **transitive closure** of R is defined by the following **MSO** formula:

$$\begin{aligned}
 & (\forall X) \\
 & [\\
 & \quad [(\forall y) (\forall z) (y \in X \wedge R(y,z) \rightarrow z \in X) \wedge (\forall u) (R(x_1, u) \rightarrow u \in X)] \rightarrow x_2 \in X \\
 &] \\
 & \quad \underbrace{\hspace{10em}}_{X \text{ is "R-closed"}} \quad \underbrace{\hspace{10em}}_{\text{For transitive, reflexive closure, simply: } x_1 \in X}
 \end{aligned}$$

(this formula has two **free variables** x_1, x_2)

edge*(x,y) \equiv $(\forall X) [((\forall u)(\forall v) (u \in X \wedge \text{edge}(u,v) \rightarrow v \in X) \wedge x \in X] \rightarrow y \in X]$

Fix a **query language / logic L=MSO** and a **class of structures C**.

Examples of MSO definable graphs (properties)

- connected
- Hamiltonian (= has Hamiltonian cycle)
- k-colorable
- is string / tree / dag
- planar (use Kuratowski's Theorem)
- ...

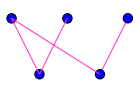
Fix a **query language / logic L=MSO** and a **class of structures C**.

Examples of MSO definable graphs (properties)

- connected
- Hamiltonian (= has Hamiltonian cycle)
- k-colorable
- is string / tree / dag
- planar (use Kuratowski's Theorem)
- ...

E.g.: **bipartite graphs**: no labels, undirected graphs, no self-loops
set of nodes can be partitioned into sets X, Y such that
an edge between x and y implies that $x \in X$ and $y \in Y$ (or vice versa)

$$\text{part}(X, Y) \equiv (\forall z) (z \in X \vee z \in Y) \wedge \neg(z \in X \wedge z \in Y)$$

$$\text{bipartite} \equiv (\exists X)(\exists Y) \text{part}(X, Y) \wedge (\forall u)(\forall v) \text{edge}(u,v) \rightarrow (u \in X \wedge v \in Y) \vee (u \in Y \wedge v \in X)$$


bipartite graph g

Fix a **query language / logic L=MSO** and a **class of structures C**.

Examples of MSO definable graphs (properties)

- connected
- Hamiltonian (= has Hamiltonian cycle)
- k-colorable
- is string / tree / dag
- planar (use Kuratowski's Theorem)
- ...

Final Note

To determine for an **arbitrary graph**, whether it satisfies an **MSO-property** is in general **computationally intractable (hard!!)**.

For instance, Hamiltonian is **NP-complete** → best known algorithms need **exponential running time** (in #nodes)!

Fix a **query language / logic** $L=MSO$ and a **class of structures** C .

Examples of MSO definable graphs (properties)

- connected
- Hamiltonian (= has Hamiltonian cycle)
- k-colorable
- is string / tree / dag
- planar (use Kuratowski's Theorem)
- ...

Final Note

To determine for an **arbitrary graph**, whether it satisfies an MSO-property is in general **computationally intractable (hard!!)**.

For instance, Hamiltonian is **NP-complete** → best known algorithms need **exponential running time** (in #nodes)!

... but ... if the graphs are **"tree-like"** (= hierarchical = built from smaller graphs..) then we have **fast (linear time) algorithms!!**

→ FORMAL LANGUAGES

Fix a **query language / logic** $L=MSO$ and a **class of structures** C .

Examples of MSO definable graphs (properties)

- connected
- Hamiltonian (= has Hamiltonian cycle)
- k-colorable
- is string / tree / dag
- planar (use Kuratowski's Theorem)
- ...

Final Note

To determine for an **arbitrary graph**, whether it satisfies an MSO-property is in general **computationally intractable (hard!!)**.

For instance, Hamiltonian is **NP-complete** → best known algorithms will need **exponential running time** (in #nodes)!

... but ... if the graphs are **"tree-like"** (= hierarchical = built from smaller graphs..) then we have **fast (linear time) algorithms!!**

→ FORMAL LANGUAGES

(2) COMPUTATION / COMPLEXITY

$(\exists X)(\forall y)(\forall z) (X(y, z) \leftrightarrow \text{TIMES}(y, y, z))$

↑ **over BIT-sequences (numbers)**

LOGIC

↓ **over ordered (acyclic) structures**

1 0 0 0 1 0 1 0

$(\forall x)(\forall y)(\exists z)(\text{one}(x) \wedge \text{one}(y) \wedge x < y \rightarrow x < z < y \wedge \neg \text{one}(z))$

$(0^* (\lambda | 10))^* (\lambda | 1)$

→ **FORMAL LANGUAGES & Automata**

(2) COMPUTATION / COMPLEXITY

NP = SO₃
[R. Fagin, 1974]

(2) COMPUTATION / COMPLEXITY

→ "Descriptive Complexity"
→ "Finite Model Theory"
(Downey / Fellows)

(2) COMPLEXITY ... logic with the BIT

Look at **ordered structures**.
Identify universe with numbers 0, 1, 2, ..., n-1

Numeric relations:

- PLUS(i, j, k) meaning $i + j = k$
- TIMES(i, j, k) meaning $i \cdot j = k$
- BIT(i, j) meaning j-th bit in the binary representation of i is 1

BIT(k,0) ≡ "k is odd" = 0-th bit in binary expansion of k is set (=1)

e.g. k=7 in binary: 0...0 1 1 1

BIT(7,0) ∧
BIT(7,1) ∧
BIT(7,2) ∧
¬BIT(7, j), j ≥ 3

0-th bit
1-st bit

(2) COMPLEXITY ... logic with the BIT

Look at *ordered structures*.
Identify universe with numbers $0, 1, 2, \dots, n-1$

Numeric relations:

PLUS(i, j, k) meaning $i + j = k$
 TIMES(i, j, k) meaning $i \cdot j = k$
 BIT(i, j) meaning j -th bit in the binary representation of i is 1

Theorem

Let C be a class of structures that includes ordering ($<$)

- If C has **BIT**, then PLUS and TIMES are FO-definable.
- If C has PLUS and TIMES, then **BIT** is FO-definable.

(2) COMPLEXITY ... logic with the BIT

- If C has **BIT**, then PLUS and TIMES are FO-definable.

"carry-look-ahead" algorithm (write $A(z)$ for **BIT**(i, z) and $B(z)$ for **BIT**(j, z))

$$\text{carry}(x) \equiv (\exists y. x < y)[A(y) \wedge B(y) \wedge (\forall z. x < z < y) A(z) \vee B(z)]$$

A	1	1	1	0	1	0
B	1	0	0	1	1	0
C	\square					

x y

(2) COMPLEXITY ... logic with the BIT

- If C has **BIT**, then PLUS and TIMES are FO-definable.

"carry-look-ahead" algorithm (write $A(z)$ for **BIT**(i, z) and $B(z)$ for **BIT**(j, z))

$$\text{carry}(x) \equiv (\exists y. x < y)[A(y) \wedge B(y) \wedge (\forall z. x < z < y) A(z) \vee B(z)]$$

A	1	1	1	0	1	0	$p \oplus q = p \leftrightarrow \neg q$
B	1	0	0	1	1	0	
C	\square						$D(x) \equiv A(x) \oplus B(x) \oplus C(x)$

x y

(2) COMPLEXITY ... logic with the BIT

- If C has **BIT**, then PLUS and TIMES are FO-definable.

"carry-look-ahead" algorithm (write $A(z)$ for **BIT**(i, z) and $B(z)$ for **BIT**(j, z))

$$\text{carry}(x) \equiv (\exists y. x < y)[A(y) \wedge B(y) \wedge (\forall z. x < z < y) A(z) \vee B(z)]$$

A	1	1	1	0	1	0	$p \oplus q = p \leftrightarrow \neg q$
B	1	0	0	1	1	0	
C	\square						$D(x) \equiv A(x) \oplus B(x) \oplus C(x)$

x y

In other words:

$$\text{PLUS}(i, j, k) \equiv (\forall x) \text{BIT}(k, x) \leftrightarrow (\text{BIT}(i, x) \oplus \text{BIT}(j, x) \oplus \text{carry}(x))$$

(2) COMPLEXITY ... logic with the BIT

- If C has **BIT**, then PLUS and TIMES are FO-definable.

Use the "Bit Sum Lemma":

$$\text{BSUM}(x, y) \equiv \text{"y is equal to the number of 1's in the binary representation of x"}$$

Lemma
BSUM is FO-expressible, using ordering and **BIT**.

Proof is nontrivial! Idea: define a "running sum".

(2) COMPLEXITY ... logic with the BIT

- If C has **BIT**, then PLUS and TIMES are FO-definable.

Use the "Bit Sum Lemma":

$$\text{BSUM}(x, y) \equiv \text{"y is equal to the number of 1's in the binary representation of x"}$$

Lemma
BSUM is FO-expressible, using ordering and **BIT**.

Proof is nontrivial! Idea: define a "running sum".

QUESTION

How can you define the **BIT**-predicate, using PLUS and TIMES?

(2) COMPLEXITY ... logic with the BIT

1. If C has BIT, then ORDER (<) is FO-definable!!

Really interesting proof! [Dawar/Doets/Lindell/Weinstein1998]

How do you define order on binary numbers, using ONLY the BIT predicate??

$$\text{LESS}(A, B) \equiv (\exists x)(B(x) \wedge \neg A(x) \wedge (\forall y. y > x)(A(y) \rightarrow B(y)))$$

A 1 0 0 0 1 1
B 1 0 0 1 1 0

x

still uses ordering on positions.. ☹

(2) COMPLEXITY ... logic with the BIT

1. If C has BIT, then ORDER (<) is FO-definable!!

Really interesting proof! [Dawar/Doets/Lindell/Weinstein1998]

How do you define order on binary numbers, using ONLY the BIT predicate??

$$\text{LESS}(A, B) \equiv (\exists x)(B(x) \wedge \neg A(x) \wedge (\forall y. y > x)(A(y) \rightarrow B(y)))$$

A 1 0 0 0 1 1
B 1 0 0 1 1 0

x

still uses ordering on positions.. ☹

Q: how to get rid of the ">" on positions..?

(2) COMPUTATION + COMPLEXITY

$(\exists x)(\forall y)(\forall z)(X(x, z) \wedge \dots \text{TIMES}(y, z))$

over BIT sequences (problems)

LOGIC

$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$
 $(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$
 $(\forall x)(\forall y)((x=y) \vee \neg(x=y))$

1 0 0 0 1 0 1 0

over ordered (acyclic) structures

$(\forall x)(\forall y)(\exists z)(\text{one}(x) \wedge \text{one}(y) \wedge x < y \rightarrow x < z < y \wedge \neg \text{one}(z))$

$(0^* (\lambda | 10)^* (\lambda | 1))$

(1) DATABASES

(3) FORMAL LANGUAGES & Automata

LOGIC

$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$
 $(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$
 $(\forall x)(\forall y)((x=y) \vee \neg(x=y))$

Tree Decomposable Graphs

Trees

Chomsky Context-Free Languages

(3) FORMAL LANGUAGES & Automata

MSO

$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$
 $(p \rightarrow q) \leftrightarrow \neg(p \wedge \neg q)$
 $(\forall x)(\forall y)((x=y) \vee \neg(x=y))$

MSO

Tree Decomposable Graphs

Trees

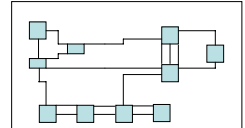
Chomsky Context-Free Languages

Fast Graph Algorithms ← **Tree Automata** → Systems Verification

(3) FORMAL LANGUAGES & Automata

Almost all interesting properties on arbitrary graphs are **NP-complete**. (e.g., k-colorable, planar, bipartite, ...)

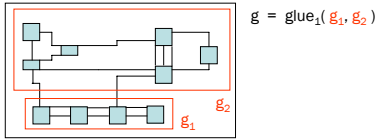
Many graphs that appear in real life are NOT arbitrary. They have a "history" (structure), i.e., they are glued together from smaller graphs.



(3) FORMAL LANGUAGES & Automata

Almost all interesting properties on arbitrary graphs are **NP-complete**. (e.g., k-colorable, planar, bipartite, ...)

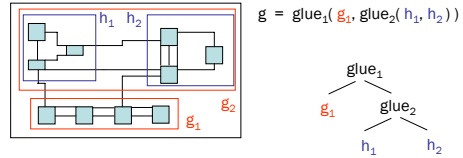
Many graphs that appear in real life are NOT arbitrary. They have a "history" (structure), i.e., they are glued together from smaller graphs.



(3) FORMAL LANGUAGES & Automata

Almost all interesting properties on arbitrary graphs are **NP-complete**. (e.g., k-colorable, planar, bipartite, ...)

Many graphs that appear in real life are NOT arbitrary. They have a "history" (structure), i.e., they are glued together from smaller graphs.



(3) FORMAL LANGUAGES & Automata

Almost all interesting properties on arbitrary graphs are **NP-complete**. (e.g., k-colorable, planar, bipartite, ...)

For graphs that have an **underlying tree structure** ("of bounded tree-width", "context-free", ...) many properties (e.g., k-colorable, planar, bipartite, connected, is-tree, ...)

can be decided in **polynomial / linear time!** → **Tree Automata**

Decidability

- Is there an *algorithm* which can check whether ϕ holds for a given s ?
- whether ϕ holds for some s ?
- whether ϕ holds for all s ?
- whether ϕ is equivalent to ψ ?
- whether ϕ is minimal?
- etc.

- Forget about graphs
- Use **tree structure**, if available!

(3) FORMAL LANGUAGES & Automata

Fix a **logic L = MSO** and a **class of structures C = STRINGS / TREES**.

→ "yes" to all of the following

Decidability

- Is there an *algorithm* which can check whether ϕ holds for a given s ?
- whether ϕ holds for some s ?
- whether ϕ holds for all s ?
- whether ϕ is equivalent to ψ ?
- whether ϕ is minimal?
- etc.

Verification does system **S** has property **P**?

Represent **S** as a **finite-state / context-free system** and **P** in **MSO logic**.

→ Automatically check if **P** holds for **S!** (using automata algorithms)

Logic and Automata. **C=STRINGS**

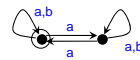


$(\forall x)(\forall y)(\exists z)(a(x) \wedge a(y) \wedge x < y) \rightarrow (x < z < y \wedge b(z))$

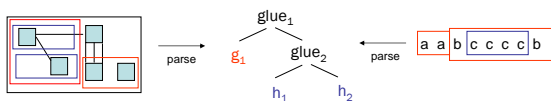
[Büchi1960]

For "string graphs" with next-edges, → power of **First-Order logic** is **VERY** limited!

E.g., *not* FO-definable: number of a's / letters is even



- use more powerful logic: **MSO**
- use more complex structures: **TREES**, graphs with tree structure



(3) MSO logic → Automata ... cave! nonelementary blowup!

Extensions of Büchi

[Büchi1960] MSO on **strings** = finite-state automata (regular languages)

[Thatcher/Wright1968] MSO on **trees** = finite-state tree automata (reg. tree lang's)

[Courcelle1988] MSO on **graphs** ⊆ recognizable graphs

(3) FORMAL LANGUAGES & Automata

Extensions of Büchi

[Büchi1960] MSO on strings =
finite-state automata (regular languages)

[Thatcher/Wright1968] MSO on trees =
finite-state tree automata (reg. tree lang's)

[Courcelle1988] MSO on graphs \subseteq recognizable graphs

Graph Transductions

→ Use one/two free variables to define node-labels and edges.

[Engelfriet/Hoogeboom2001] MSO definable strings transductions =
finite-state transducers

[BloemEngelfriet2000 & EngelfrietManeth2003] MSO definable tree transductions =
macro tree transducers of linear growth

(3) FORMAL LANGUAGES & Automata

Extensions of Büchi

[Büchi1960] MSO on strings =
finite-state automata (regular languages)

[Thatcher/Wright1968] MSO on trees =
finite-state tree automata (reg. tree lang's)

[Courcelle1988] MSO on graphs \subseteq recognizable graphs

Graph Transductions

→ Use one/two free variables to define node-labels and edges.

[Engelfriet/Hoogeboom2001] MSO definable strings transductions =
finite-state transducers

[BloemEngelfriet2000 & EngelfrietManeth2003] MSO definable tree transductions =
macro tree transducers of linear growth

→ Fresh Power: XML!! ☺

Why Tree Automata??

Regular tree languages
= BU-TA's = detBU-TAs
= TD-TA's = tree reg. exps .. [Büchi1960]

↔ MSO Logic (over trees)

↑

Parse trees of context-free grammars
(= "local" tree languages)

Why Tree Automata??

Regular tree languages
= BU-TA's = detBU-TAs
= TD-TA's = tree reg. exps ..

← MSO Logic (over trees)

↑

Parse trees of context-free grammars
(= "local" tree languages)

Decidable model Checking problem!

Why Tree Automata??

Regular tree languages
= BU-TA's = detBU-TAs
= TD-TA's = tree reg. exps ..

← MSO Logic (over trees)

↑

Parse trees of context-free grammars
(= "local" tree languages)

CTL, LTL, Mu-calc.
FO + TC
FO

Decidable model Checking problem!

Why Tree Automata??

Regular tree languages
= BU-TA's = detBU-TAs
= TD-TA's = tree reg. exps ..

← MSO Logic (over trees)

↑

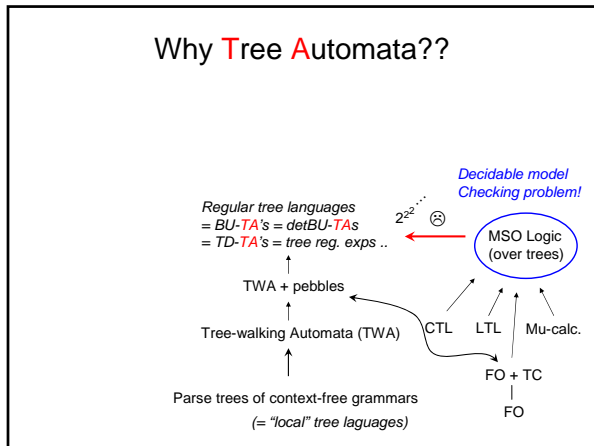
Parse trees of context-free grammars
(= "local" tree languages)

CTL, LTL, Mu-calc.
FO + TC
FO

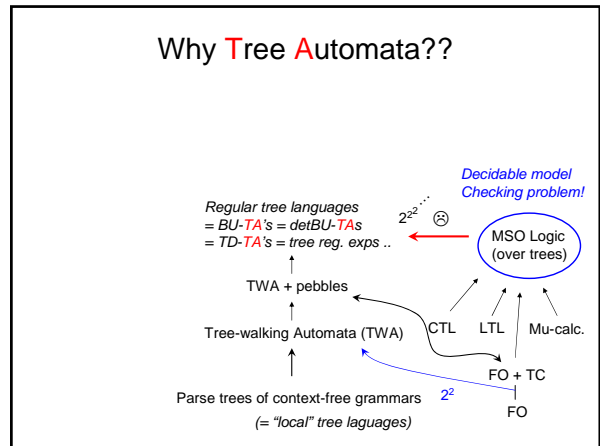
Decidable model Checking problem!

$2^{2^{\dots}}$ ☹

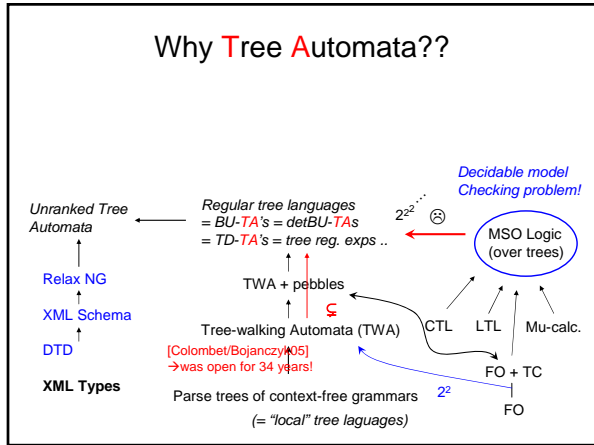
Why Tree Automata??



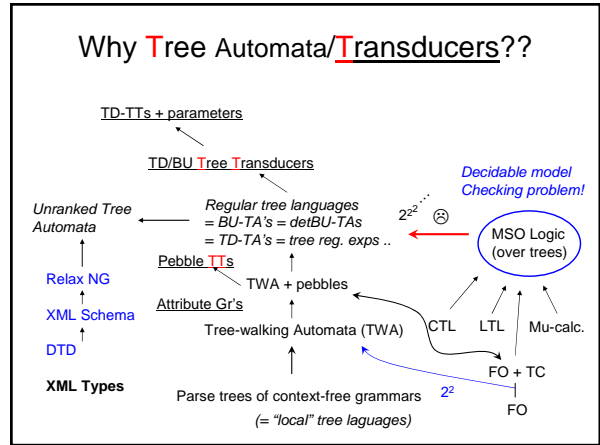
Why Tree Automata??



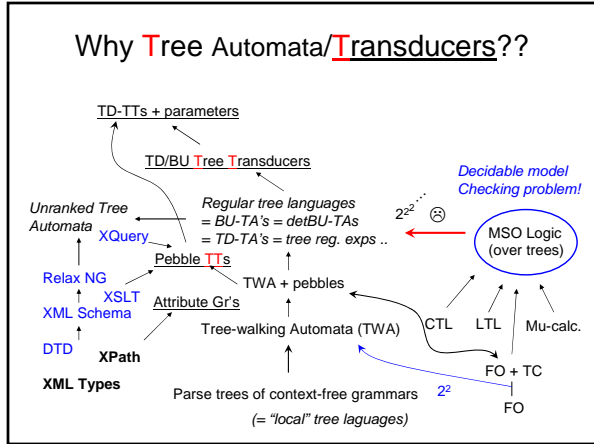
Why Tree Automata??



Why Tree Automata/Transducers??



Why Tree Automata/Transducers??



Why Tree Automata/Transducers??

