In the Maze of Data Languages

Loris D'Antoni

WPE II
05/08/2012

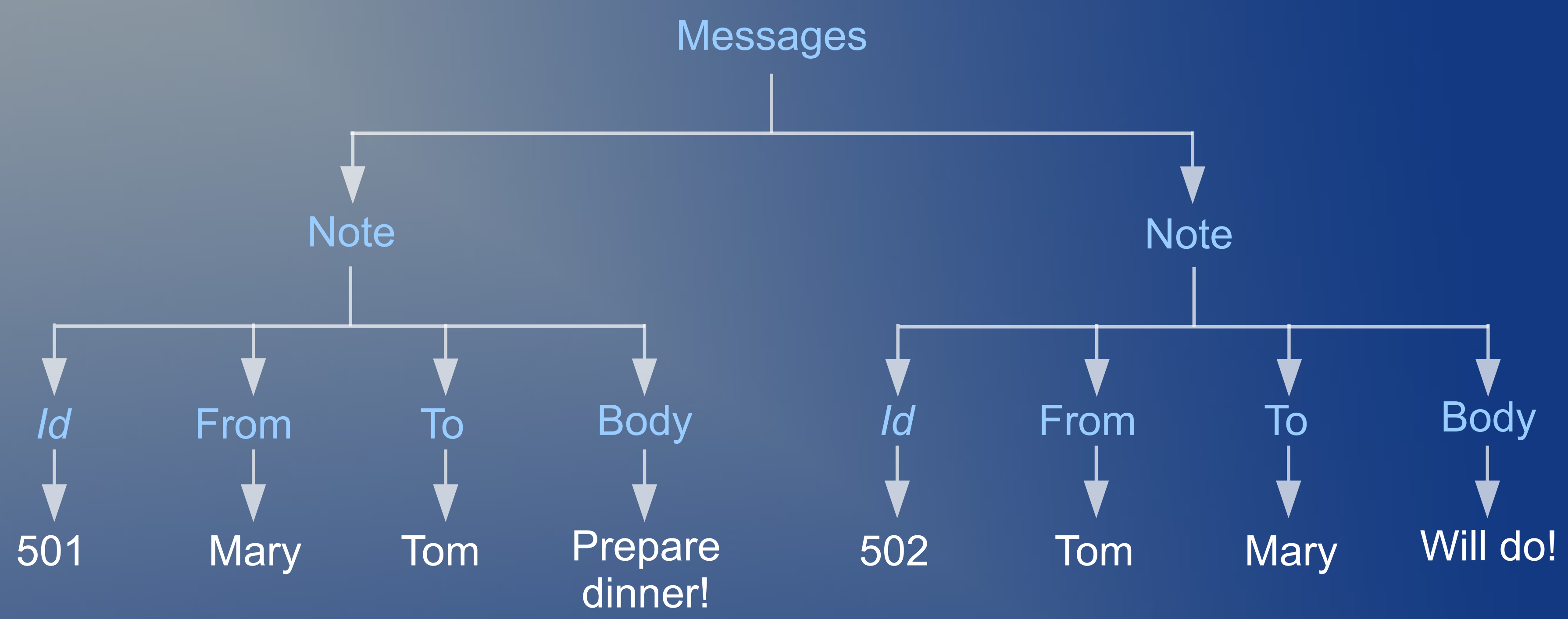
Data Languages

- ***<u>Motivation</u>***
- Data Model
- Data Strings
  - Automata and Logics
  - Regularity
- Data Trees
- Conclusion

Introduction

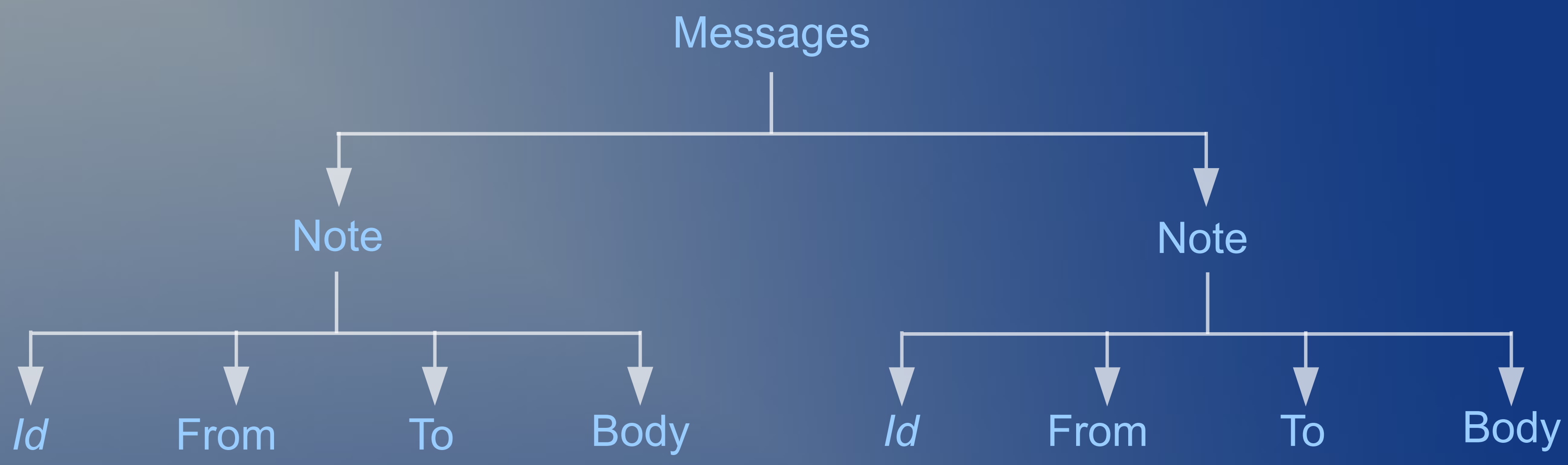
- Most analysis techniques over programs and data consider the domain to be ***finite*** in order to achieve decidability
- Often this restriction is too strong
  - XML documents and languages over XML use ***data comparison***
  - Interesting properties about programs compare **values of variables** at different points in the program

Motivation 1: XML Processing



- An XML document can be seen as an unranked tree in which
  - Inner nodes correspond to **elements** (tags)
  - Leaves correspond to **data** (attributes, text content)

Motivation 1: XML Processing



- For many useful tasks data values can be ignored
  - we can consider the **tree** to be over a **finite alphabet**
  - good for *navigation, validation, transformation*...

WHAT ABOUT TASKS IN WHICH WE WANT TO SPECIFY CONSTRAINTS OVER DATA?

Motivation 1: XML Processing

- A concrete example: ***XPath query optimization***
- **SCHEMA:** can define XML language and can also specify constraints on data
- **XPATH:** query language for XML that also allows data comparison
  - **Q1**: select all notes someone sent to himself
  - **Q2**: select people who sent more than 3 notes
- **QUERY OPTIMIZATION:** given two XPath queries *q1*,*q2* and a Schema *S*, decide whether,

for each valid document x in S, q1(x) ⊆ q2(x)

Motivation 2: Verification

- **Model Checking:** checking properties about programs that can have possibly infinite reachable states
  - Represent system as a finite structure
  - Define a transition relation
  - Use algorithm for reachability of some particular state
- Several **ad-hoc solutions** for particular cases of infinite alphabets and infinite states
  - Timed Automata [Alur90]
  - Regular model checking [Bouajjani00]

Motivation 2: Verification

- No model considers inter-state properties such as

  the same resource is never granted twice (with infinitely many resources)

- A run of the transition system can be seen as a string/list of the form

q0	r1	→	q1	r4	→	q3	r1	→	….	→	qf	r1

  where the states are from a finite alphabet and the resources are from an infinite domain and

- Now we can ask:

  Is there a list with the same resource appearing twice

Some Models for Infinite

- Several models have been proposed to work with **infinite alphabets**:
  - LTL with Freeze Quantifiers (LTL with storing registers)
  - Timed Automata (can reason about Time)
  - Symbolic Automata and Transducers (theory over input)
- Most of these models are quite domain specific even though they come with nice properties
- We want a ***general theory*** for

structures over infinite alphabets

Data Languages

- Motivation
- ***Data Model***
- Data Strings
  - Automata and Logics
  - Regularity
- Data Trees
- Conclusion

Data Model: Design Principles

- We need a simple model with some decidable features
- The model should be useful
- Possibly it should be guided by some practical applications

DATA STRINGS and DATA TREES

Data Languages

- We take languages of words and trees over finite alphabets
- Then, **one data element** from an **infinite domain** is allowed for every position/node
- The **only operation** that can be performed over data is checking for **equality**
- It is a bit restrictive but **easy to study** and **useful** in practice
- Moreover, most extensions immediately lead to undecidability

Data Strings

- In a data string each position carries
  - a **label** from a *finite alphabet* and
  - a **data value** from an *infinite alphabet*

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

- For example in the data string above
  - the finite alphabet is {r,w,s}
  - the infinite alphabet is the natural numbers

Data Trees

- Similarly, in a data tree each NODE carries
  - a **label** from a *finite alphabet* and
  - a **data value** from an *infinite alphabet*

And now?

- Data languages seem to nicely extend regular languages
- The framework is set but now:
  - How do we define data languages?
  - What is the **best/right** model for data string languages?
  - What is the **best/right** model for data tree languages?
  - What is a **regular** data language?

Regularity

- Ideally we are looking for a model for data languages with all the nice properties of regular string languages
  - Good tradeoff between **expressiveness** and **decidability**
  - **Efficiency** of the membership problem
  - Good **closure properties**
  - **Robustness:** clear counter part in logic and several characterizations

DOES A MODEL LIKE THAT EVEN EXIST?

Data Languages

- Motivation
- Data Model
- ***Data Strings***
  - ***Automata and Logics***
  - Regularity
- Data Trees
- Conclusion

Models for Data Strings

- Several models have been proposed for data strings and they are mainly of two kinds:
  - **Auotomata** based models
  - **Logic** based models
- Usually an automata model is good when it has an equivalent logic model
- Here we present the models that are considered more relevant in the `treasure hunt' for regular data string languages

Register Automata 1/4

- Finite state automaton + **finite set of registers** that can store data values and test for equality

STATE q

R1=4, R2=1

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

(q,R1,r) steps to (q',L)

STATE **q'**

R1=4, R2=1

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 2/4

STATE q

R1=4, R2=1

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

if **no R** contains current value **(q,r) steps to (q',R2,R)**

STATE **q'**

R1=4, R2=**5**

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - (q0,{r,w,s}) → (q1,R1,R)
  - (q1,R1,{r,w,s})   (qf,S)
  - (q1,{r,w,s})   (q1,R1,R)

STATE q0

R1=

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - **(q0,{r,w,s}) → (q1,R1,R)**
  - (q1,R1,{r,w,s}) (qf,S)
  - (q1,{r,w,s}) (q1,R1,R)

STATE q0

R1=

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - (q0,{r,w,s}) → (q1,R1,R)
  - (q1,R1,{r,w,s})   (qf,S)
  - (q1,{r,w,s})   (q1,R1,R)

STATE **q1**

R1=**1**

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - (q0,{r,w,s}) → (q1,R1,R)
  - (q1,R1,{r,w,s})   (qf,S)
  - **(q1,{r,w,s})   (q1,R1,R)**

STATE q1

R1=1

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - (q0,{r,w,s}) → (q1,R1,R)
  - (q1,R1,{r,w,s})   (qf,S)
  - (q1,{r,w,s})   (q1,R1,R)

STATE **q1**

R1=**4**

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - (q0,{r,w,s}) → (q1,R1,R)
  - (q1,R1,{r,w,s})   (qf,S)
  - **(q1,{r,w,s})   (q1,R1,R)**

STATE q1

R1=4

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - (q0,{r,w,s}) → (q1,R1,R)
  - (q1,R1,{r,w,s})   (qf,S)
  - (q1,{r,w,s})   (q1,R1,R)

STATE **q1**

R1=**1**

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - (q0,{r,w,s}) → (q1,R1,R)
  - **(q1,R1,{r,w,s})   (qf,S)**
  - (q1,{r,w,s})   (q1,R1,R)

STATE q1

R1=1

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

Register Automata 3/4

- Language of data strings were two adjacent positions contain the same data value
  - (q0,{r,w,s}) → (q1,R1,R)
  - (q1,R1,{r,w,s})   (qf,S)
  - (q1,{r,w,s})   (q1,R1,R)

STATE **qf**

R1=1

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

DATA STRING ACCEPTED

Register Automata 4/4

- The **Register Automata (RA)** in the previous example is called one-way determistic
- Based on the restrictions we impose RAs can be
  - Deterministic, Nondeterministic or Alternating
  - One way or Two way
- Each of the above choices affects the expressiveness and the decidability of the model

What do we have so far?

	Emptiness	Universality	Inclusion	Equivalence
1Way–ND Register Automata	**Decidable**	Undecidable	Undecidable	Undecidable
2Way–D Register Automata	Undecidable	Undecidable	Undecidable	Undecidable

Maybe we should slow down a little bit...

Let's put PAs aside and try to improve **1Way-ND RAs**

Generalizing a Little Bit...

- **1Way-ND-RAs** are limited in expressiveness: they can't even represent the language of strings where all the data values are different!!
- We need a slightly **more expressive** model...but not too expressive otherwise we hit undecidability
- **Weakness of RAs**: they can only talk about global properties but not about local properties:
  - **GLOBAL**: property of the whole string
  - **LOCAL**: all label of position with same data have some property

Class Memory Automata 1/3

- **More expressive** than **Register Automata** :)
- **Single pass** and **one way**!! :(
- **Non-deterministic** :(
- At every point transitions depend on *class history*
- Let's see an example...

Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. **For every q,**
  - **(q,{r,w,s},-) → q1**
  - **(q,{r,w,s},q1) → q2**
  - **(q,{r,w,s},q2) → q2**
- Global acceptance {q0,q1,q2}, local acceptance {q1}

1	-
4	-
34	-
5	-

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

q0

Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - **(q,{r,w,s},-) → q1**
  - (q,{r,w,s},q1) → q2
  - (q,{r,w,s},q2) → q2
- Global acceptance {q0,q1,q2}, local acceptance {q1}

1	-
4	-
34	-
5	-

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

q0

Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - (q,{r,w,s},-) → q1
  - (q,{r,w,s},q1) → q2
  - (q,{r,w,s},q2) → q2
- Global acceptance {q0,q1,q2}, local acceptance {q1}

1	**q1**
4	-
34	-
5	-

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

q1

Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - **(q,{r,w,s},-) → q1**
  - (q,{r,w,s},q1) → q2
  - (q,{r,w,s},q2) → q2
- Global acceptance {q0,q1,q2}, local acceptance {q1}

1	q1
4	-
34	-
5	-

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

q1

Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - (q,{r,w,s},-) → q1
  - (q,{r,w,s},q1) → q2
  - (q,{r,w,s},q2) → q2
- Global acceptance {q0,q1,q2}, local acceptance {q1}

1	q1
4	**q1**
34	-
5	-

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

q1

Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - (q,{r,w,s},-) → q1
  - **(q,{r,w,s},q1) → q2**
  - (q,{r,w,s},q2) → q2
- Global acceptance {q0,q1,q2}, local acceptance {q1}

1	q1
4	q1
34	-
5	-

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

q1

Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - (q,{r,w,s},-) → q1
  - (q,{r,w,s},q1) → q2
  - (q,{r,w,s},q2) → q2
- Global acceptance {q0,q1,q2}, local acceptance {q1}

1	q1
4	**q2**
34	-
5	-

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

q2

Class Memory Automata 2/3

- Following transitions to implement the same language as before where all data values are different. For every q,
  - (q,{r,w,s},-) → q1
  - (q,{r,w,s},q1) → q2
  - (q,{r,w,s},q2) → q2

DATA STRING REJECTED

- Global acceptance {q0,q1,q2}, local acceptance {q1}

1	q2
4	q2
34	q1
5	q2

r	w	w	r	s	r	r	s	r	w
1	4	1	1	4	34	4	5	5	4

q2

Class Memory Automata 3/3

- This example can't be computed by a 1WAY-ND-RA and in general
  - **1WAY-ND-RA** are **strictly included** in **CMA**s for what concern expressiveness
- **Emptiness** is still **decidable** for **CMAs**!!!
- **Membership** was easy now is **NP-Complete** :(

Why is this model right?

- Class memory automata are **usable** :)
- More **expressive** than 1Way-ND-Register Automata :)
- Still **decidable** emptiness :)
- But who tells us there is no better model?? :(
- Regular string languages are equivalent to MSO
  - Models equivalent to a logic are more robust,
  - They have closure properties for free

- We need a **declarative model equivalent** to CMA...

Data Languages

- Motivation
- Data Model
- ***Data Strings***
  - Automata and ***Logics***
  - Regularity
- Data Trees
- Conclusion

First-Order Logic for Data Strings

- **Variables** range over **positions** of the data string
- **Formulae** of the form

$$\Phi ::= a(x) \mid x=y \mid \exists x.\Phi \mid \Phi \vee \Phi \mid \neg\Phi \mid$$

$$\mathbf{x=y+1 \mid x<y \mid x\sim y}$$

- We call this logic **FO(+1,<,~)**
- Here is a simple formula for the language of data strings with all different data values

$$\neg\exists x.\exists y.\neg(x=y) \wedge x\sim y$$

Monadic Second-Order Logic for Data Strings

- **Variables** range over **positions** of the data string
- **Second-order variables** range over **sets of positions**
- **Formulae** of the form

$$\Phi ::= a(x) \mid x=y \mid \exists x.\Phi \mid \Phi \vee \Phi \mid \neg\Phi \mid$$

$$x<y \mid x=y+1 \mid x\sim y \mid \exists X.\Phi \mid x \in X$$

- We call this logic **MSO(+1,<,~)**
- Here is a formula for the language of data strings with at least one position with label **a**

$$\exists X.\ \exists x.\ x \in X \wedge a(x)$$

Bad news...

- **Emptiness:** Given a **formula Φ** in FO(+1,<,~) or MSO(+1,<,~) checking whether there **exists** a data string **s** that is a **model of Φ** is **undecidable**
- Too much expressiveness, we need something **weaker**
- Maybe we can consider a logic where the **number of variables** that can be used is **limited**...

Two-Variable Logics 1/3

- We call **FO2** first-order logic with only two variables **x,y**
- In general adding variables adds expressiveness
  - **FOK(+1,<,~)** is
    less expressive than **FO(K+1)(+1,<,~)**
- And...
  - **Emptiness** for a formula in **FO2(...)** is **decidable**
  - **Emptiness** for a formula in **FO3(...)** is **undecidable**
- This is a nice cut-off!!

Two-Variable Logics 2/3

- We can actually do a bit better
- We call **EMSO2(+1,<,~)**, monadic second order logic where all the **set variables X** are **existentially** quantified and they appear at the **beginning** of a formula
  - **∃X1...∃Xn.Φ** and **Φ** only contains **FO2(...)**
- And..
  - **Emptiness** for a formula in **EMSO2(...)** is **decidable**
  - **EMSO2(...)** is strictly more expressive than **FO2(...)**

Two-Variable Logics 3/3

- We can push the decidability even a bit further
- Consider the operator **$\oplus$1** such that:
  - **x=y$\oplus$1** iff **y** is the **next position** after **x** **with the same data value** of **x**
- We call **EMSO2(+1,<,~,$\oplus$1)**, **EMSO2(+1,<,~)** enriched with the **$\oplus$1** operator
- **Emptiness** for a formula in **EMSO2(+1,<,~,$\oplus$1)** is **decidable**
- **EMSO2(+1,<,~,$\oplus$1)** is strictly **more expressive** than **EMSO2(+1,<,~)**

Grand Finale

- Incredible but true

CLASS MEMORY AUTOMATA

$EMSO2(+1,<,\sim,\oplus 1)$

all have the **same expressiveness**

Data Languages

- Motivation
- Data Model
- ***Data Strings***
  - Automata and Logics
  - ***Regularity***
- Data Trees
- Conclusion

Regularity 1/2

- It seems that class memory automata are a good model but it still not **regular** in the standard sense

	Regular String Languages	Class Memory Automata
Equivalence	**Decidable**	**Undecidable**
Emptiness	**Decidable**	**Decidable**
Deterministic	**Yes**	**No**
Closure under intersection, union, star	**Yes**	**Yes**
Closure under complement	**Yes**	**No**
Logical counter part	**Yes**	**Yes**
Membership	**Linear**	**NP-Complete**

Regularity 2/2

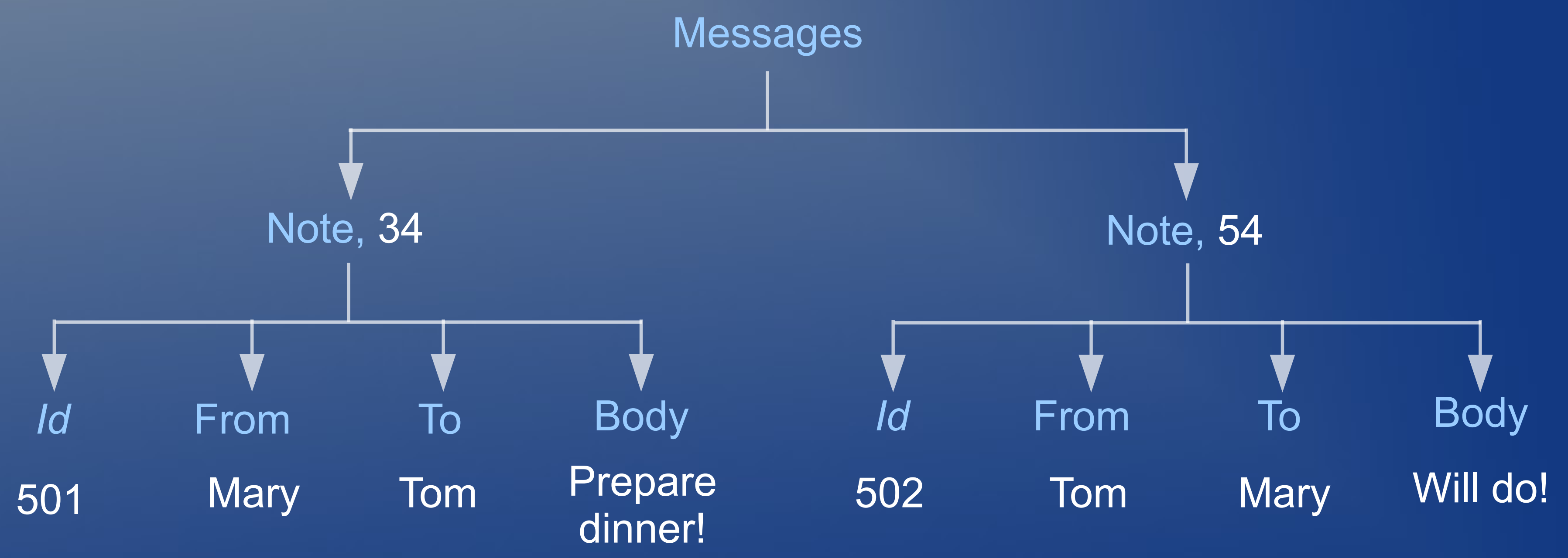
- Unfortunately this is the best we have so far... so:
  - We can look for a **less expressive** model maybe **equivalent to** some form of **FO** so that we have closure under complement
  - We accept that Data Languages are **hard to work with** and we give up on some properties
- In any case, what defines a **regular string data language** is still and **open problem**

Data Languages

- Motivation
- Data Model
- Data Strings
  - Automata and Logics
  - Regularity
- ***Data Trees***
- Conclusion

Data Trees

- Similarly, in a data tree each NODE carries
  - a **label** from a *finite alphabet* and
  - a **data value** from an *infinite alphabet*

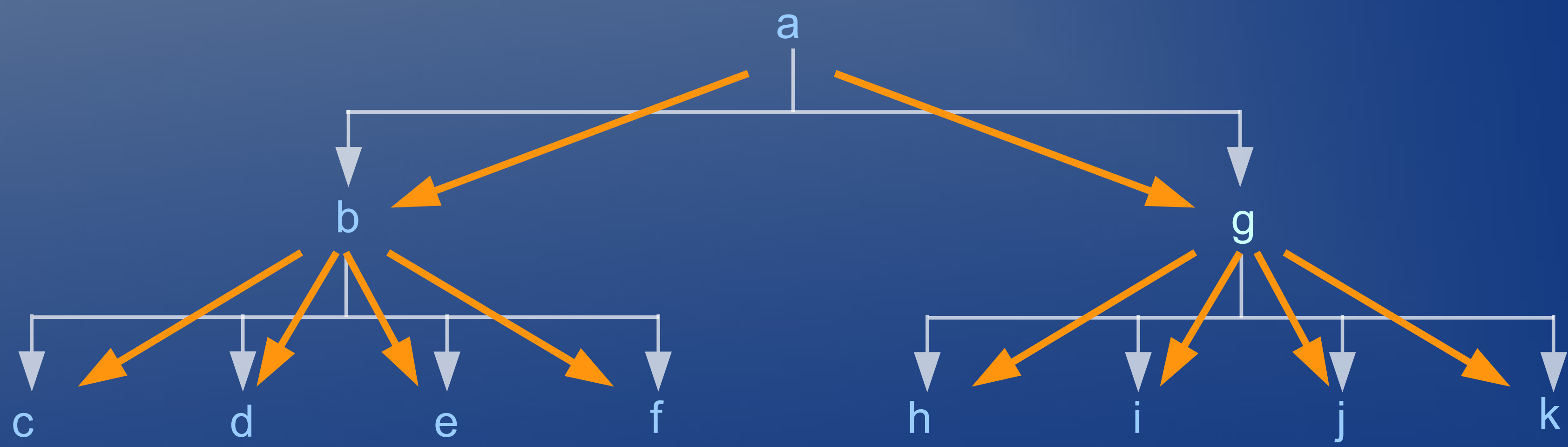


Models for Data Trees

- The **state of the art** for automata and logics over data trees is **embryonic**
- Very few studied models with decidable properties
  - Some extensions of **register automata**, but without intuitive semantics and limited expressiveness
  - Extensions of **FO2(+1,<,~), EMSO2(+1,<,~)** to data trees
- Logic is more relevant for regularity, so let's concentrate on that

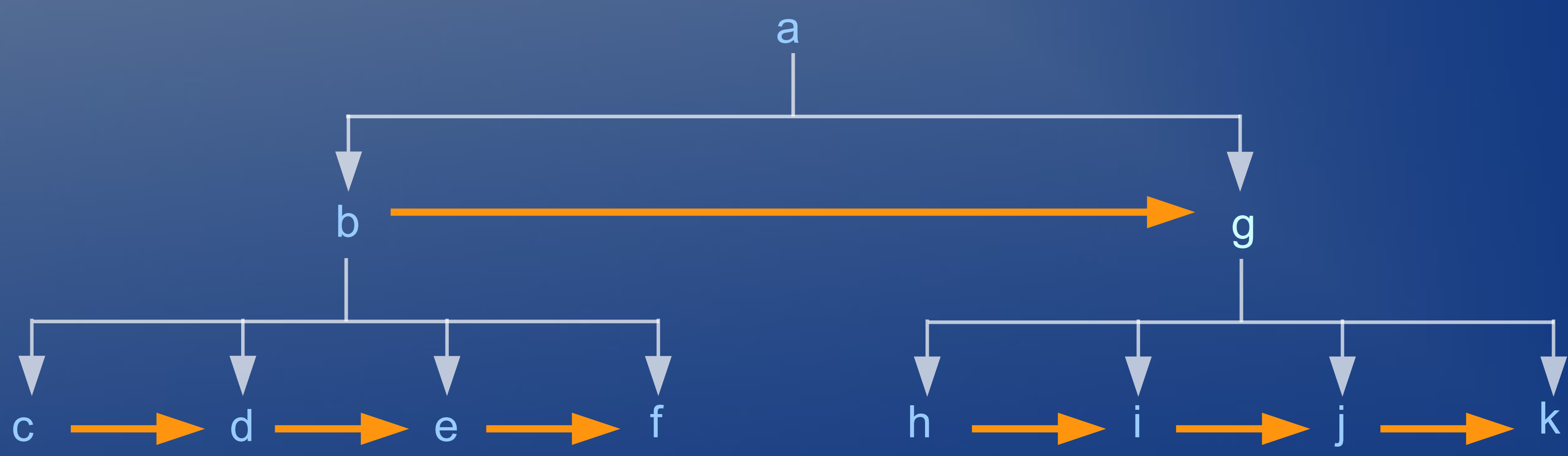
Two-Variable Logics 1/3

- In a tree the predicates +1 and < don't have a clear interpretation...
- We replace them with **two** new predicates
  - **$x = y \downarrow 1$** parent relation and



Two-Variable Logics 1/3

- In a tree the predicates +1 and < don't have a clear interpretation...
- We replace them with **two** new predicates
  - $x = y \downarrow 1$ parent relation and
  - **$x = y \rightarrow 1$** next sibling relation



Two-Variable Logics 2/3

- Now the **+1** predicate corresponds to ↓ and →
- The **<** predicate corresponds to the **transitive closures** of ↓ and →, ↓* and →*
- We now have an interpretation for
  - **FO2(+1,<,~)** and **EMSO2(+1,<,~)** over data trees
- But we can also consider simpler logics where we **drop** the **<** operator
  - **FO2(+1,~)** and **EMSO2(+1,~)** over data trees
- But why would we do that??

Two-Variable Logics 3/3

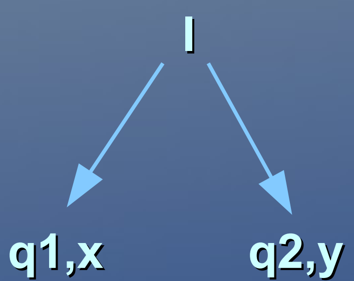
- **Emptiness** for
  - **FO2(+1,~)** and **EMSO2(+1,~)** over data trees

  is **decidable**
  - We will see a proof sketch
- **Emptiness** for
  - **FO2(+1,<,~)** and **EMSO2(+1,<,~)** over data trees

  is a very **hard open problem**
  - **Emptiness** for **vector addition tree automata** reduces to it...

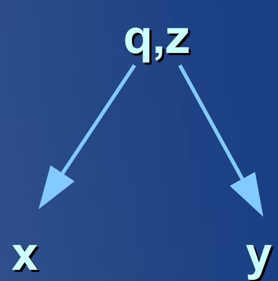
Vector Addition Tree Automata

- **Bottom up** tree automata over binary trees in which transitions have three vectors
  - Every leaf is assigned a vector of values over Nat
  - Transitions of the form **q1,q2,a,b,c,l → q**



If x-a>0 and y-b>0

z=(x-a)+(y-b)+c



- Is there a run in which **root** is labeled with the **vector 0**
- **Very hard open problem** reduces to **emptiness** for **FO2(+1,<,~)**

Emptiness for FO(+1,~) is Decidable 1/4

- Proof outline, given a formula **F** in **FO(+1,~)**
  - Compute a **``puzzle'' P** that has **solutions** iff **F** is **satisfiable**
  - If a **``puzzle'' P** has a **solution**, then there also exists a **``small'' solution**
  - **Find** if there exists a **``small'' solution** of **P** using some **extended tree automata**

Emptiness for FO(+1,~) is Decidable 2/4

- Compute a ``**puzzle" P** that **has solutions** iff **F** is **satisfiable**
  - Reduce **F** to a **normal form F'**
  - For every formula **F'** we can **co**
- A **puzzle** over Σ is a pair **(L,F)**
  - **L** is a **tree automata** over an exten
  - **F** is a **set of accepting pairs (D,S)** where D,S are disjoint subsets of Σ
  - **Solution:** a data tree ``in" **L** where for each **class:** there exists pair **(D,S)** such that **all labels** are **from D U S** and each **label in D** appears **at most once**

Class:
Maximal set of connected nodes with the same data value

Emptiness for FO(+1,~) is Decidable 3/4

- If a **``puzzle'' P** **has a solution**, then there a **``small'' solution**
  - Let's assume there is a **solutio**

Sibilinghood: Set of children of a node

- A **(M,N)-reduced solution** is a solutio
  - At most **M classes** are of size **greater** than **N**
  - There are at most **M sibilinghoods** with more than **N classes**
- Given a solution we can **compute** a **(M,N)-reduced solution** and
  - **M** and **N** are effectively **computable** numbers from the size of the puzzle

Emptiness for FO(+1,~) is Decidable 4/4

- **Find** if there exists a **``small'' solution** of **P** using some **extended tree automata**
- Given a tree automata **A** and a tree **t**, a **run** of **A** can be seen as a **la**[…] **Q** of **A**
  - **Linea**[…]**CTA**) are e[…] over **Q**
  - A ru[…] instantiated w[…]e run
  - **Emptines**[…] of LC[…] **ecidable**
- We can compute a **LCTA** that is **empty** iff the puzzle **P** **does not** have (M,N)-reduced **solutions**

QED

Regularity

- Talking about **regularity** for **data tree languages doesn't** make quite sense
- **Only one** very limited **logic** with decidable emptiness
- No **automata** equivalent model
- No proof of undecidability for more expressive logic
- **Very little research so far**

Conclusion

- Motivation
- Data Model
- Data Strings
  - Automata and Logics
  - Regularity
- Data Trees
- ***Conclusion***

We have seen...

- First proposals for automata models for string data languages **(RA)**
- Improved automata models with equivalent logical counterpart **(CMA)**
- Most relevant logics for data languages **(FO2,EMSO2)**
- Basic (only) result on decidability of **FO2** for **data trees**
- Some discussion on what **regularity** might be for **data languages** (no hope for full package)

Open Problems

- Is there a variant of **RA** closed under **complementation**?
- Is there a variant of **FO2** **equivalent** to **1WAY-ND-RA**?
- Is there a **logic** closed under **complementation** with **decidable equivalence**?
- Can we **extend** current **data strings models** to **data trees** preserving good properties?
- Is **FO2(+1,<,~)** **decidable** for **data trees**?

References

- Frank Neven... Finite state machines for strings over innite alphabets. 2004
- Mikolaj Bojanczyk... Two-variable logic on words with data. 2006
- Mikolaj Bojanczyk...Two-variable logic on data trees and xml reasoning. 2006
- Henrik Bjrklund... On notions of regularity for data languages. 2007
- …..

Thank you... Questions?