Hamming Seminar

Leonid Libkin

24 October 2012

The traditional part

- What Hamming said in his Bellcore talk:
  - What are the important problems of your field?
  - Are you working on them? If not, why?
  - Do you have a way of approaching them?
  - What is the time-frame for solutions to emerge?

The traditional part

- What Hamming said in his Bellcore talk:
  - What are the important problems of your field?
  - Are you working on them? If not, why?
  - Do you have a way of approaching them?
  - What is the time-frame for solutions to emerge?
- My fields:
  - Data management – relational, XML, and graph databases, query languages, power and complexity, constraints and design, integration and exchange of data, incomplete information
  - Logic in CS (it has been called "the calculus of computer science) – finite model theory, automata theory, applications in databases, complexity, verification

Logic in CS problems

- We use logic mainly as a tool
- So many famous problems can be seen as pure logic problems
- Example:
  - Is existential second-order logic strictly more expressive than least fixed-point logic over finite ordered graphs?

Logic in CS problems

- We use logic mainly as a tool
- So many famous problems can be seen as pure logic problems
- Example:
  - Is existential second-order logic strictly more expressive than least fixed-point logic over finite ordered graphs?
- Also known as the P vs NP problem
- Crucial notion: ordering
  - addresses the mismatch between machine models and logical representations
- No chance to handle it for very expressive logics
- So we are crawling before learning how to fly: looking at weaker logics to get insights

Data management problems: Data models

- Early history
  - Network
  - Hierarchical
  - Relational
- 30 years of relational paradise, then history reverses itself:
  - The hierarchical model strikes back: XML
  - The network model strikes back: Graph Databases
- Applications of graph databases: social networks, RDF and the Semantic Web, lots of others
- Databases tend to be extremely large
- Queries mix handling data and topology, cannot be handled well by relational DBMSs

Data management problems: Incomplete Data

- It is ubiquitous in modern applications
- Arises every time data is moved between applications
  - data integration and exchange
- The most poorly developed part of database theory and practice
- Every database practitioner/seasoned SQL programmer thinks the following statements are consistent

$$|X| > |Y| \text{ and } X - Y = \emptyset$$

- What to do?
- Several attempts to create proper models in the 1980s and the early 1990s: inadequate for what we are dealing with today

Data management problems: Query answering

- What does it mean to answer a query?
- In huge databases, getting the answer may be prohibitively high
- Solution: try to approximate
- Two approaches: look at the query only, or both query and data
- Very different techniques

Data management problems: Query answering

- What does it mean to answer a query?
- In huge databases, getting the answer may be prohibitively high
- Solution: try to approximate
- Two approaches: look at the query only, or both query and data
- Very different techniques
- It all fits nicely into EPSRC challenge address the deluge of data and deliver understanding from information
- And we are working on it

Data management problems: Query answering

- What does it mean to answer a query?
- In huge databases, getting the answer may be prohibitively high
- Solution: try to approximate
- Two approaches: look at the query only, or both query and data
- Very different techniques
- It all fits nicely into EPSRC challenge address the deluge of data and deliver understanding from information
- And we are working on it
- But today I want to talk about something that, by definition, has nothing to do with £££$$$

Real Hamming Seminar

What Hamming didn't talk about:

Is there a place for beauty in our research?

Transcript of Hamming's talk

- Never talks about beauty.
- Why is this? Is it not important?
- We strive to find beautiful solutions.
- But are they necessary to solve problems?
- Do they please only their creators/inventors/discoverers?

What types of beauty are we talking about?

- Beautiful definition
- Beautiful idea/concept
- Beautiful proof

What types of beauty are we talking about?

- Beautiful definition
- Beautiful idea/concept
- Beautiful proof

- Do we really care?
  - Who are we? Researchers? Users?
  - And what is beauty in the first place?

What is beauty?

It is universal or individual?

What is beauty?

It is universal or individual?

Sometimes it seems to be universal

What is beauty?

It is universal or individual?

Sometimes it seems to be universal

This is beautiful:

What is beauty?

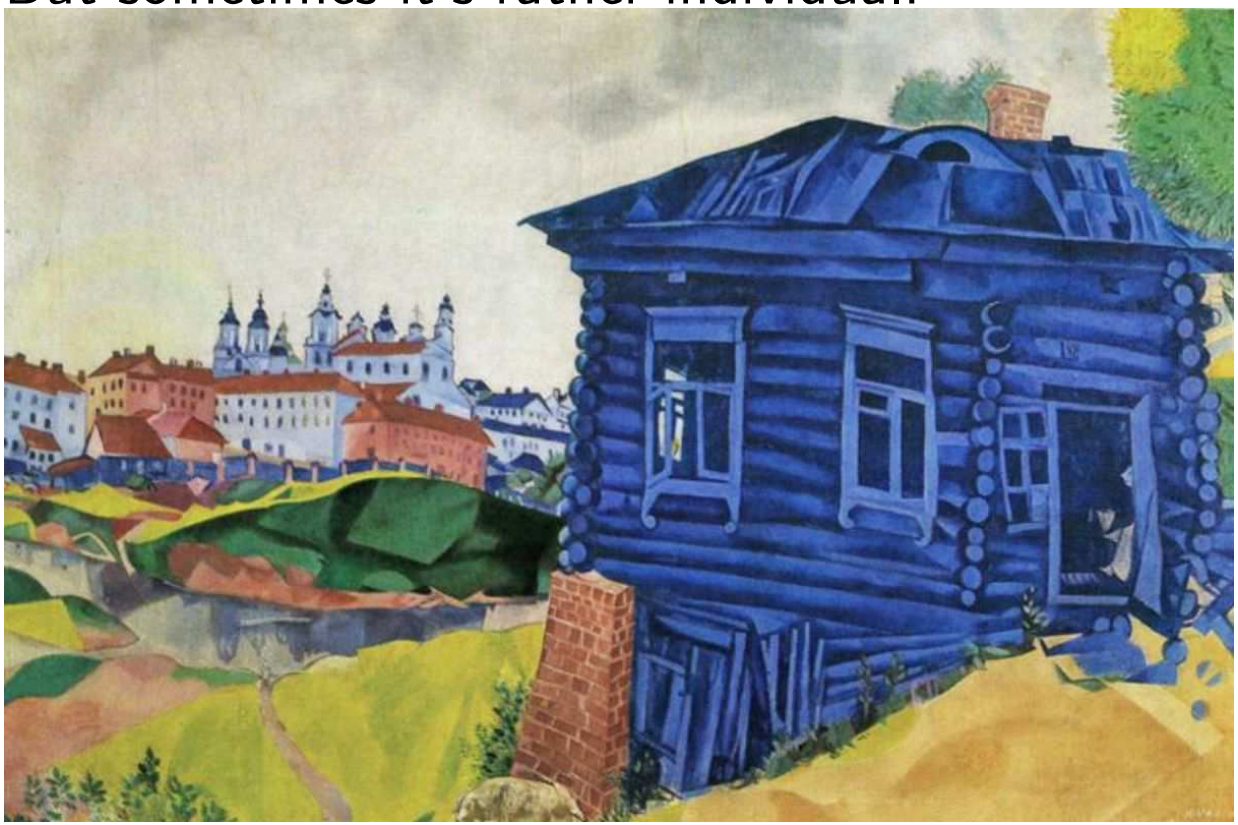
It is universal or individual?

Sometimes it seems to be universal
This certainly isn't:

What is beauty?

It is universal or individual?

But sometimes it's rather individual:



What is beauty?

It is universal or individual?

- Plato's view: universal
- Hume's view: individual
- We are in Scotland, so we ought to subscribe to Hume's view!

Getting closer to science

The most distinct and beautiful statements of any truth must take at least the mathematical form.

Henry Thoreau (1873)

Getting closer to science

The most distinct and beautiful statements of any truth must take at least the mathematical form.

Henry Thoreau (1873)

The mathematician's patterns, like the painter's or the poet's must be beautiful; the ideas, like the colours or the words must fit together in a harmonious way. Beauty is the first test: there is no permanent place in this world for ugly mathematics.

G H Hardy (1941)

Why do we need it?

- Beautiful definition: crystallize the concept
- Beautiful idea:
  - make it manageable
  - make it attractive to people
- Beautiful proof:
  - opens up new directions
  - sometimes simplicity leads to practical benefits

First example: relational databases

- Early days: messy models — network, hierarchical
  - hard to represent data, hard to query without knowing how it is organized
- Codd 1969: relational model. A beautiful concept:
  - separates logical and physical structure
  - logical structure: a fundamental mathematic concept – relations
  - querying language: first-order logic (FO)

First example: relational databases

- Early days: messy models — network, hierarchical
  - hard to represent data, hard to query without knowing how it is organized
- Codd 1969: relational model. A beautiful concept:
  - separates logical and physical structure
  - logical structure: a fundamental mathematic concept – relations
  - querying language: first-order logic (FO)
- The rest is history. It's a $\$20 \cdot 10^9$/year business now.

First example: relational databases

- Early days: messy models — network, hierarchical
  - hard to represent data, hard to query without knowing how it is organized
- Codd 1969: relational model. A beautiful concept:
  - separates logical and physical structure
  - logical structure: a fundamental mathematic concept – relations
  - querying language: first-order logic (FO)
- The rest is history. It's a $\$20 \cdot 10^9$/year business now.
- No one has done more for the employment of logicians

After 30 years of paradise

- Return to the hierarchical model: XML
- But this time with a nice definition
  - Labeled unranked trees capture structure of documents
  - Data trees capture the abstraction of XML documents (including both structure and data)
- Lessons learned from the relational world:
  - Need a clean mathematical abstraction to work with
  - Need to separate declarative and procedural languages
  - It all needs to be built on top of a solid theory (in this case, formal languages and automata)

Meet the new data model

- Same as the old data model
- Now we have returned to the original network model
- We call it graph databases
- A clean definition was already worked out by Alberto Mendelzon and colleagues in the late 1980s
  - Inspired by structures seen in verification (labeled transition systems), but with different languages
- It had to wait 20 years to find its applications
  - social networks, RDF, others

Writing program specifications

- The story is somewhat similar to that of relational databases
- Pnueli finds a nice clean formalism based on what had earlier been done by logicians studying the logic of time
- The formalism turns out be:
  - very nice and clean
  - suitable for writing specifications
  - for instance: the program will be never be in a bad state

$$\mathbf{G}\neg\text{bad}$$

  - every request is eventually granted

$$\mathbf{G}(\text{request} \rightarrow \mathbf{F}\ \text{granted})$$

- Also like Codd's discovery, led to a Turing award

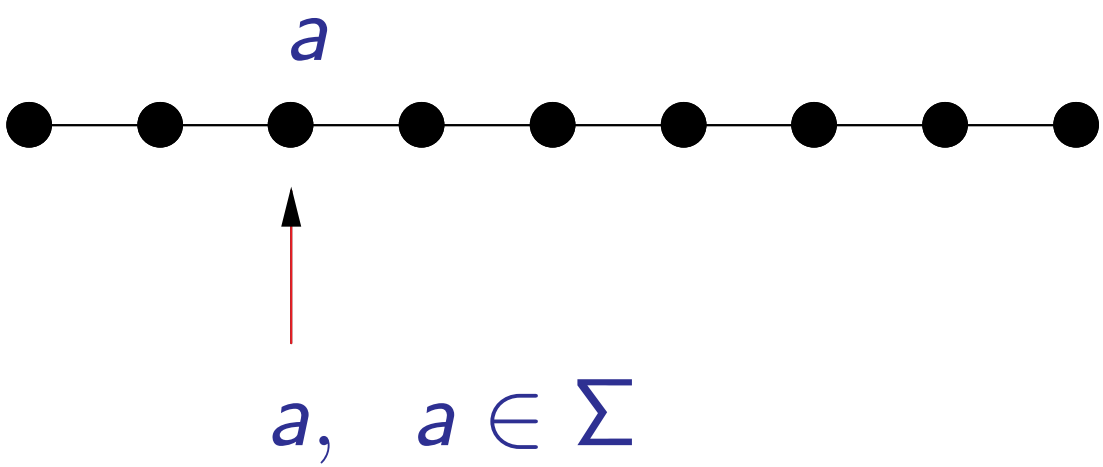
Writing program specifications: LTL

Syntax: $\varphi := a \ (\in \Sigma) \mid \varphi \vee \varphi' \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi'$

Writing program specifications: LTL

Syntax: $\varphi := a \ (\in \Sigma) \mid \varphi \vee \varphi' \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi'$

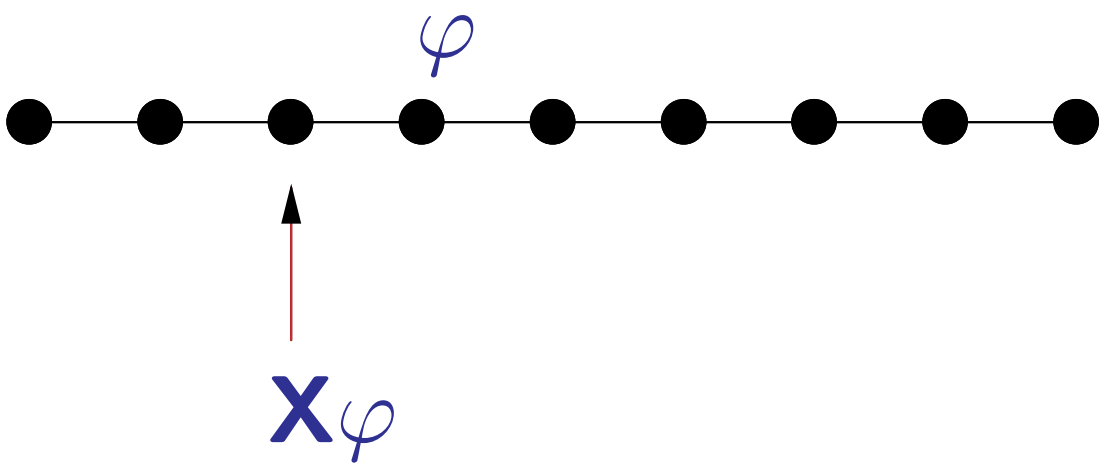
Semantics:



Writing program specifications: LTL

Syntax: $\varphi := a \ (\in \Sigma) \mid \varphi \vee \varphi' \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi'$

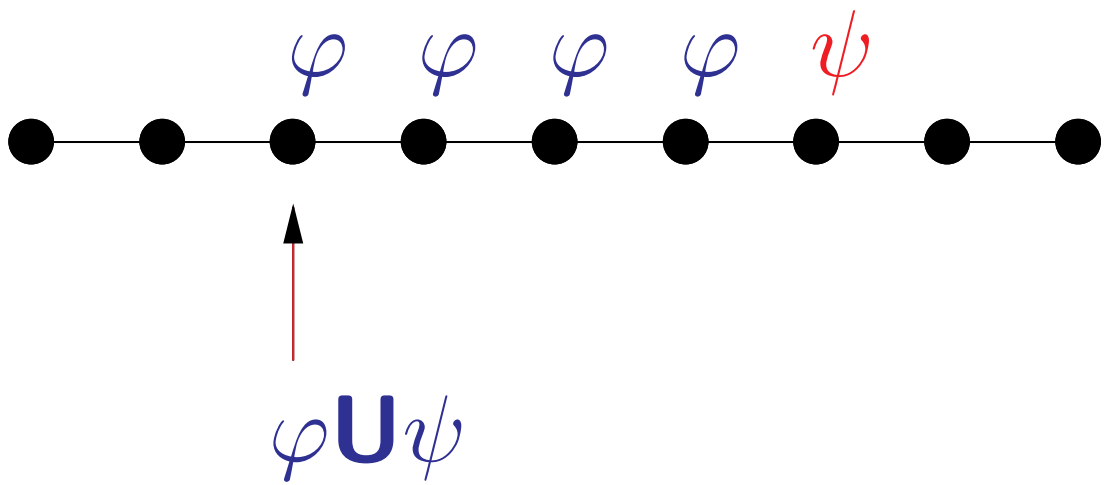
Semantics:



Writing program specifications: LTL

Syntax: $\varphi := a \ (\in \Sigma) \mid \varphi \vee \varphi' \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi'$

Semantics:



Abbreviations:

- $\mathbf{F}\varphi$ (or φ is true in the future) $= \text{true}\mathbf{U}\varphi$
- $\mathbf{G}\varphi$ (or φ is true always/globally) $= \neg\mathbf{F}\neg\varphi$

More complex specifications

- For LTL, executions are linear
- What if they can branch? Each state can be the beginning of many execution paths.
- Another nice and clean idea (Clarke/Emerson): CTL*

$$\begin{array}{lrcl} \text{(state formulae)} & \varphi & := & a \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathbf{E}\psi \\ \text{(path formulae)} & \psi & := & \text{LTL over state formulae} \end{array}$$

- $\mathbf{E}\varphi$: there is a path starting in this state on which φ is true
- Lots of fragments heavily used in verification
- Another Turing award

Beautiful definitions: summary

- Always strive to find one
- They may pay off
- Sometimes sooner, sometimes later, sometimes never
  - this is research, one never knows
- But ugly (just-before-deadline) definitions won't!

What's next? A beautiful idea/concept

- Back to Hamming: we have a concept, now we need an attack
- A nice definition is not enough, we need to know how to use it
- In all of the above examples, there was such an attack

Relational databases

- Codd 1969, relational model. Not only a beautiful concept that separates logical and physical structure.
- It comes equipped with
  - a new notion: querying becomes declarative
  - Key language for querying: first-order logic (FO)
  - Nice math comes to the rescue again with procedural languages that are implemented by DBMSs
  - Not out of nowhere (relation algebra as an algebraization of FO)
- A key new element: all this happens on finite relations
- Most of the logic development before Codd was on infinite structures of interest in math

Relational databases and finite model theory

- Finite model theory studies the behavior of logics on finite structures
- The backbone of relational database theory
- Logics:
  - FO (as originally proposed by Codd)
  - FO+counting/aggregates aka SQL
  - FO+fixed point aka Datalog
- Lots of beautiful tools and results
- Also with some practical consequences
  - e.g., adding recursion in the SQL3 standard

XML: concepts

- The model:
  - labeled unranked trees (structure)
  - data trees (structure+data)
- Tools/concepts:
  - Automata on trees (defines schemas)
  - Logics on trees (particularly MSO = monadic second order logic, becomes the yardstick query language)
  - Temporal logics become the basis for navigation

An XML story: rediscovering beautiful concepts from the past

- XML schema specifications = essentially tree automata
- One of the most common schema specifications: DTDs (Document Type Definition)
  - reinvention of extended context-free grammars
- A common addition to DTDs: specialization
  - allows the same label to behave differently in different contexts
- A result rediscovered several times in the XML world: DTDs with specialization = unranked tree automata

An XML story: rediscovering beautiful concepts from the past

- XML schema specifications = essentially tree automata
- One of the most common schema specifications: DTDs (Document Type Definition)
  - reinvention of extended context-free grammars
- A common addition to DTDs: specialization
  - allows the same label to behave differently in different contexts
- A result rediscovered several times in the XML world: DTDs with specialization = unranked tree automata

- This result was proved in 1967, in a nice and beautiful paper by Thatcher:
  - it defined unranked tree automata
  - and proved that their languages are projections of derivation trees of extended context-free grammars
  - i.e., documents that conform to DTDs with specialization

Another XML story: a beautiful definition rediscovered by a committee

- XPath – navigational language of XML
- When you look at its syntax abstractly, it is exactly CTL*
  - without **U**
  - but with **F** and **G**
- In fact people later added **U**
  - and called it conditional XPath

Another XML story: a beautiful definition rediscovered by a committee

- XPath – navigational language of XML
- When you look at its syntax abstractly, it is exactly CTL*
  - without **U**
  - but with **F** and **G**
- In fact people later added **U**
  - and called it conditional XPath

- Moral: beautiful concepts live for a long time and show up unexpectedly in different areas

Graph databases: concepts/problems

- We are still in very early stages of graph db research
- Trying to understand models/languages
- But already discovering nice and neat problems

Graph databases: concepts/problems

- We are still in very early stages of graph db research
- Trying to understand models/languages
- But already discovering nice and neat problems
- Example: Let $R \subseteq \Sigma^* \times \Sigma^*$ be a regular relation on words
- Checking if R has a pair (w, w') so that w is a prefix of w' is easily solvable in linear time
- Checking if R has a pair (w, w') so that w is a suffix of w' – was not known!

Graph databases: concepts/problems

- We are still in very early stages of graph db research
- Trying to understand models/languages
- But already discovering nice and neat problems
- Example: Let $R \subseteq \Sigma^* \times \Sigma^*$ be a regular relation on words
- Checking if R has a pair (w, w') so that w is a prefix of w' is easily solvable in linear time
- Checking if R has a pair (w, w') so that w is a suffix of w' – was not known!
  - turns out to be undecidable

Beautiful concepts cont'd: an LTL example

- Problem: given a program P and a specification φ, does P satisfy φ?
- More interesting: Give me an example where P does not satisfy φ.
  - aka a bug in the program

Beautiful concepts cont'd: an LTL example

- Problem: given a program P and a specification φ, does P satisfy φ?
- More interesting: Give me an example where P does not satisfy φ.
  - aka a bug in the program
- Idea: P is naturally viewed as an automaton A_P (going from state to state)
- So if we turn $\neg\varphi$ into an automaton $A_{\neg\varphi}$, then bugs are

$$L(A_P) \cap L(A_{\neg\varphi}) \;=\; L(A_P \times A_{\neg\varphi})$$

- Verification becomes non-emptiness problem for automata

Beautiful concepts cont'd: an LTL example

- Problem: given a program P and a specification φ, does P satisfy φ?
- More interesting: Give me an example where P does not satisfy φ.
  - aka a bug in the program
- Idea: P is naturally viewed as an automaton A_P (going from state to state)
- So if we turn $\neg\varphi$ into an automaton $A_{\neg\varphi}$, then bugs are

$$L(A_P) \cap L(A_{\neg\varphi}) \;=\; L(A_P \times A_{\neg\varphi})$$

- Verification becomes non-emptiness problem for automata
- Nice and clean algorithms developed:
  - translation into nondeterministic automata (Vardi-Wolper)
  - translation into alternating automata (Vardi)
  - both beautiful and heavily used

A database concurrency example

- Each DMBS runs transactions
  - buying an airline ticket
  - withdrawing money from a bank, etc etc
- They need to run concurrently without causing conflicts
  - you don't want to have 2 people with the same seat on a plane
  - or wrong amount of money to disappear from your account
- Scheduling problems: schedules must be "equivalent" to serial ones (with no concurrency at all)

A database concurrency example

- Each DMBS runs transactions
  - buying an airline ticket
  - withdrawing money from a bank, etc etc
- They need to run concurrently without causing conflicts
  - you don't want to have 2 people with the same seat on a plane
  - or wrong amount of money to disappear from your account
- Scheduling problems: schedules must be "equivalent" to serial ones (with no concurrency at all)
- First attempt at the equivalence definition for schedules S_1, S_2:
  1. S_1 and S_2 have the same transactions in the same order.
  2. If transaction T_i reads the initial value of object A in S_1, it must also read the initial value of A in S_2.
  3. If T_i reads a value A written by T_j in S_1, it must also read the value of A written by T_j in S_2.
  4. For each data object A, the transaction (if any) that performs the final write on A in S_1 must also perform the final write on A in S_2.

.

A database concurrency example

- Each DMBS runs transactions
  - buying an airline ticket
  - withdrawing money from a bank, etc etc
- They need to run concurrently without causing conflicts
  - you don't want to have 2 people with the same seat on a plane
  - or wrong amount of money to disappear from your account
- Scheduling problems: schedules must be "equivalent" to serial ones (with no concurrency at all)
- First attempt at the equivalence definition for schedules S_1, S_2:
  1. S_1 and S_2 have the same transactions in the same order.
  2. If transaction T_i reads the initial value of object A in S_1, it must also read the initial value of A in S_2.
  3. If T_i reads a value A written by T_j in S_1, it must also read the value of A written by T_j in S_2.
  4. For each data object A, the transaction (if any) that performs the final write on A in S_1 must also perform the final write on A in S_2.
- Disgusting! Not only that, it's also NP-complete to test for.

Concurrency example: a beautiful notion

- Schedules S_1 and S_2 are equivalent if one can be transformed into the other by swapping non-conflicting operations:
  - on different items
  - or reading the same item
- A schedule equivalent to a serial one is called conflict-serializable
- Immediately works. Testing is linear-time.
- Found its way into SQL standard:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

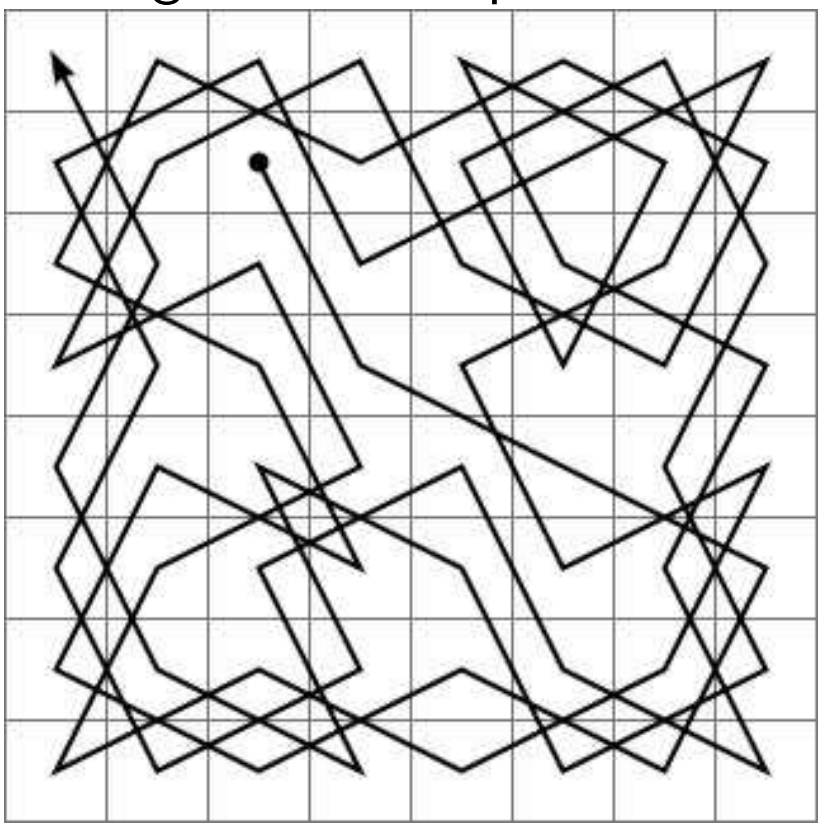
  - in fact this is the default

Beautiful proofs

- *"Beauty is the first test: there is no permanent place in this world for ugly mathematics."* (G H Hardy)
- We want a nice and clean argument
- Often we are not satisfied with hitting it with all the hammers we've got until it works
  - although the deadline-driven "culture" makes us do precisely that too often...
- The notion of what is beautiful here may be more controversial and dependent on one's background.
- A couple of examples now.

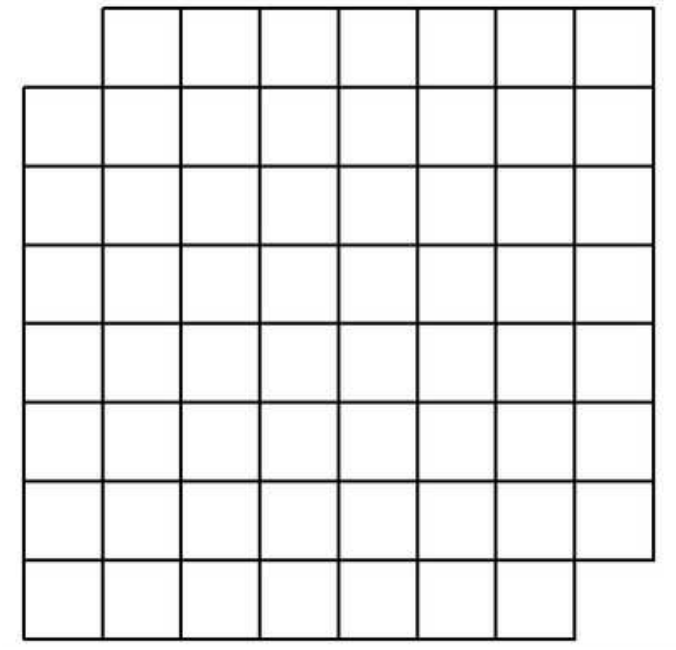
Chess problem: a cute short (and overused) proof

It is well known that a knight can cover the entire chessboard without visiting the same square twice.



Chess problem: a cute short (and overused) proof

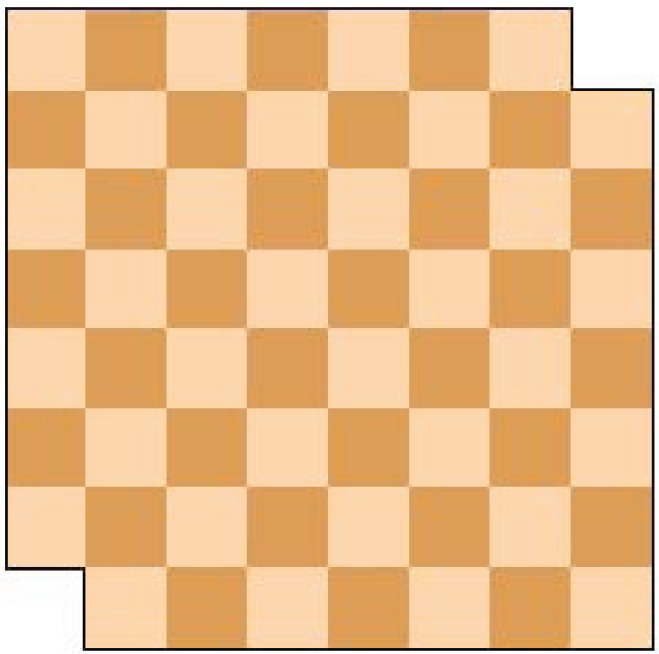
But what if we remove two squares in opposite corners?



Chess problem: a cute short (and overused) proof

But what if we remove two squares in opposite corners?

Solution: remember colors of squares



Opposite corners have the same color.

A knight move changes color – so no chance!

More controversial: there are infinitely many primes

1. Euler's product formula:

$$\prod_{p \text{ prime}} \frac{1}{1-1/p^2} = \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

2. If there were finitely many primes, the product would be a rational number, and hence π would be algebraic
3. But π is transcendental, so there are infinitely many primes.

A side remark

- Thinking as a mathematician doesn't always help in CS
- What comes next? 1,2,4,8,16,?
- If you are a computer scientist, you say 32 without thinking.
- But why? because we have a function f so that $f(0) = 1,\ f(1) = 2,\ \ldots, f(4) = 16$ and we are looking for $f(5)$

A side remark

- Thinking as a mathematician doesn't always help in CS
- What comes next? 1,2,4,8,16,?
- If you are a computer scientist, you say 32 without thinking.
- But why? because we have a function f so that $f(0) = 1,\ f(1) = 2,\ \ldots, f(4) = 16$ and we are looking for $f(5)$
- No mathematician would ever approximate by an exponential function – instead they use the lowest degree polynomial
- The unique polynomial of degree 4 agreeing with f is

$$f(x) = \frac{1}{24} \cdot \left((x-1)^4 - 6(x-1)^3 + 23(x-1)^2 - 18x + 42\right)$$

- And the value of $f(5)$ is

A side remark

- Thinking as a mathematician doesn't always help in CS
- What comes next? 1,2,4,8,16,?
- If you are a computer scientist, you say 32 without thinking.
- But why? because we have a function f so that $f(0) = 1,\ f(1) = 2,\ \ldots, f(4) = 16$ and we are looking for $f(5)$
- No mathematician would ever approximate by an exponential function – instead they use the lowest degree polynomial
- The unique polynomial of degree 4 agreeing with f is

$$f(x) = \frac{1}{24} \cdot \left((x-1)^4 - 6(x-1)^3 + 23(x-1)^2 - 18x + 42\right)$$

- And the value of $f(5)$ is

- 31

Three proofs that changed things

- 0-1 law for first-order logic
- A short proof of Büchi Theorem (MSO = automata) using composition
- Two-phase locking guarantees serializability

0-1 law

- Most of the things you want to know are really boring (at least at infinity)
- First-order logic and many database query languages are such

0-1 law

- Most of the things you want to know are really boring (at least at infinity)
- First-order logic and many database query languages are such
- Pick a database "at random".
- Check if it satisfies a property $\mathcal{P}$.
- What's the probability of that?
- If $\mathcal{P}$ is expressed in FO (or relational calculus/algebra), it is 0 or 1: 0-1 law.

0-1 law

- Most of the things you want to know are really boring (at least at infinity)
- First-order logic and many database query languages are such
- Pick a database "at random".
- Check if it satisfies a property $\mathcal{P}$.
- What's the probability of that?
- If $\mathcal{P}$ is expressed in FO (or relational calculus/algebra), it is 0 or 1: 0-1 law.
- Pick a graph at random:
  - throw n vertices
  - for each pair of vertices toss a coin to see if they are connected
  - compute the probability for each n and see how it behaves as $n \longrightarrow \infty$

The 0-1 law for FO story

- First proved by 4 Russians (Glebskii, Kogan, Liogonki, and Talanov) in 1969
- The proof was very proletarian
  - emphasis on heavy tools, weight rather than technique
- English translations appeared in the early 1970s, but were very hard to follow.
- Ron Fagin could not follow them, and came up with a beautiful proof.
- Sounds like a recipe. Take:
  - a bit of probability
  - a bit of combinatorics
  - a bit of logic
- mix them quickly and get the result.

Fagin's proof

- The probability bit. Look at the statement $EA_{n,m}$

  for any disjoint sets X and Y with n and m nodes in a graph, there is a node v connected to everything in X and to nothing in Y

  - with probability 1 all such statements are true
- The combinatorial bit (that goes infinite). There is exactly one countable graph **G** that satisfies all the $EA_{n,m}$s.
- The logic bit. A first-order sentence is true with probability 1 iff it is true in **G** (traditional proof via compactness).
- Since in a concrete structure (like **G**) every sentence is either true or false, the result follows.
- Magic!

0-1 laws: what happened later

- Fagin's proof gave a methodology for proving 0-1 laws.
- We now have them for many logics (e.g., fixed-point logics)
- We also have them for complex probability distributions

0-1 laws: what happened later

- Fagin's proof gave a methodology for proving 0-1 laws.
- We now have them for many logics (e.g., fixed-point logics)
- We also have them for complex probability distributions
- For instance, with n vertices we can put edges with probabilities $\frac{1}{n^\alpha}$, for $0 < \alpha < 1$.
- An amazing result by Spencer-Shelah: FO has the 0-1 law iff α is irrational.
- There are lots of papers and books on the subject.

A beautiful proof found useful after 30 years

For words/trees and similar structures, there are many results saying

$$\text{MSO} = \text{Automata}$$

- Monadic Second Order Logic — adds quantification over sets to FO: $\exists X_1 \forall X_2 \ldots\ \varphi(X_1, X_2, \ldots)$ where φ is FO.

We shall deal with words for simplicity

Words as structures/databases

A word w over $\Sigma = \{a_1, \ldots, a_m\}$ is a database with relations $E(\cdot,\cdot),\ L_1(\cdot), \ldots, L_m(\cdot)$:

- E is the ordering of positions;
- L_i's define labelings.

$w = a_1 a_2 a_1 a_2$:

positions $0, 1, 2, 3$; positions 0,2,3 labeled a_1; position 1 labeled a_2

$E =$

0	1
1	2
2	3
0	2
1	3
0	3

$L_1 =$

0
2
3

$L_2 =$

1

MSO over words

Each MSO sentence φ defines a language

$$\mathcal{L}(\varphi) \;=\; \{w \in \Sigma^* \mid w \models \varphi\}$$

Theorem (Büchi, Elgot, Trakhtenbrot 1960)

MSO-definability $=$ *Regular languages*

A similar result holds for trees as well – both binary and unranked.

Original proof: induction on formulae. Nothing pretty, a bit tedious, and it works

MSO over words

Each MSO sentence φ defines a language

$$\mathcal{L}(\varphi) \;=\; \{w \in \Sigma^* \mid w \models \varphi\}$$

Theorem (Büchi, Elgot, Trakhtenbrot 1960)

MSO-definability = Regular languages

A similar result holds for trees as well – both binary and unranked.

Original proof: induction on formulae. Nothing pretty, a bit tedious, and it works

But there is a very short and beautiful proof of MSO $\subseteq$ automata. We present it for FO to make things simpler.

Types

Each FO sentence is a disjunction of types.

- Rank-k type $\mathrm{tp}_k(D)$: set of all sentences of quantifier depth up to k true in a structure D.
- Types are finite objects, definable in the logic: finitely many distinct FO sentences of quantifier rank k, up to logical equivalence.

Types

Each FO sentence is a disjunction of types.

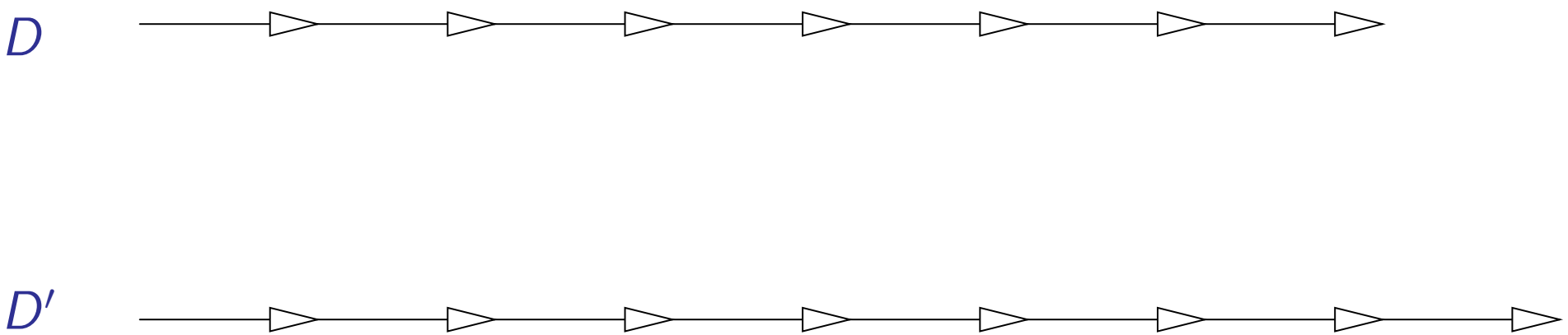
- Rank-k type $\mathrm{tp}_k(D)$: set of all sentences of quantifier depth up to k true in a structure D.
- Types are finite objects, definable in the logic: finitely many distinct FO sentences of quantifier rank k, up to logical equivalence.

$$\mathrm{tp}_k(D) = \mathrm{tp}_k(D')$$

$$\Updownarrow$$

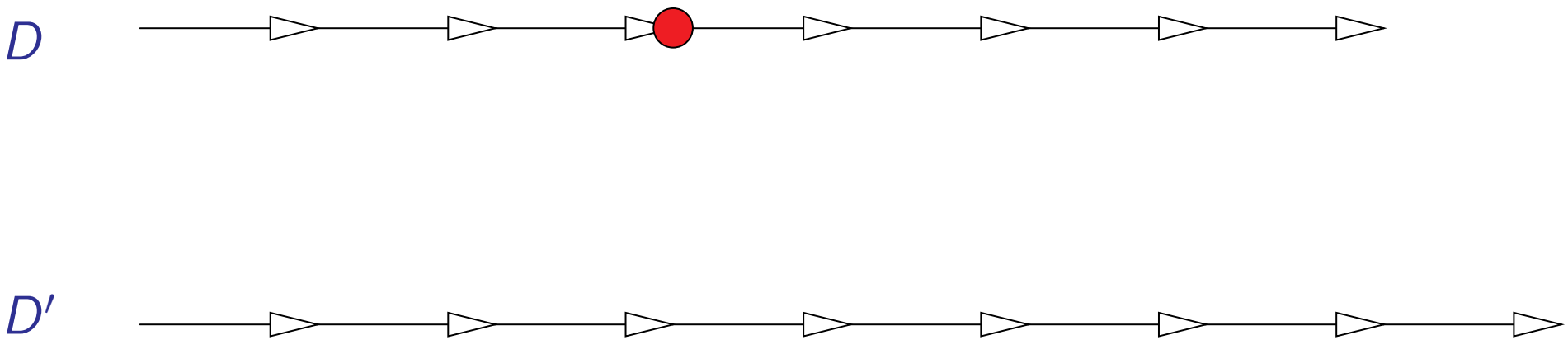
A player has a win in a specific k-round game on D and D'.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



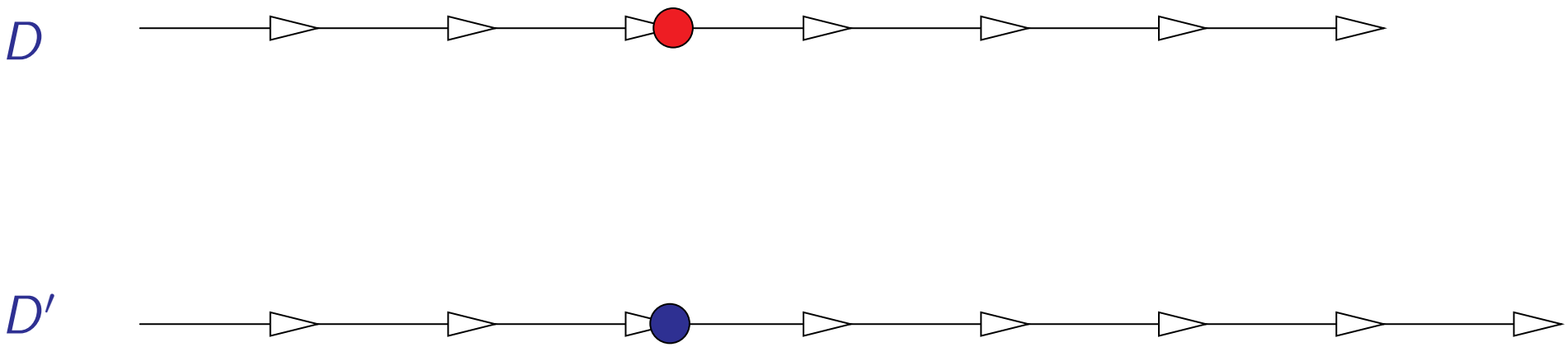
Spoiler and Duplicator play for 3 rounds.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



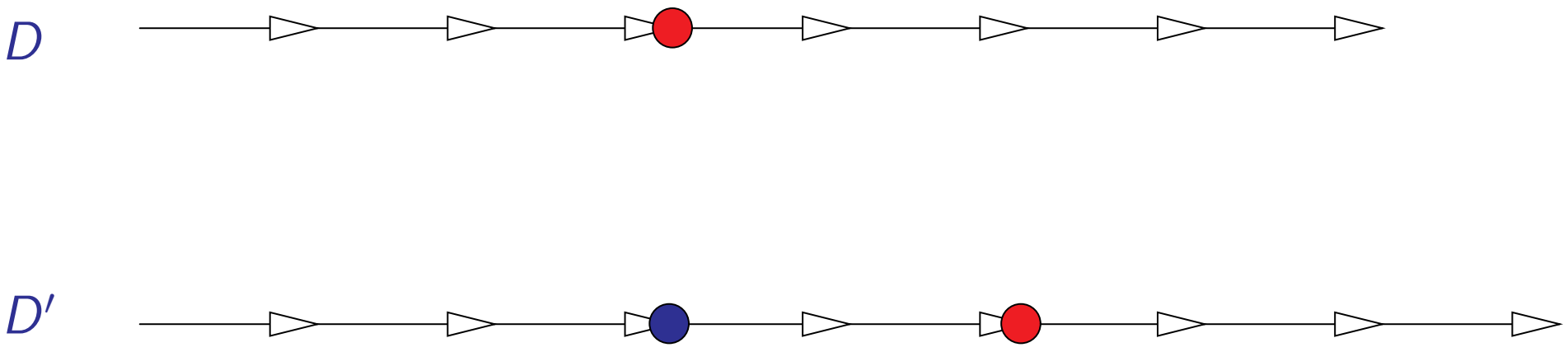
Spoiler and Duplicator play for 3 rounds.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



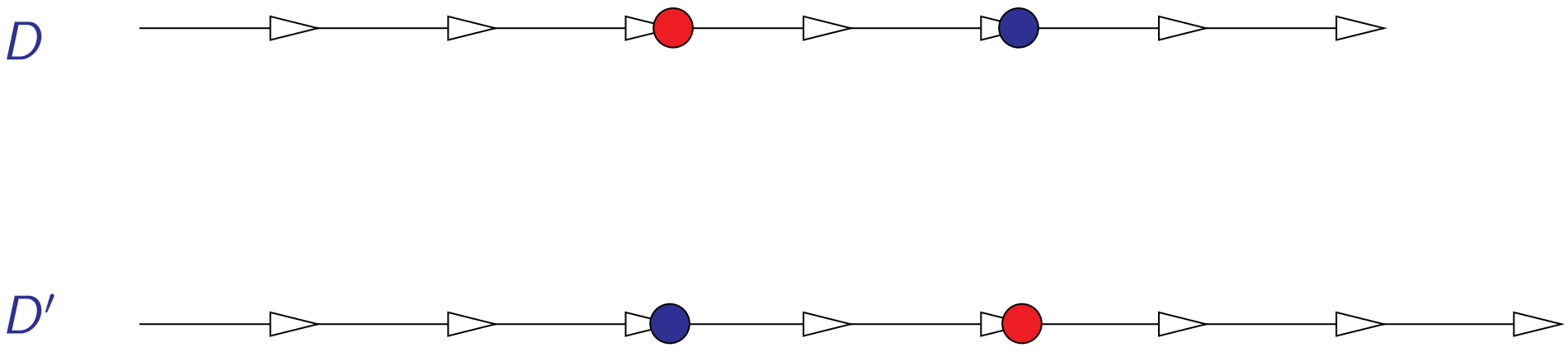
Spoiler and Duplicator play for 3 rounds.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



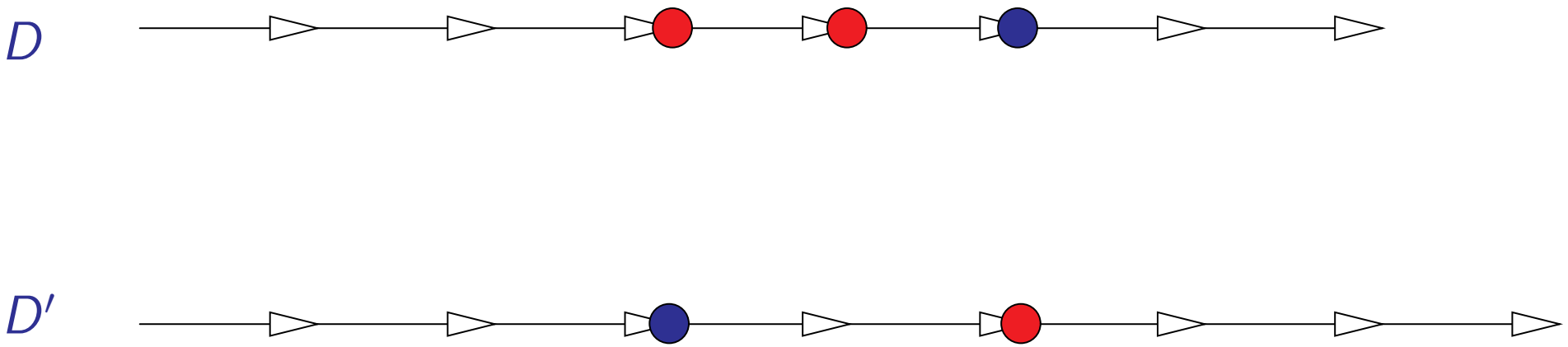
Spoiler and Duplicator play for 3 rounds.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



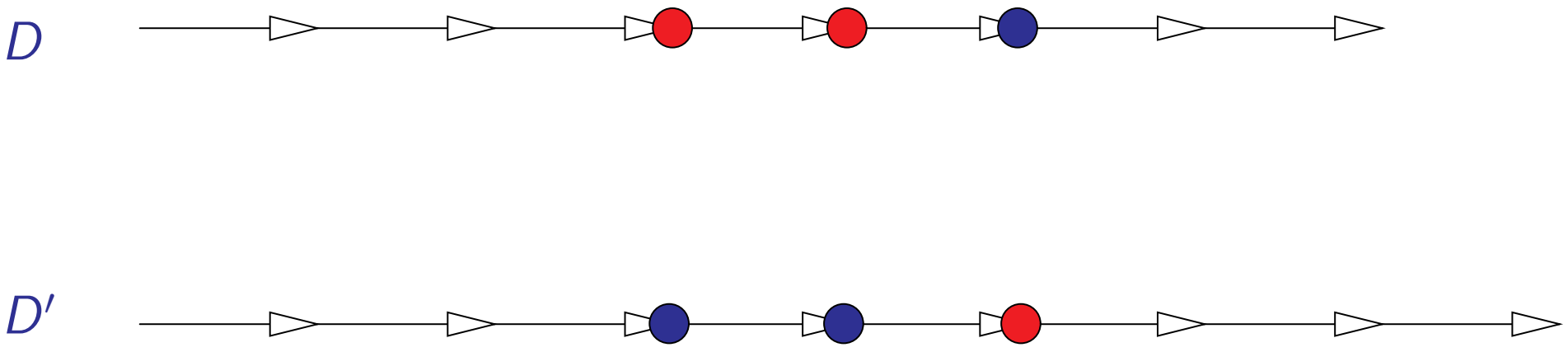
Spoiler and Duplicator play for 3 rounds.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



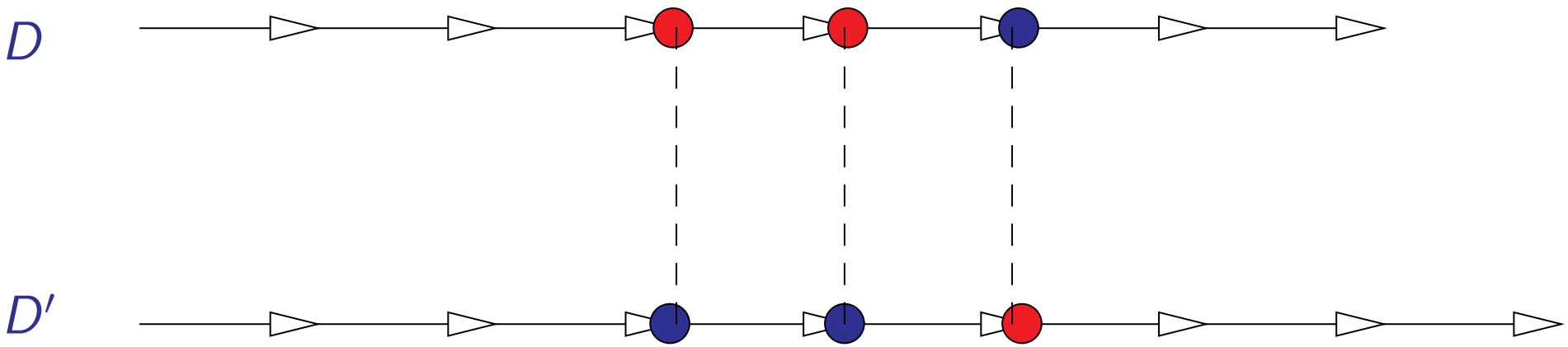
Spoiler and Duplicator play for 3 rounds.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



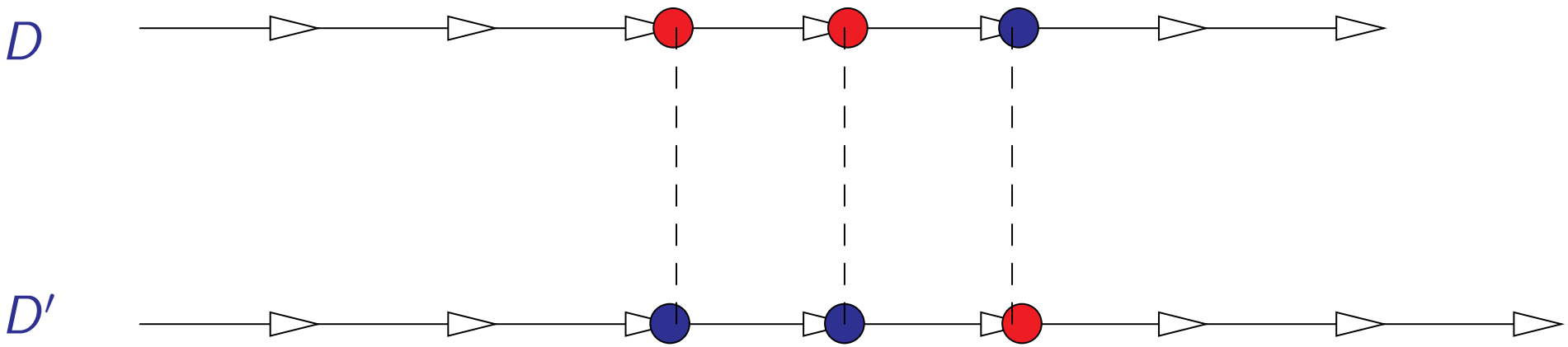
Spoiler and Duplicator play for 3 rounds.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



Spoiler and Duplicator play for 3 rounds.

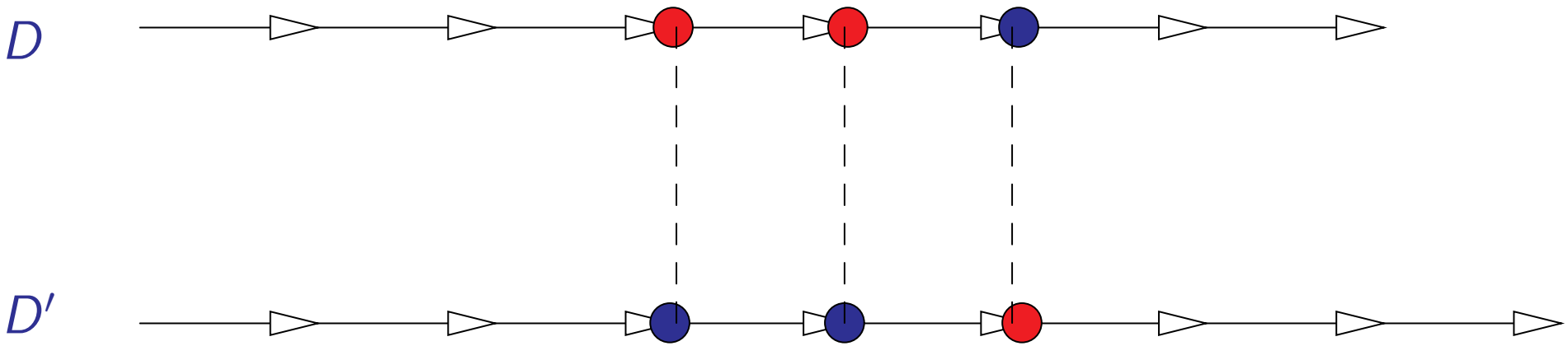
Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



Spoiler and Duplicator play for 3 rounds.

The duplicator wins in 3 rounds.

Ehrenfeucht-Fraïssé game, played by Spoiler and Duplicator



$$\mathrm{tp}_k(D) = \mathrm{tp}_k(D')$$

$$\Updownarrow$$

Duplicator has a winning strategy in the k-round game on D and D'.

From MSO to automata via composition

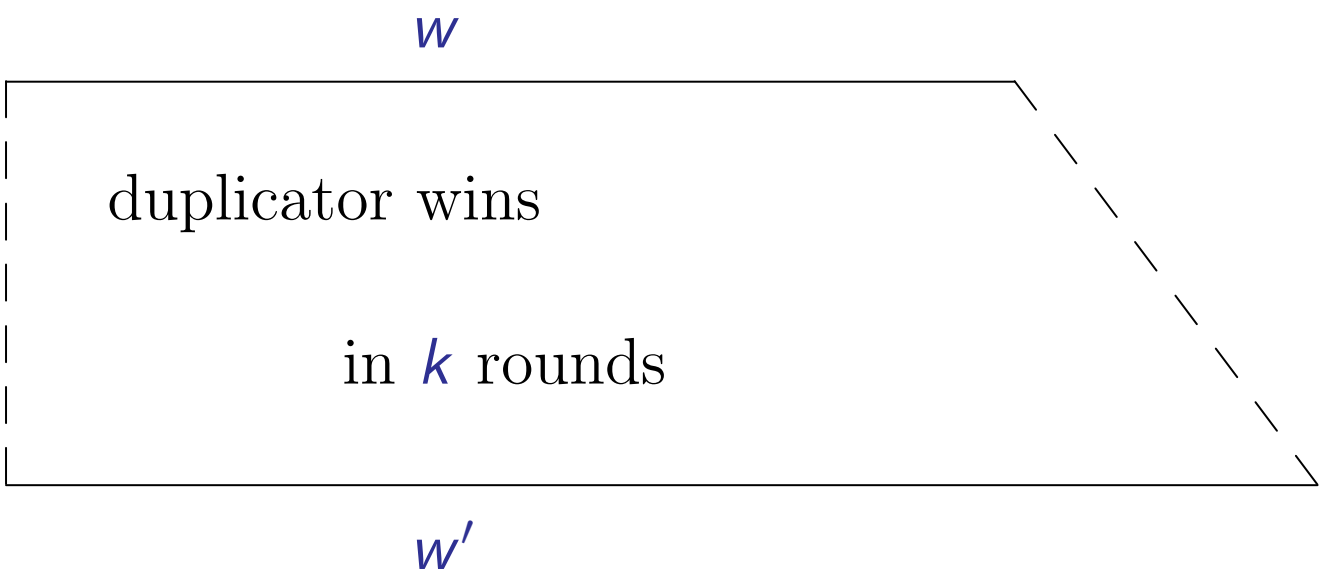
Rank-k type of w uniquely determines rank-k type of $w \cdot a$.

If $\mathrm{tp}_k(w) = \mathrm{tp}_k(w')$, then $\mathrm{tp}_k(w \cdot a) = \mathrm{tp}_k(w' \cdot a)$: compose games!

From MSO to automata via composition

Rank-k type of w uniquely determines rank-k type of $w \cdot a$.

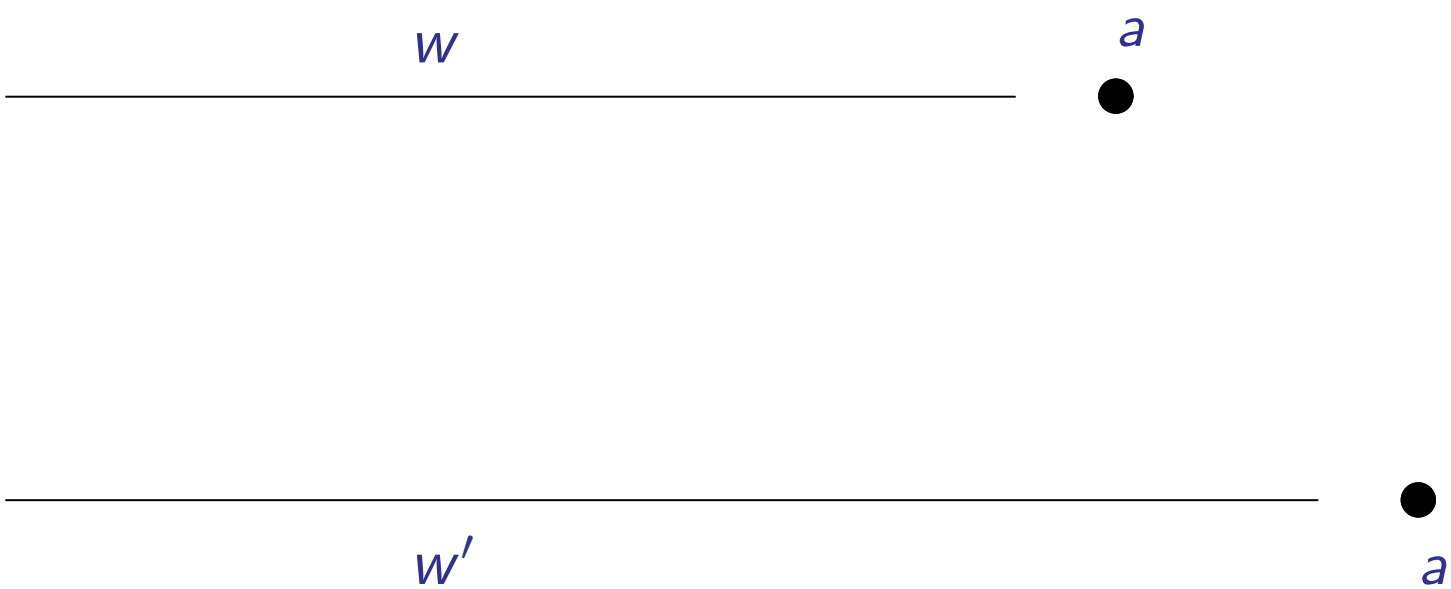
If $\mathrm{tp}_k(w) = \mathrm{tp}_k(w')$, then $\mathrm{tp}_k(w \cdot a) = \mathrm{tp}_k(w' \cdot a)$: compose games!



From MSO to automata via composition

Rank-k type of w uniquely determines rank-k type of $w \cdot a$.

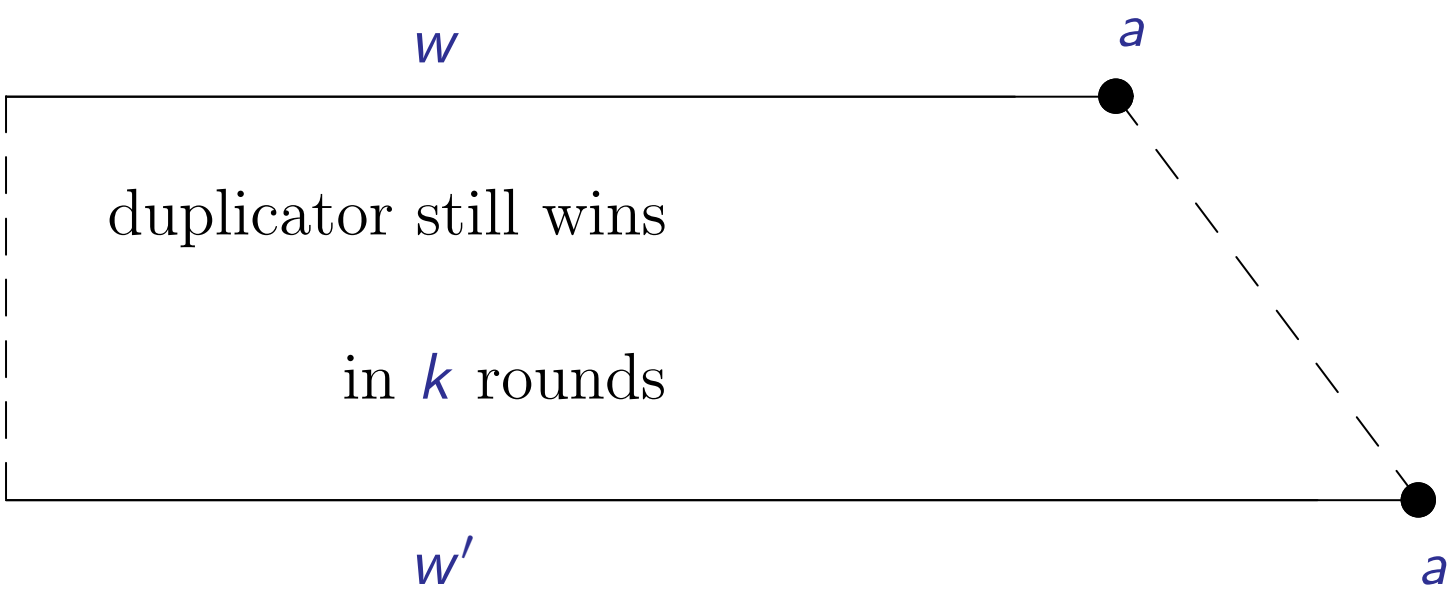
If $\mathrm{tp}_k(w) = \mathrm{tp}_k(w')$, then $\mathrm{tp}_k(w \cdot a) = \mathrm{tp}_k(w' \cdot a)$: compose games!



From MSO to automata via composition

Rank-k type of w uniquely determines rank-k type of $w \cdot a$.

If $\mathrm{tp}_k(w) = \mathrm{tp}_k(w')$, then $\mathrm{tp}_k(w \cdot a) = \mathrm{tp}_k(w' \cdot a)$: compose games!



From MSO to automata: automata compute types

The rank-k type of w uniquely determines the rank-k type of $w \cdot a$.

Ladner (1977) construction of a Deterministic Automaton for sentence φ:

- States are rank-k types;
- Initial state: the type of the empty word;
- Final states: those types whose disjunction forms φ.
- Transition $\delta(\tau, a)$: the type of $w \cdot a$ if the type of w is τ.

After reading w, the state of the automaton is $\mathrm{tp}_k(w)$.

Languages for information extraction

Ladner's proof was dismissed as a practical tool: the number of types is astronomical.

And yet it turned out useful for information extraction from XML documents:

- Work by Gottlob, Koch, and colleagues; Lixto system

We demonstrate the idea on words; it works for trees as well.

Languages for information extraction

Ladner's proof was dismissed as a practical tool: the number of types is astronomical.

And yet it turned out useful for information extraction from XML documents:

- Work by Gottlob, Koch, and colleagues; Lixto system

We demonstrate the idea on words; it works for trees as well.

Use the composition method:

$$1) \quad \begin{array}{l} \mathrm{tp}_k(w) = \mathrm{tp}_k(w') \\ \mathrm{tp}_k(u) = \mathrm{tp}_k(u') \end{array} \quad \Rightarrow \quad \mathrm{tp}_k(w \cdot a \cdot u) = \mathrm{tp}_k(w' \cdot a \cdot u')$$

$$2) \quad \mathrm{tp}_k(u) = \mathrm{tp}_k(w) \quad \Rightarrow \quad \mathrm{tp}_k(w^{-1}) = tp_k(u^{-1})$$

Language for extracting positions in words

How to express MSO (or FO) $\varphi(x)$ over words?

Idea: for $w \cdot a \cdot u$, compute

1. $\mathrm{tp}_k(w)$ going forward from the first position;
2. $\mathrm{tp}_k(u^{-1})$ going backwards from the last position;
3. These types tell us whether the a position is selected.

Express this in Datalog. Compute $\mathrm{tp}_k(w)$ going forward – use predicates U_τ for types:

$$\begin{array}{lll} U_{\tau_a}(x) & :- & \mathrm{First}(x), L_a(x); \qquad a \in \Sigma \\ U_{\tau'}(x) & :- & Succ(y,x), L_a(x), U_\tau(y); \qquad a \in \Sigma,\ \delta(\tau,a) = \tau' \end{array}$$

Datalog program cont'd

- Types going forward:

$$\begin{array}{lll} U_{\tau_a}(x) & :- & \mathrm{First}(x), L_a(x); \qquad a \in \Sigma \\ U_{\tau'}(x) & :- & Succ(y,x), L_a(x), U_\tau(y); \qquad a \in \Sigma,\ \delta(\tau,a) = \tau' \end{array}$$

- Types going backwards V_τ: symmetric.
- Answer – for all triples (τ, a, τ') saying that a is selected, add:

$$\mathrm{Answer}(x) :- U_\tau(y), Succ(y,x), P_a(x), Succ(x,z), V_{\tau'}(z)$$

- We used Monadic Datalog: all idb predicates are monadic.
  - Edb predicates: successor, labelings, First and Last.
- It captures MSO over words.
- Complexity of evaluating program P on w:

$$O(\|P\| \cdot |w|)$$

Composition proof is practical!

Composition technique suggested using monadic datalog:

- captures MSO;
- has very good complexity bounds;
- is in fact a convenient language for the programmer.

The approach works for trees and yields many XML languages:

- Monadic datalog captures MSO for trees, with the same complexity – one needs to add predicates for the root, leaves, first and last children of nodes (Gottlob, Koch, '01)
- Led to a practical system and a company!

A proof with lots of practical implications

- How to ensure good concurrent schedules? Make them serializable.
- Can be tested in linear time. But still impractical:
  - cannot recompute each time a new transaction appears.

A proof with lots of practical implications

- How to ensure good concurrent schedules? Make them serializable.
- Can be tested in linear time. But still impractical:
  - cannot recompute each time a new transaction appears.
- Solution: find a policy so that if each transaction follows it, the schedule is guaranteed to be serializable.
- Policy – two-phase locking:
  - each item needs to be locked before it is read, and unlocked after it is no longer needed
  - a transaction cannot request a new lock after it released at least one

A proof with lots of practical implications

- How to ensure good concurrent schedules? Make them serializable.
- Can be tested in linear time. But still impractical:
  - cannot recompute each time a new transaction appears.
- Solution: find a policy so that if each transaction follows it, the schedule is guaranteed to be serializable.
- Policy – two-phase locking:
  - each item needs to be locked before it is read, and unlocked after it is no longer needed
  - a transaction cannot request a new lock after it released at least one
- It does the job, and does so optimally: a very short and elegant graph-theoretic proof
- The policy is implemented in all big DBMS.
- Another Turing award (Gray)

Effects of beautiful solutions

Beautiful definitions: long lasting concepts that change fields.

Effects of beautiful solutions

Beautiful definitions: long lasting concepts that change fields.

Beautiful concepts/ideas: paradigm shifts, they attract people to the field.

Effects of beautiful solutions

Beautiful definitions: long lasting concepts that change fields.

Beautiful concepts/ideas: paradigm shifts, they attract people to the field.

Beautiful proofs: can have both theoretical and practical impacts.

Effects of beautiful solutions

Beautiful definitions: long lasting concepts that change fields.

Beautiful concepts/ideas: paradigm shifts, they attract people to the field.

Beautiful proofs: can have both theoretical and practical impacts.

Do all beautiful results have impact? Of course not.

But most definitions/ideas/results that have a big impact are indeed nice and beautiful.

Is it really worth it?

- Absolutely!
- To start with, you are not likely to address a really big problem if you are doing something ugly (back to Hamming).

Is it really worth it?

- Absolutely!
- To start with, you are not likely to address a really big problem if you are doing something ugly (back to Hamming).
- Even more importantly, we are researchers.

Is it really worth it?

- Absolutely!
- To start with, you are not likely to address a really big problem if you are doing something ugly (back to Hamming).
- Even more importantly, we are researchers.
- We must enjoy what we do!

Is it really worth it?

- Absolutely!
- To start with, you are not likely to address a really big problem if you are doing something ugly (back to Hamming).
- Even more importantly, we are researchers.
- We must enjoy what we do!
- If we don't, we stop being researchers.

Is it really worth it?

- Absolutely!
- To start with, you are not likely to address a really big problem if you are doing something ugly (back to Hamming).
- Even more importantly, we are researchers.
- We must enjoy what we do!
- If we don't, we stop being researchers.
- It's hard to enjoy ugly things – and it is easy to enjoy beautiful ones.

Personal experiences

- Three examples: I spent a lot of time looking for beautiful solutions
- Example 1: First-Order Logic plus Constraints as a query language
  - aka embedded finite model theory
- Example 2: Normalization of databases and documents
- Example 3: Consistency of XML specifications

FO + Complex constraints

- Graph nodes: numbers. Query "does a graph lie on a circle?":

$$\exists r \exists a \exists b \; \forall x \forall y \; E(x,y) \rightarrow (x-a)^2 + (y-b)^2 = r^2$$

- What is the power of such extensions? Can graph connectivity be expressed?
- Look at queries that talk about proper graph properties (formally, isomorphism types of graphs).
- The answer depends on the class of numbers and arithmetic operations.
- More precisely, on deep mathematical properties of the underlying structure

FO + Complex constraints cont'd

- Graph connectivity is expressible over $\mathbb{N}$ with arithmetic $0, 1, +, \times, <$ (rather easy)
- But it is not expressible over $\mathbb{R}$ with arithmetic
- This result is much harder
  - conjecture by Kanellakis in the late 1980s
  - several wrong "proofs" in the early 90s
  - A solution for $+$ in 1995
  - Finally a proof (M. Benedikt) in 1996
  - uses nonstandard models, model theory of o-minimal structures, algebraic topology
- In 1998: a shorter constructive proof, still relies heavily on o-minimality
  - Citations: a fraction of the 1996 paper
- In 2004, a clean 4-page long proof, based on basic principles

Normalization: relations and XML

- Normalization: a way of organizing your data
  - eliminates redundancies, update anomalies
  - one of the earliest and best understood subjects for relational databases
- Question: How to normalize XML?

Normalization: relations and XML

- Normalization: a way of organizing your data
  - eliminates redundancies, update anomalies
  - one of the earliest and best understood subjects for relational databases
- Question: How to normalize XML?
- Answer: A normal form XNF proposed in 2002 (Arenas, L.)
  - many results proved, lots of nice properties
  - but still not clear whether it was the right one

Normalization: relations and XML

- Normalization: a way of organizing your data
  - eliminates redundancies, update anomalies
  - one of the earliest and best understood subjects for relational databases
- Question: How to normalize XML?
- Answer: A normal form XNF proposed in 2002 (Arenas, L.)
  - many results proved, lots of nice properties
  - but still not clear whether it was the right one
- Justifying normal forms: a new approach based on classical information theory
- Measures expected redundancy in a design, tries to minimize it
- Justifies all standard relational normal forms, and the new XML form

Normalization: relations and XML

- Normalization: a way of organizing your data
  - eliminates redundancies, update anomalies
  - one of the earliest and best understood subjects for relational databases
- Question: How to normalize XML?
- Answer: A normal form XNF proposed in 2002 (Arenas, L.)
  - many results proved, lots of nice properties
  - but still not clear whether it was the right one
- Justifying normal forms: a new approach based on classical information theory
- Measures expected redundancy in a design, tries to minimize it
- Justifies all standard relational normal forms, and the new XML form
- Citation checks:
  - original big-hammer paper: 400+
  - the clean information-theoretic one: 120

Consistency of XML specifications

- In relational databases, we define tables with some basic constraints
  - keys and foreign keys
- These specifications are always consistent
- No longer the case for XML. Discovered in (Fan, L., 2001):
  - consistency is undecidable
  - in the most common case of document nodes carrying a single attribute, it is NP-complete
  - Proletarian upper bound proof: Mao and Stalin would have been proud

Consistency of XML specifications

- In relational databases, we define tables with some basic constraints
  - keys and foreign keys
- These specifications are always consistent
- No longer the case for XML. Discovered in (Fan, L., 2001):
  - consistency is undecidable
  - in the most common case of document nodes carrying a single attribute, it is NP-complete
  - Proletarian upper bound proof: Mao and Stalin would have been proud
- 10 years later: a new idea (with David, Tan, L.)
- Turn everything into linear constraints and apply integer linear programming. A cute one-page proof of a stronger result

Consistency of XML specifications

- In relational databases, we define tables with some basic constraints
  - keys and foreign keys
- These specifications are always consistent
- No longer the case for XML. Discovered in (Fan, L., 2001):
  - consistency is undecidable
  - in the most common case of document nodes carrying a single attribute, it is NP-complete
  - Proletarian upper bound proof: Mao and Stalin would have been proud
- 10 years later: a new idea (with David, Tan, L.)
- Turn everything into linear constraints and apply integer linear programming. A cute one-page proof of a stronger result
- Citation checks:
  - original proletarian paper from 2001: 300
  - the clean and neat 2011 paper: 6

Should one stop trying?

Of course not! Citations don't tell the whole story.
But I'd like to finish citing others nonetheless.

Should one stop trying?

Of course not! Citations don't tell the whole story.
But I'd like to finish citing others nonetheless.

Being a good Russian, I start with Dostoevsky: *Beauty will save the world*

Should one stop trying?

Of course not! Citations don't tell the whole story.
But I'd like to finish citing others nonetheless.

Being a good Russian, I start with Dostoevsky: *Beauty will save the world*

And an immediate refutation by another giant: *How could that be possible? When in bloodthirsty history did beauty ever save anyone from anything? Ennobled, uplifted, yes - but whom has it saved?*
(Solzhenitsyn)

Should one stop trying?

Of course not! Citations don't tell the whole story.
But I'd like to finish citing others nonetheless.

Being a good Russian, I start with Dostoevsky: *Beauty will save the world*

And an immediate refutation by another giant: *How could that be possible? When in bloodthirsty history did beauty ever save anyone from anything? Ennobled, uplifted, yes - but whom has it saved?*
(Solzhenitsyn)

But it has saved – concepts, ideas, fields, results – in science!

Should one stop trying?

Of course not! Citations don't tell the whole story.
But I'd like to finish citing others nonetheless.

Being a good Russian, I start with Dostoevsky: *Beauty will save the world*

And an immediate refutation by another giant: *How could that be possible? When in bloodthirsty history did beauty ever save anyone from anything? Ennobled, uplifted, yes - but whom has it saved?* (Solzhenitsyn)

But it has saved – concepts, ideas, fields, results – in science!

Returning to a remote forest in Massachusetts: *The eye which can appreciate the naked and absolute beauty of a scientific truth is far more rare than that which is attracted by a moral one.* (Thoreau)

Should one stop trying?

Of course not! Citations don't tell the whole story.
But I'd like to finish citing others nonetheless.

Being a good Russian, I start with Dostoevsky: *Beauty will save the world*

And an immediate refutation by another giant: *How could that be possible? When in bloodthirsty history did beauty ever save anyone from anything? Ennobled, uplifted, yes - but whom has it saved?* (Solzhenitsyn)

But it has saved – concepts, ideas, fields, results – in science!

Returning to a remote forest in Massachusetts: *The eye which can appreciate the naked and absolute beauty of a scientific truth is far more rare than that which is attracted by a moral one.* (Thoreau)

If he was right and we belong to that select group, we must keep looking for beauty in our work.

Thank you!

Questions?