How To Use Automata for Solving Mathematical Problems

Johannes Kalmbach

University of Freiburg

johannes.kalmbach@gmail.com

January 26, 2018

Introduction

- Talk about **MATHEMATICS**
- How to use automata to prove mathematical theorems
- Especially relevant if "pure" mathematical methods were not sufficient

Binary squares

- With natural numbers, "$\cdot$" generally denotes multiplication, so $n^2 := n \cdot n$ gives us

$$5^2 = 25$$

- With formal languages, "$\cdot$" generally denotes concatenation, so $w^2 := w \cdot w$ gives us

$$1011^2 = 1011\ 1011$$

Definition

Set of *binary squares* $\mathcal{B}$:= all possible results of such square computations (only canonical binary representations)

$$\mathcal{B} := \{ww | w \in \{1\} \cdot \{0,1\}^*\} \cup \{\epsilon\}$$

- Note: 0 is a binary square (canonical binary representation is the empty string ϵ with $\epsilon^2 = \epsilon$)

Theorem

Every natural number $n > 686$ is the sum of four binary squares.

- There are 56 numbers ≤ 686 for which this does not hold, e.g. 2 and 686
- Original version: Every natural number is them sum of four "ordinary" squares (Joseph-Louis Lagrange, 1736-1813)
- Example:

$$6 = 3 + 3 + 0 + 0 = 11_2 + 11_2 + \epsilon_2 + \epsilon_2$$

The Main Lemma

The following lemma will help us prove Lagrange's Theorem for Binary Squares:

Lemma (part 1)

Every length-n integer, n odd, $n \geq 13$, is the sum of binary squares as follows: either

- one of length $n-1$ and one of length $n-3$, or
- two of length $n-1$ and one of length $n-3$, or
- one of length $n-1$ and two of length $n-3$, or
- one each of lengths $n-1, n-3$ and $n-5$
- two of length $n-1$ and two of length $n-3$, or
- two of length $n-1$, one of length $n-3$ and one of length $n-5$

The Main Lemma

Lemma (part 2)

Every length-n integer, n even, $n \geq 18$ is the sum of binary squares as follows: either:

- two of length $n-2$ and two of length $n-4$, or
- three of length $n-2$ and one of length $n-4$, or
- one each of lenghts $n, n-4$ and $n-6$, or
- two of lengths $n-2$, one of length $n-4$, and one of length $n-6$.

Main Lemma

Theorem (repetition)

Every natural number $n > 686$ is the sum of four binary squares.

- $a \in \mathbb{N}, a \geq 2^{17}$ has binary representation of length ≥ 18, existence of binary square summands follows from lemma
- For $686 < a < 2^{17}$ find summands by brute-force computation
- "Missing" summands can be set to 0 (which is binary square as seen above)
- Proving the main lemma also proves the theorem.

Solving Mathematical Problems Using Automata

Given: odd-length part of main lemma, three different formulations:

Main Lemma (repetition of part 1)

- Every length-n integer, n odd, $n \geq 13$, is the sum of binary squares as follows: [several cases . . .]
- Predicate Logic: $\forall x \in \mathbb{N} : E(x) \vee S(x) \vee \bigvee M_i(x)$
- Sets: $\mathbb{N} = E \cup S \cup \bigcup M_i$

where

- $E(x)$ is true $\Leftrightarrow x \in E \Leftrightarrow x$ has even (non-odd) length in binary representation
- $S(x)$ is true $\Leftrightarrow x \in S \Leftrightarrow x$ is too short to be handled by the lemma (shorter than 13)
- $M_i(x)$ is true $\Leftrightarrow x \in M_i \Leftrightarrow$ the i-th case of main lemma applies to x.

Main Lemma (part 1, expressed as sets)

$$\mathbb{N} = E \cup S \cup \bigcup M_i \tag{1}$$

Approach:

1. find a representation of $\mathbb{N}$ as Kleene closure Σ^* of alphabet Σ, so a bijective mapping $r : \mathbb{N} \to \Sigma^*$ (e.g. canonical binary representation and $\Sigma = \{0, 1\}$)
2. for each of the sets mentioned in (1) construct an automaton that accepts exactly this set, e.g.

$$L_E = \{r(x) \in \mathbb{N} : x \text{ has even length}\}$$

3. show that (1) holds, so that

$$L_{\mathbb{N}} = L_E \cup L_S \cup \bigcup L_{M_i}$$

Solving Mathematical Problems Using Automata

$$L_{\mathbb{N}} = L_E \cup L_S \cup \bigcup L_{M_i}$$

Prerequisites

- We must find an automata model which is powerful enough to express all of the sets mentioned above.
- In our chosen model, the equation above must be decidable.
- True for nondeterministic finite automata (**NFAs**): closed under union and equality is decidable.
- Nondeterministic $\Rightarrow$ able to "guess" summands.

Automata for the main lemma

main lemma (part 1)

$$L_{\mathbb{N}} = L_E \cup L_S \cup \bigcup L_{M_i}$$

- Automata for $L_{\mathbb{N}}$, L_E (even lenght) and L_S (shorter than 13) can be constructed easily.
- In the following, construct automaton L_{M_1} for first case of main lemma:

L_{M_1}

A binary number x of odd length n $\geq$ 13 is in L_{M_1} iff x is the sum of two binary squares of length $n-1$ and $n-3$

Automaton for L_{M_1}

L_{M_1}

A binary number x of odd length n ≥ 13 is in L_{M_1} iff x is the sum of two binary squares of length $n-1$ and $n-3$

- Idea: NFA gets x as an input and guesses the summands in a nondeterministic way.
- Make sure that only valid summands can be guessed (binary squares and length constraints)
- Accept x iff valid summands a, b could be guessed.

$$
\begin{array}{cccccccccccccc}
 & & & b_{2k-3} & b_{2k-4} & \ldots & b_{k+1} & b_k & b_{k-1}| & b_{k-2} & b_{k-3} & \ldots & b_1 & b_0 \\
 & a_{2k-1} & a_{2k-2} & a_{2k-3} & a_{2k-4} & \ldots & a_{k+1} & a_k| & a_{k-1} & a_{k-2} & a_{k-3} & \ldots & a_1 & a_0 \\
\hline
x_{2k} & x_{2k-1} & x_{2k-2} & x_{2k-3} & x_{2k-4} & \ldots & x_{k+1} & x_k| & x_{k-1} & x_{k-2} & x_{k-3} & \ldots & x_1 & x_0
\end{array}
$$

Folded Representation of Binary Numbers

- Problem: Binary squares $\mathcal{B}$ do not form regular language (Pumping lemma, NFAs cannot "remember" words of arbitrary length)
- Idea: Add high and low half of bits simultaneously
- Addition of higher bits depends on carry of lower bits
- Similar idea: Conditional Sum Adder from "TI"
- For this we use a more sophisticated, "folded" representation of binary numbers

Folded Representation of Binary Numbers

Our automaton gets pairs of bits, one of the higher and lower half each:

$$\Sigma = \{[h, l] \mid h, l \in \{0, 1\}\}$$

The "folding" mechanism can be seen in the following figure (the a_k are bits of an 9-bit integer, leading bit must be 1):

$$1a_7a_6a_5a_4|a_3a_2a_1a_0 \rightarrow \begin{pmatrix} 1 & \\ a_7 & a_3 \\ a_6 & a_2 \\ a_5 & a_1 \\ a_4 & a_0 \end{pmatrix} \rightarrow [a_4, a_0][a_5, a_1][a_6, a_2][a_7, a_3][1]_\zeta$$

$$1\ 1100\ 1001 \rightarrow \begin{pmatrix} 1 & \\ 1 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \rightarrow [0, 1][0, 0][1, 0][1, 1][1]_\zeta$$

Reversed order more logical when adding up numbers.

Folded Representation of Binary Numbers

- highest bit of odd-length number has no “folding partner” $\Rightarrow$ special character called $[1]_\zeta$
- Automata will need to know if we are near the end of the addition.
  - Pairs are annoated with letters $\alpha, \beta, \gamma, \delta, \epsilon$
  - ϵ means “last pair in even-length number or second-to-last in odd-length number”, other subscripts definied similarly
- This extends our language to

$$\Sigma = \{[1]_\zeta\} \cup (\{[h, l] \mid h, l \in \{0, 1\}\} \times \{\alpha, \beta, \gamma, \delta, \epsilon\})$$

Adding with NFAs

L_{M_1}

A binary number x of odd length $n = 2k + 1 \geq 13$ is in L_{M_1} iff x is the sum of two binary squares of length $n - 1$ and $n - 3$

- Basic setup for adding two numbers of length $n - 1 = 2k$ and $n - 3 = 2k - 2$
- "|" marks the middle of the numbers (rounded down in the odd length case)

$$
\begin{array}{cccccccccccccc}
 & & & b_{2k-3} & b_{2k-4} & \dots & b_{k+1} & b_k & b_{k-1}| & b_{k-2} & b_{k-3} & \dots & b_1 & b_0 \\
 & a_{2k-1} & a_{2k-2} & a_{2k-3} & a_{2k-4} & \dots & a_{k+1} & a_k| & a_{k-1} & a_{k-2} & a_{k-3} & \dots & a_1 & a_0 \\
\hline
x_{2k} & x_{2k-1} & x_{2k-2} & x_{2k-3} & x_{2k-4} & \dots & x_{k+1} & x_k| & x_{k-1} & x_{k-2} & x_{k-3} & \dots & x_1 & x_0
\end{array}
$$

Adding with NFAs

- Summands are binary squares $\rightarrow$ digits repeat
- First digit of each number must be 1 (definition of length)
- add carry at starting places

$$
\begin{array}{cccccccc|ccccc}
 & & & 1 & b_{k-3}\ldots & b_2 & b_1 & b_0 & 1 & b_{k-3}\ldots & b_1 & b_0 \\
 & 1 & a_{k-2} & a_{k-3} & a_{k-4}\ldots & a_1 & a_0 & 1 & a_{k-2} & a_{k-3}\ldots & a_1 & a_0 \\
c_{2k} & c_{2k-1} & c_{2k-2} & c_{2k-3} & c_{2k-4}\ldots & c_{k+1} & \mathbf{c_k} & c_{k-1} & c_{k-2} & c_{k-3}\ldots & c_1 & 0 \\
\hline
1_\zeta & x_{2k-1_\epsilon} & x_{2k-2_\delta} & x_{2k-3_\gamma} & x_{2k-4_\beta}\ldots & x_{k+1_\alpha} & x_{k_\alpha} & x_{k-1_\epsilon} & x_{k-2_\delta} & x_{k-3_\gamma}\ldots & x_{1_\alpha} & x_{0_\alpha} \\
 & & & & \text{upper half} & & & & & \text{lower half} & &
\end{array}
$$

Adding with NFAs: Creating the Transition Relation

$$
\begin{array}{cccccccc|ccccc}
 & & & 1 & b_{k-3}\dots & b_2 & b_1 & b_0 & & 1 & b_{k-3}\dots & b_1 & b_0 \\
 & 1 & a_{k-2} & a_{k-3} & a_{k-4}\dots & a_1 & a_0 & & 1 & a_{k-2} & a_{k-3}\dots & a_1 & a_0 \\
c_{2k} & c_{2k-1} & c_{2k-2} & c_{2k-3} & c_{2k-4}\dots & c_{k+1} & \mathbf{c_k} & c_{k-1} & & c_{k-2} & c_{k-3}\dots & c_1 & 0 \\
1_\zeta & x_{2k-1_\epsilon} & x_{2k-2_\delta} & x_{2k-3_\gamma} & x_{2k-4_\beta}\dots & x_{k+1_\alpha} & x_{k_\alpha} & x_{k-1_\epsilon} & & x_{k-2_\delta} & x_{k-3_\gamma}\dots & x_{1_\alpha} & x_{0_\alpha} \\
 & \multicolumn{6}{c}{\underbrace{\hphantom{xxxxxxxxxxxxxxxxxxxxxxxxxxxx}}_{\text{upper half}}} & \multicolumn{6}{c}{\underbrace{\hphantom{xxxxxxxxxxxxxxxxxxxxxx}}_{\text{lower half}}}
\end{array}
$$

- Start in initial state q_0
- Read $[x_k, x_0]_\alpha$ as input, “guess” b_0, b_1, a_0
- properties that have to be stored in state:
  - b_0 to be used later
  - b_1 to be used in next step
  - Carries c_l, c_h (c_1, c_{k+1}) for next step
  - Upper half carry c_k must be known for the first transition, is property of automaton (two separate automata for the two choices of c_k)
- next step has form (b_0, b_1, c_l, c_h) with $b_0, b_1, c_l, c_h \in \{0, 1\}$ (1 is highest possible carry when adding two binary numbers)

Adding with NFAs

$$
\begin{array}{ccccccccccccccc}
 & & & 1 & b_{k-3} & \dots & b_2 & b_1 & b_0| & 1 & b_{k-3} & \dots & b_1 & b_0 \\
 & 1 & a_{k-2} & a_{k-3} & a_{k-4} & \dots & a_1 & a_0| & 1 & a_{k-2} & a_{k-3} & \dots & a_1 & a_0 \\
c_{2k} & c_{2k-1} & c_{2k-2} & c_{2k-3} & c_{2k-4} & \dots & c_{k+1} & \mathbf{c_k} & c_{k-1} & c_{k-2} & c_{k-3} & \dots & c_1 & 0 \\
\hline
1_\zeta & x_{2k-1_\epsilon} & x_{2k-2_\delta} & x_{2k-3_\gamma} & x_{2k-4_\beta} & \dots & x_{k+1_\alpha} & x_{k_\alpha}| & x_{k-1_\epsilon} & x_{k-2_\delta} & x_{k-3_\gamma} & \dots & x_{1_\alpha} & x_{0_\alpha} \\
 & \multicolumn{7}{c}{\underbrace{\hspace{20em}}_{\text{upper half}}} & \multicolumn{6}{c}{\underbrace{\hspace{16em}}_{\text{lower half}}}
\end{array}
$$

- Transition from q_0 to state (b_0, b_1, c_l, c_h) on the character $[x_k, x_0]_\alpha$ is allowed (nondeterministic) iff for any $a_0 \in \{0, 1\}$ all of the following conditions hold:
  - $b_0 + a_0 = c_l \ x_0$ (seen as bit sequence)
  - $b_1 + a_0 + c_k = c_h \ x_k$

Adding with NFAs

Example Start in initial state q_0, assume automaton with $c_k = 0$. First input character is $[0,1]_\alpha$.

$$
\begin{array}{cccccccc|ccccc}
 & & & 1 & b_{k-3} \dots & b_2 & b_1 & b_0 & & 1 & b_{k-3} \dots & b_1 & b_0 \\
 & 1 & a_{k-2} & a_{k-3} & a_{k-4} \dots & a_1 & a_0 & & 1 & a_{k-2} & a_{k-3} \dots & a_1 & a_0 \\
c_{2k} & c_{2k-1} & c_{2k-2} & c_{2k-3} & c_{2k-4} \dots & c_{k+1} & \mathbf{0} & c_{k-1} & c_{k-2} & c_{k-3} \dots & & c_1 & 0 \\
\hline
1_\zeta & \underbrace{x_{2k-1_\epsilon} & x_{2k-2_\delta} & x_{2k-3_\gamma} & x_{2k-4_\beta} \dots & x_{k+1_\alpha} & 0_\alpha}_{\text{upper half}} & \underbrace{x_{k-1_\epsilon} & x_{k-2_\delta} & x_{k-3_\gamma} \dots & & x_{1_\alpha} & 1_\alpha}_{\text{lower half}}
\end{array}
$$

$\delta(q_0, [0,1]_\alpha) = \{$

b_0	b_1	a_0	c_l	c_h
0	0			
0	1			
1	0			
1	1			

- Similarly for $[0,0]_\alpha, [1,0]_\alpha, [1,1]_\alpha$
- Other subscripts do not occur in q_0 if numbers are long enough and correctly folded

Adding with NFAs

Example Start in initial state q_0, assume automaton with $c_k = 0$. First input character is $[0,1]_\alpha$.

$$
\begin{array}{ccccccc|cccccc}
 & & & 1 & b_{k-3} \dots & b_2 & b_1 & b_0 & 1 & b_{k-3} \dots & & b_1 & b_0 \\
 & 1 & a_{k-2} & a_{k-3} & a_{k-4} \dots & a_1 & a_0 & 1 & a_{k-2} & a_{k-3} \dots & & a_1 & a_0 \\
c_{2k} & c_{2k-1} & c_{2k-2} & c_{2k-3} & c_{2k-4} \dots & c_{k+1} & \mathbf{0} & c_{k-1} & c_{k-2} & c_{k-3} \dots & & c_1 & 0 \\
\hline
1_\zeta & x_{2k-1_\epsilon} & x_{2k-2_\delta} & x_{2k-3_\gamma} & x_{2k-4_\beta} \dots & x_{k+1_\alpha} & 0_\alpha & x_{k-1_\epsilon} & x_{k-2_\delta} & x_{k-3_\gamma} \dots & & x_{1_\alpha} & 1_\alpha \\
 & \multicolumn{6}{c|}{\underbrace{\hspace{20em}}_{\text{upper half}}} & \multicolumn{6}{c}{\underbrace{\hspace{18em}}_{\text{lower half}}}
\end{array}
$$

$$
\delta(q_0, [0,1]_\alpha) = \{(0,1,0,1), (1,0,0,0)\}
$$

$\dots$

b_0	b_1	a_0	c_l	c_h
0	0	x	x	x
0	1	1	0	1
1	0	0	0	0
1	1	x	x	x

- Similarly for $[0,0]_\alpha, [1,0]_\alpha, [1,1]_\alpha$
- Other subscripts do not occur in q_0 if numbers are long enough and correctly folded

Adding with NFAs

$$
\begin{array}{cccccccc|cccccc}
 & & & 1 & b_{k-3} \dots & b_2 & b_1 & b_0 & 1 & b_{k-3} \dots & & b_1 & b_0 \\
 & 1 & a_{k-2} & a_{k-3} & a_{k-4} \dots & a_1 & a_0 & 1 & a_{k-2} & a_{k-3} \dots & & a_1 & a_0 \\
c_{2k} & c_{2k-1} & c_{2k-2} & c_{2k-3} & c_{2k-4} \dots & c_{k+1} & \mathbf{0} & c_{k-1} & c_{k-2} & c_{k-3} \dots & & c_1 & 0 \\
\hline
1_\zeta & \multicolumn{6}{c|}{\underbrace{x_{2k-1_\epsilon} \; x_{2k-2_\delta} \; x_{2k-3_\gamma} \; x_{2k-4_\beta} \dots x_{k+1_\alpha} \; x_{k_\alpha}}_{\text{upper half}}} & \multicolumn{6}{c}{\underbrace{x_{k-1_\epsilon} \; x_{k-2_\delta} \; x_{k-3_\gamma} \dots x_{1_\alpha} \; x_{0_\alpha}}_{\text{lower half}}}
\end{array}
$$

- Rules for other states and inputs can be derived in a similar way, e.g.
- When reading $[u, v]_\epsilon$ we have to use the b_0 from the state tuple for the lower bits and in the upper half of the bits there is no b.
- We can only choose 1 for a.
- We have to make sure that we get $\mathbf{c_k}$ as a carry for the lower bits.

Putting together the automata

- Similar techniques are used to construct automata for remaining cases of main lemma (also for even-length numbers)
- The actual verification is done by the *ULTIMATE* framework developed at the chair for software engineering (University of Freiburg)
- Actual verification took less than one minute

Discussion of method

- Automata theory can deliver proofs where pure mathematicians did not suceed so far
- especially good for computational proofs (e.g. case distinctions with many cases like in our example)
- Critics: Computer does actual proving.
  - Hard to see and verify if working correctly
  - Hard to get intuition why proof works
- "Mechanical" proof better than no proof?
- Sometimes "elegant" proof is found some time after computational proof

Bibliography

P. Madhusudan, D. Nowotka, A. Rajasekaran, J. Shallit
Lagrange's Theorem for Binary Squares
ArXiv e-prints https://arxiv.org/abs/1710.04247