Special course in Computer Science: Molecular Computing

Lecture 4: Introduction to DNA Computing

Vladimir Rogojin
Department of IT, Åbo Akademi
http://combio.abo.fi/teaching/special

Fall 2013

DNA computing

- Basic idea:
  - Molecular biology processes can be used to perform arithmetic and logic operations on information encoded as DNA strands
- Principle:
  - DNA could be synthesized, copied, cut/fused, filtered, measured, sequenced
  - Due to the complementarity principle it could form complex “programmable” nano-structures

DNA computing

- We will study today:
  - 2 historically important DNA experiments:
    - First experimental demonstration of DNA computing – Adleman's celebrated experiment, searching for HPP in directed graph with DNA
    - First experiment demonstrating the potential of molecular computing to outperform unaided human computational ability
  - Influence of DNA-based computation on other areas of computer science:
    - Formal language theory, coding theory,

First DNA computing experiment

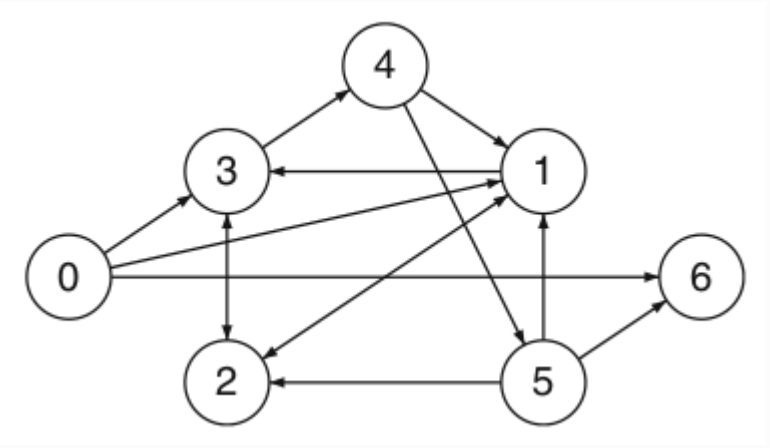
- Performed by Leonard Adleman in 1994, MIT
- Problem:
  - An instance of HPP with 7 nodes
- Hamiltonian path problem (HPP):
  - Search for a path (starting with some v_{start} and ending with some v_{end}) that contains all the verticies from the graph repeating exactly once
  - Known NP-complete problem

Solving HPP with DNA computing

- The algorithm:
  - *Input:* A directed graph *G* with *n* vertices and designated vertices v_{start} and v_{end}
  - *Step1:* Generate random paths through the graph
  - *Step2:* Keep only those paths that begin with v_{start} and end with v_{end}
  - *Step3:* Keep only those paths that enter exactly *n* vertices
  - *Step4:* Keep only those paths that enter all of the vertices of the graph at least once
  - *Output:* If any paths remain, output “YES”; otherwise output “NO”

Solving HPP with DNA computing

- Bio-algorithm:
  - Encoding the input:
    - Vertex → 20-mer single strand of DNA
    - Directed edge → second half of the sequence encoding the source edge followed by the first half of the sequence encoding the destination edge



- Exceptions:
- Edges starting with v_{start} → sequence representing v_{start} followed by the first half representing the target vertex

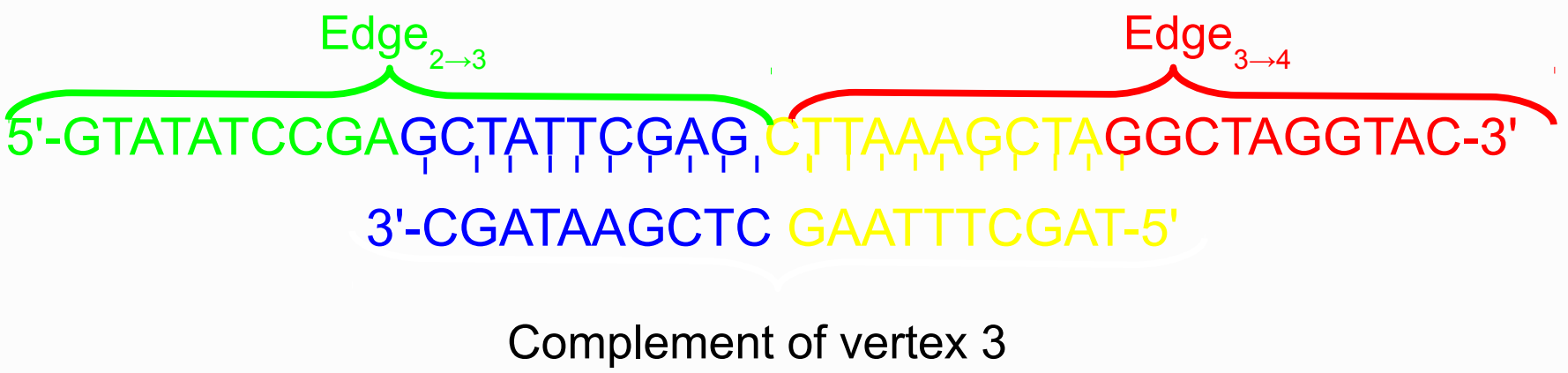
Solving HPP with DNA computing

- Example:
  - DNA sequences for the vertex 3 and the directed edges 2 → 3 and 3 → 4 encoded as:
    - O_3 = 5'-GCTATTCGAGCTTAAAGCTA-3'
    - $O_{2\to3}$ = 5'-GTATATCCGAGCTATTCGAG-3'
    - $O_{3\to4}$ = 5'-CTTAAAGCTAGGCTAGGTAC-3'

Solving HPP with DNA computing

- Implementing *Step1*:
  - Amplify with PCR complements $\overline{O}_i$
  - Amplify with PCR single strands $O_{i \to j}$
  - Hybridize copies of $\overline{O}_i$ with copies of $O_{j \to i}$ and $O_{i \to k}$

Solving HPP with DNA computing

- Example:
- 

- *Result:* formation of DNA molecules encoding random paths through *G*

Solving HPP with DNA computing

- Implementing *Step2*:
  - keep only paths that start with v_{start} and end with v_{end}
  - Amplify the product of Step1 with PCR and primers $\overline{O}_0$ and $\overline{O}_6$
  - Result: molecules encoding paths starting with 0 and ending with 6 are amplified

Solving HPP with DNA computing

- Implementing *Step3*:
  - keep only paths of the correct length
  - By use of gel electrophoresis separate DNA strands by their length
  - Filtering criteria: retain only molecules corresponding to paths of length 7,
  - i.e., retain 20*7=140 bp molecules

Solving HPP with DNA computing

- Implementing *Step4*:
  - keep only paths that pass through each vertex at least once
  - Filtering by affinity purification:
    - Generate single-stranded DNA from the product of *Step3*
    - For each vertex *i*:
      - Attach $\overline{O}_i$ to magnetic bead
    - Result:
      - The retained DNAs encode paths containing vertex *i*
      - By repeating successively the process with $\overline{O}_1$, $\overline{O}_2$, $\overline{O}_3$, $\overline{O}_4$ and $\overline{O}_5$, one gets molecules encoding paths containing all the verticies

Solving HPP with DNA computing

- Implementing the *Output*:
  - are there any paths left?
  - The presence of a molecule encoding for a Hamiltonian path to be checked:
    - Amplify the result of Step4 by PCR with primers O_0 and $\overline{O}_6$
    - *Output:* Sequence the result

Discussion

- The entire computation required approximately 7 days of wet lab work
- It was the first proof-of-concept experiment that DNA computation was possible
- Advantages:
  - *Step1* is cause of time-complexity exponential explosion:
    - One time-step with Adleman's algorithm
- Disadvantages:
  - Time and cost of the experiment
  - Exponential explosion for the volume of DNA material:
    - e.g., for 200 nodes one would need more than the Earth's weight of DNA

Milestone experiment demonstrating the potential of DNA computing

- In 2002 it was demonstrated that DNA computing devices can exceed computational capabilities of an unaided human (Braich et al. 2002)
- Solved with DNA 20-variable 3-SAT problem with 24 clauses
- Exhaustive search in solution space of size 2^{20}

3-SAT problem

- Input:
  - Boolean formula in three-conjunctive-normal-form (3-CNF)
    - Conjunction of disjunctive clauses, where each clause is the disjunction of at most 3 literals
    - A literal – a boolean variable or its negation
    - Example: $(X_1 \mid \overline{X}_2 \mid \overline{X}_3) \,\&\, (\overline{X}_4 \mid X_5 \mid \overline{X}_6)$
  - The boolean formula is *satisfiable* if there exists a truth assignment what makes the formula *true*
- Output:
  - yes, if there exists variable truth value assignment satisfying the input boolean formula

Solving 3-SAT

- Nondeterministic algorithm:
  - Input:
    - A boolean formula *F* in *3-CNF*
  - *Step1:* Generate the set of all possible truth value assignments
  - *Step2:* Remove the set of truth value assignments that make the first clause false
  - *Step3:* Repeat Step 2 for all the clauses of the input formula
  - Output:
    - The remaining (if any) truth value assignments

Solving 3-SAT with DNA

- Encode variable:
  - For every variable X_k two 15-mer strands:
    - 1 strand X^T_k associated with *true* assignment of X_k
    - Another strand X^F_k associated with *false* assignment of X_k
- Generate library of all truth assignments by using mix-and-match combinatorial synthesis technique. In total all 2^{20} different assignments were generated – 300-mer DNA strand

Solving 3-SAT with DNA

- Filtering truth assignments satisfying the boolean formula *F*:
  - A truth assignment: $X_1X_2\ldots X_{20}$ where Xi either is equal to X^T_k or to X^f_k
  - 25 glass modules:
    - 0 glass module – initial library of all assignments
    - *i*'th glass module contains probes hybridizing to molecules representing the assignments satisfying clause *I*
    - By applying consecutively filtering from modules 1, 2, …, 24 to the library from the module 0 one gets molecules representing the truth assignments satisfying *F*

Discussion

- Observation:
  - Only Watson-Crick complementarity-based annealing and melting was used to compute
  - Brute-force search: 60-70 variables
  - Breadth-first search algorithm: limit estimated 120 variables

Discussion

- These two experiments are historically significant instances in the area of DNA computing experiments
- Potential applications:
  - Nanorobotics, nanocomputing, bioengineering, bio-nanotechnology, and micromedicine

Formal language theory/Codding theory and DNA Computing

- It is natural to investigate DNA/ RNA and their interactions in the framework of formal language and codding theory
- One can abstract from all the biochemical properties and reactions related to DNA/RNA and focus solely on the DNA sequences

Formal language theory/Codding theory and DNA Computing

- Fundamental difference between electronic information and DNA information:
  - Electronic-based information has a fixed address, freely accessible and is reusable
  - DNA-encoded info is accessible in less controllable manner and once used in bio-operations becomes unavailable
- Problems with DNA-based information:
  - Undesired/unplanned hybridization between partially complementary DNA sequences is possible

Formal language theory/Codding theory and DNA Computing

- Standard biological methods assessing hybridization between DNA sequences rely on thermodynamical parameters:
  - For instance, the *free energy (energy needed to melt DNA)*
  - And melting temperature of DNA
- For calculation of these properties one needs:
  - Not only WK complementarity between single bases, bu also WK complementarity of neighboring bases with their counterparts
- Approximate characterization of DNA hybridization:
  - WK complementarity
  - Other similarity measures

Encoding of information with DNA

- Designing a set of “good” DNA strands:
  - *Positive* design problem:
    - Design a set of input DNA molecules such that there is a sequence of reactions that produces the correct result.
  - *Negative* design problem:
    - Design a set of input DNA molecules that do not interact in undesirable ways
      - i.e., do not produce incorrect outputs, and
      - do not consume molecules necessary for other “programmed” interactions
      -

Encoding of information with DNA

- The *positive* design problem is highly related to a specific experiment and it is hard to find a generic framework
- The *negative* design problem can be solved on general basis – construct a library of molecules not allowing for undesired mutual hybridization:
  - no strand forms any undesired secondary structure such as hairpin loops
  - no string in the library hybridizes with any string in the library
  - no string in the library hybridizes with the complement of any string in the library

Encoding of information with DNA

- Uniqueness of the oligonucleotides:
  - individual oligonucleotides in a mixture differ substantially from each other
  - Goal: the nucleotides and any longer sequences containing the nucleotides should be easily distinguishable
  - In math terms those are codes
- DNA codewords
  - sets of unique oligonucleotides of a fixed length

Encoding of information with DNA

- DNA encoding design:
  - Thermodynamical methods provide the most precise results but are computationally the most expensive
  - The opposite approach: WK complementarity allows for fastest but least-precise methods
  - Approximation methods: capture key aspects of the nearest neighbor thermodynamic model
    - This is an intermediate step between these two methodologies
  - Various discrete metrics based often on Hamming or Levenshtein distance have been studied

Formalizing DNA

- Single-stranded DNA molecules →
  - strings over the DNA alphabet $D = \{A, C, T, G\}$
- Reactions on DNA →
  - Formal manipulation on the respective strings
- An alphabet:
  - A finite non-empty set of symbols
- A word:
  - A sequence of symbols from the alphabet

Hamming distance

- Hamming distance between two equal-length strings:
  - the number of positions at which the corresponding symbols are different
- Example:
  - "toned" and "roses":
    - 3
  - 1011101 and 1001001:
    - 2
  - 2173896 and 2233796:
    - 3

Levenshtein distance

- Levenshtein (edit) distance between two strings:
  - the minimum number of single-character edits (insertion, deletion, substitution) required to change one word into the other
- Example:
  - "kitten" and "sitting":
    - 3
    - **k**itten → **s**itten
    - sitt**e**n → sitt**i**n
    - sittin → sittin**g**

Intramolecular bonds in DNA

- Hairpin:
  - Self-hybridized DNA

- Hairpin-like secondary structures play important role in:
  - insertion/deletion operations with DNA
  - Whiplash PCR computing techniques
  - DNA RAM
  - *In vivo*: one of intramolecular operations in gene assembly process in ciliates

Intramolecular bonds in DNA

- Hairpin-freeness is crucial in the design of primers for the PCR reaction

Some elements of formal languages

- Let Σ be an alphabet:
  - Set of all words over Σ is denoted by Σ^*
  - Set of all non-0-length words over Σ is Σ^+
    - $\Sigma^+=\Sigma^*/\lambda$, where λ is empty word
  - Set of all words of length k over Σ is Σ^k
- The length of a word w is denoted by $|w|$
- Number of occurrences of a nonempty subword x in word w is denoted by $|w|_x$

Some elements of formal languages

- Let $w=xvy$, then v is a subword of a word w
- $Sub(w)$ – the set of all subwords of w
- $Sub(w)_k$ – the set of all subwords of length k in w
- u prefix of w, if $w=ut$. Denoted as $u \leq w$
- u and w are prefix comparable, if either of them is a prefix of the other one. Denoted as $u \sim_p w$

Some elements of formal languages

- *u* suffix of *w* if *w*=*su*.
- *Pref(w)* – set of all prefixes of *w*
- *Suff(w)* – set of all suffixes of *w*
- *Embedding order* over *u* and *w*: $u \leq_e w$
  - $u = u_1 u_2 \ldots u_n$; $w = v_1 u_1 v_2 u_2 \ldots v_n u_n v_{n+1}$
  - Where u_i and v_i are words in Σ^*

Some elements of formal languages

- Languages:
  - A language L over Σ is a set of words over Σ^*
  - L^n is a set of words $w=w_1w_2 \dots w_n$, where w_i is a word from language L
  - $L^{\geq n}$ is a set of words $w=w_1w_2 \dots w_k$, where w_i is a word from language L and $k \geq n$
  - $L^*=L^0UL^1UL^2U \dots$, and $L^+= L^*/\{\lambda\}$
  - $Sub(L)$ – the set of all subwords of L

Formalizing DNA computing

- Tube languages:
  - Formalizes set of molecules in a test tube
  - Tube language *L* is equal to, or a subset of K^+, where *k* is a finite language whose elements are called *codewords*
  - In majority cases *K* consists of words of some fixed length *l*, i.e., *K* – a *code of length l*

Some elements of formal languages

- Morphism:
  - A mapping α: $\alpha(uv)=\alpha(u)\alpha(v)$
- Antimorphism:
  - A mapping α: $\alpha(uv)=\alpha(v)\alpha(u)$
- Involution:
  - A mapping θ from Σ to Σ such that $\theta(\theta(x))=x$
  - $\theta=\theta^{-1}$

Formalizing DNA computing

- Palindrome:
  - A word *w* that is equal to its inverse w^R
  - Example: $1234321=(1234321)^R=1234321$
- Formalizing DNA:
  - DNA alphabet: $\Delta=\{A,C,G,T\}$
  - Involution τ: $\tau(A)=T$, $\tau(T)=A$, $\tau(C)=G$, $\tau(G)=C$
  - Extending τ to antimorphism:
    - $\tau(w)=\overline{w}$. T transfroms w strand into its complement strand

Intramolecular bonds in DNA

- Formalizing hairpin DNA structure:
  - θ morphic or antimorphic involution of Σ^*
  - k – a positive integer
  - A word u in Σ^* is said to be θ-k-hairpin-free if u=xvyθ(v)z for some words x, v, y, z implies $|v|<k$
- Intuitively:
  - The condition above describes a hairpin structure whose stem v consists of at least k base pairs

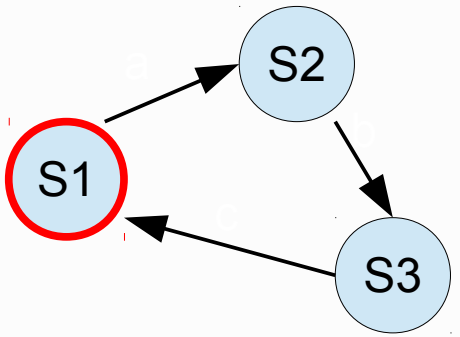
Family of regular languages

- Can be expressed by using regular expression
  - Specifies a set of strings through a pattern
  - Example: (a|b|c)* - any sequence of a,b,c letters
- Can be recognized by finite automation
  - Set of states and conditional transitions between them
  - Represented by directed graph:
    - Nodes – states
    - Edges – transitions. Labels define when a transition fires

Finite automation

- S1(a) → S2
- S2(b) → S3
- S3(c) → S1
- Initial state: S1
- Terminal state: S1
- Recognizes: (abc)*

- S1(abcabc) → S2(bcabc) →
- S3(cabc) → S1(abc) → S2(bc) → S3(c) → S1 – ACCEPT!
-
- S1(abb) → S2(bb) → S3(b) – REJECT!



Deterministic and nondeterministic FA

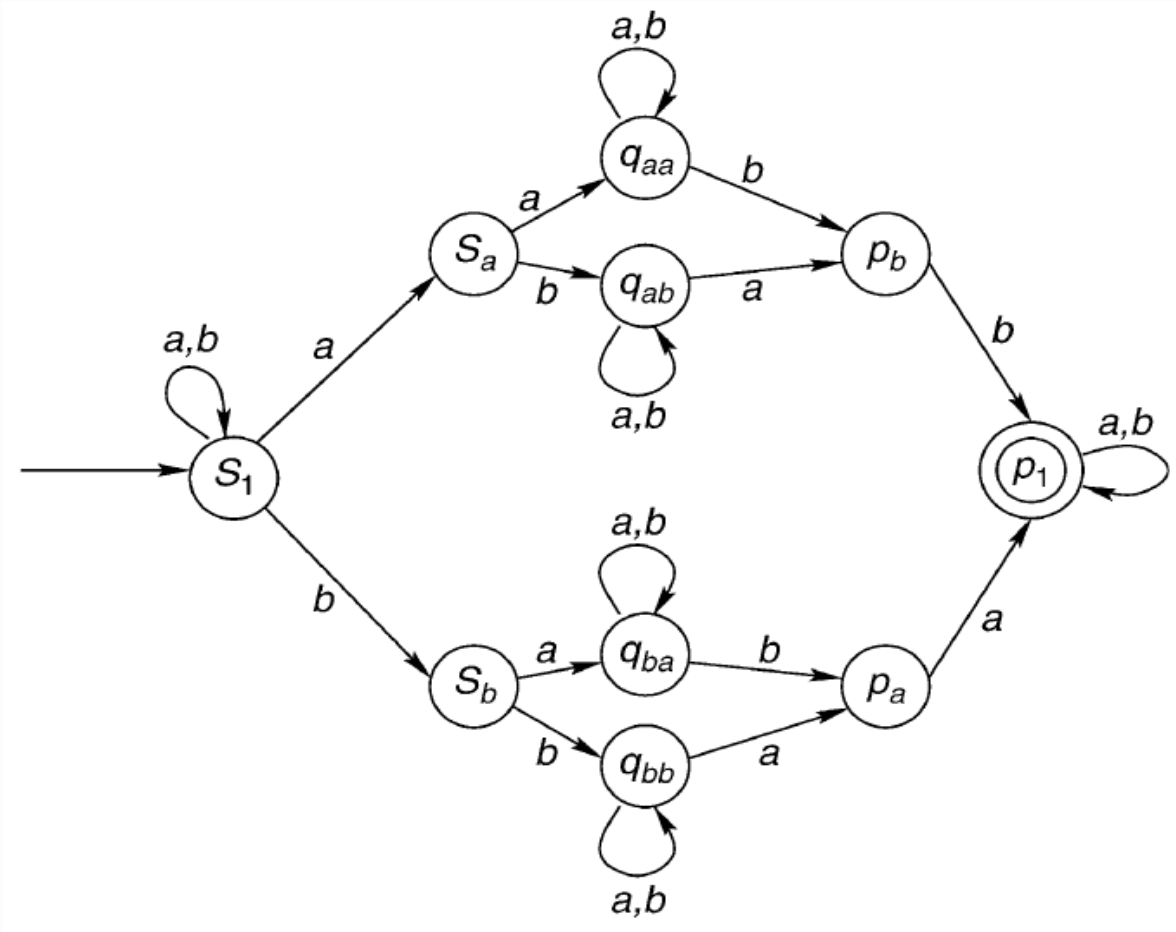
- Deterministic FA (DFA):
  - For each letter *a* and state *S* there is only one transition $S(a) \rightarrow S'$
  - i.e., there is only one possible computation for given input
  -
  - DFA ≡ NFA
- Nondeterministic FA (NFA):
  - For each letter a and state *S* there may be many different transitions: $S(a) \rightarrow S'$, or $S(a) \rightarrow S''$, or $S(a) \rightarrow S'''$, etc.
  - A transition is chosen nondeterministically
  - For a given input there may exist many different computations

Hairpin-free words and languages

- Set of θ-k-hairpin-free words we denote as *hpf(θ,k)*
- A language *L* is θ-k-hairpin-free if *L* is a subset of *hpf(θ,k)*
- Set of all hairpin words over Σ is complement to *hpf(θ,k)* and is denoted as *hp(θ,k)*
- Theorem1: The languages *hp(θ,k)* and *hpf(θ,k)* are regular

Hairpin-free words and languages

- An NFA accepting the language *hp(θ,2)* over the alphabet *{a,b}*, where the antimorphism isdefined as *θ(a)=b* and *θ(b)=a*.
-



Context-free language

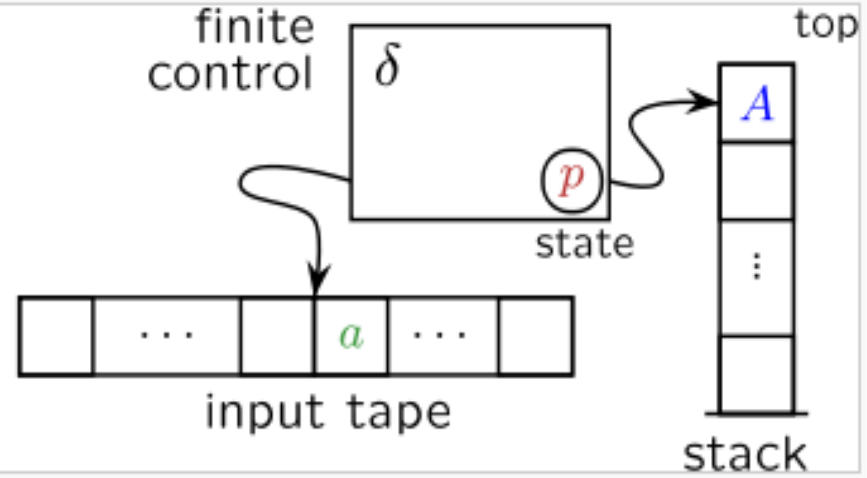
- Generated by context-free grammar:
  - Defined by set of rules of form $V \rightarrow w$,
    - Where V – is a nonterminal, and w – a string of terminals and non-terminals
    - Starting from start symbol S, by applying a sequence of CF rules a word of terminals only is obtained
    - The set of such words is the language generated by the CF grammar
    - Example:
      - Rules $S \rightarrow (S)$, $S \rightarrow \lambda$
      - Generate words ((((...)))) with equal number of opening and closing parentheses

Context-free language

- Recognized by nondeterministic pushdown automata:
  - Difference from a FA: equipped with stack:
    - Stack: a variable length vector where only the top-most element is accessible
      - Operations on the top element:
        - Push
        - Pop
        - Read
  - Transition is chosen basing on:
    - Incoming signal
    - Current state
    - Top-most symbol in the stack

Pushdown automata

- Transition is chosen basing on:
  - Incoming signal
  - Current state
  - Top-most symbol in the stack
- During transition:
  - New state is chosen
  - Optionally: a modification on top of the stack is performed (either push a symbol or pop)
- Nondeterministic PDA recognize all CF languages



a diagram of the pushdown automaton -Wikipedia

Hairpin-free words and languages

- The following problem is decidable in linear (or cubic, respectively) time with respect to size of M, where:
  - *Input*: A nondeterministic regular (pushdown, respectively) automaton *M*
  - *Output*: Yes/No depending on whether *L(M)* is *hpf(θ,k)*

Hairpin-free words and languages

- The maximality problem of hairpin-free languages:
  - Can a given hairpin-free language be extended and still remain hairpin-free
  - A hairpin-free language if maximal if it does not have any nontrivial superset being a hairpin-free language

Hairpin-free words and languages

- The following problem is decidable in time $O(|M_1|*|M_2|)$ (or $O(|M_1|*|M_2|^3)$, respectively):
  - *Input*: a DFA (PDA, respectively) automation M_1 accepting an $hp(\theta,k)$-free language, and an NFA M_2
  - *Output*: *Yes/No* depending on whether there is a word w from the language accepted by M_2 and not accepted by M_1 such that extending $L(M_1)$ with w is an $hp(\theta,k)$-free language

Hairpin-free words and languages

- Optimal negative design problem:
  - Construct a set of hairpin-free DNA words of a given size,
  - Where words can be chosen from certain library
- All hairpin-free sets are subsets of *hpf(θ,k).*
- By Theorem1 *hpf(θ,k)* can be accepted by a FA
- The size of the FA growth exponentially with respect to *k*

Hairpin-free words and languages

- Scattered hairpins:
  - Definition2: Let θ be an involution of Σ^* and let k be a positive integer. A word $u=wy$ is θ-k-scattered-hairpin-free ($shp(\theta,k)$-free) if for all words t, $t \leq_e w$ and $\theta(t) \leq_e y$ we have $|t|<k$

Hairpin-free words and languages

- Hairpin frames:
  - The pair $(v,\theta(v))$ in a word u of the form $u=xvy\theta(v)z$ is called an hp-pair of u. The sequence of hp-pairs $(v_1,\theta(v_1)), (v_2,\theta(v_2)), \ldots, (v_j,\theta(v_j))$ of the word u in the form:
    - $u=x_1v_1y_1\theta(v_1)z_1x_2v_2y_2\theta(v_2)z_2 \ldots x_jv_jy_j\theta(v_j)z_j$
  - Is called an hp-frame of degree j of u

Hairpin-free words and languages