Intersection Non-Emptiness for Tree Shaped Finite Automata

Michael Wehar
University at Buffalo

A&C Seminar - Waterloo

November 3, 2015
Overview

1. Classic Problem
2. Adding a Stack
3. Restricted Classes of DFA’s
4. Fundamental Connections
5. Complexity of Tree Shaped DFA’s
6. Fine Grained Hardness Results
Intersection Non-Emptiness for DFA’s

Given a finite list of DFA’s, is there a string that simultaneously satisfies all the DFA’s?

- We denote this problem by $IE_D$.
- We use $n$ for the total length of the input’s encoding.
- We use $k$ for the number of DFA’s in the list.
Intersection Non-Emptiness for DFA’s

Given a finite list of DFA’s, is there a string that simultaneously satisfies all the DFA’s?

- Let DFA’s $D_1, D_2, \ldots, D_k$ be given.
- Construct the product DFA $D$.
- If each DFA has at most $m$ states, then the product has at most $m^k$ states.
- $D$ accepts a string $x$ $\iff$ $D_i$ accepts $x$ for each $i \in [k]$.
- We just need to check if $D$ is satisfiable.
The General Problem

Intersection Non-Emptiness for DFA’s

Given a finite list of DFA’s, is there a string that simultaneously satisfies all the DFA’s?

- It is a classic \textit{PSPACE}-complete problem [Kozen 77].
  - Directed Reachability: Can I get from a start state to a final state in the product graph?
  - Constraint Satisfaction: Is there a string that satisfies all of the DFA’s?
- We use $k$-$\text{IE}_D$ to denote the problem for fixed $k$-many DFA’s.
Fixed Parameter Problem

**Theorem**
Solving $k$-$\text{IE}_D$ is equivalent to simulating a NTM that uses $k \log(n)$ bits of memory.

- Let a $k \log(n)$-space bound NTM $M$ be given.
- Let an input string $x$ of length $n$ be given.
- A computation of $M$ on $x$ is a sequence of configurations.
- Each configuration includes the tape content.
Theorem

Solving $k$-$\text{IE}_D$ is equivalent to simulating a NTM that uses $k \log(n)$ bits of memory.

- The tape is a sequence of $k \log(n)$ bits.
- We can break up this sequence into $k$ regions.
- Each region will consist of $\log(n)$ bits from the tape.
- We build $k$ DFA’s to collectively verify a “computation”.
- We assign one DFA to each region.
Overview

1. Classic Problem

2. Adding a Stack

3. Restricted Classes of DFA’s

4. Fundamental Connections

5. Complexity of Tree Shaped DFA’s

6. Fine Grained Hardness Results

Michael Wehar
Tree Shaped Finite Automata
Intersection Non-Emptiness for DFA’s and One PDA

Given a finite list of DFA’s and one PDA, is there a string that simultaneously satisfies all of the automata?

- We denote this problem by \( \text{IE}_P \).
- We use \( n \) for the total length of the input’s encoding.
- We use \( k \) for the number of DFA’s in the list.
Intersection Non-Emptiness for DFA’s and One PDA

Given a finite list of DFA’s and one PDA, is there a string that simultaneously satisfies all of the automata?

- We can solve $IE_P$ by building a product PDA.
- This variation of the problem is EXPTIME-complete.
- We use $k$-$IE_P$ to denote the problem for fixed $k$-many DFA’s.
We described how to build $k$ DFA's that verify a $k \log(n)$-space bounded NTM's computation.

We will show that with $k$ DFA's and one PDA, we can verify a $k \log(n)$-space bounded AuxPDA's computation.

An auxiliary PDA has a two-way read only input tape, a stack, and a two-way read/write auxiliary binary work tape.
Theorem

Solving $k$-IE$_P$ is equivalent to simulating an AuxPDA that uses $k \log(n)$ bits of memory.

- The reduction is essentially the same.
- We build $k$ DFA’s and one PDA to collectively verify an AuxPDA “computation”.
- The auxiliary work tape is split up into $k$ regions.
- We assign one DFA to each region.
- The single PDA is used to keep track of the stack.
Deterministic Polynomial Time is equivalent to Auxiliary Logspace [Cook 71].

\[ P = \text{AuxL}. \]

A more careful look reveals that \( n^k \)-time bounded DTM's are essentially equivalent to \( k \log(n) \)-space bounded AuxPDA's.

Therefore, solving \( k\text{-IE}_\mathcal{P} \) is equivalent to simulating a DTM that runs for at most \( n^k \) time.
Solving $k$-IE$_D$ is equivalent to simulating a NTM that uses $k \log(n)$ bits of memory.

$\exists c_1 \forall k \; k$-IE$_D \notin \text{NSPACE}(c_1k \log(n))$

Solving $k$-IE$_P$ is equivalent to simulating a DTM that runs for at most $n^k$ time.

$\exists c_2 \forall k \; k$-IE$_P \notin \text{DTIME}(n^{c_2k})$
Let’s make the problem easier by looking at restricted classes of DFA’s.

We only want classes of DFA’s that are closed under the product construction.

Consider the following restriction examples:
  - Graph Structure: DFA’s with an acyclic state diagram.
  - Algebraic Structure: DFA’s with a commutative transition monoid.
Definition

A Tree Shaped DFA is a DFA whose state diagram forms a tree (ignoring the dead state).

- Tree shaped DFA’s have a root, a height, and they only accept finite languages.
- Balanced if the tree is balanced and the final states are exactly the leaves of the tree.
- Tree shaped DFA’s and balanced tree shaped DFA’s are both closed under products.
Notice that the DFA above accepts the finite language:
\{0, 11, 000, 001, 1001, 1011\}.

We can simply represent this language by: \{0, 11, 00*, 10*1\}.
Overview

1. Classic Problem
2. Adding a Stack
3. Restricted Classes of DFA’s
4. Fundamental Connections
5. Complexity of Tree Shaped DFA’s
6. Fine Grained Hardness Results
The following are equivalent:

- Intersection Non-Emptiness for $k$ Balanced Tree Shaped DFA's
- $\bigwedge_k \bigvee \land$-SAT
- Boolean Query Join for $k$ (incomplete) tables
- $k$-Clique
Balanced Tree Shaped DFA: Equivalence

Height of 3 with branches: \{0*0, 10*, 111\}.
$\land_k \lor \land$-SAT: Equivalence

- $\land \lor$-SAT is the same as CNF-SAT.
- An instance of $\land_k \lor \land$-SAT looks like:

\[
\land \lor \land_{a,b,c}^{\ell_{a,b,c}}
\]

\[
a \in [k] \quad b \in [n] \quad c \in [n]
\]

- Example $\lor \land$-clause corresponding to $\{0^*0, 10^*, 111\}$:

\[
(\neg v_1 \land \neg v_3) \lor (v_1 \land \neg v_2) \lor (v_1 \land v_2 \land v_3)
\]
Boolean Query Join: Equivalence

Problem

Given $k$ boolean tables with incomplete partial data, is there a non-empty join?

- Example table corresponding to \{0\ast0, 10\ast, 111\}:

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- An asterisk represents an incomplete data entry.
Notice from the previous examples that:

- Balanced Tree DFA = $\lor \land$-Clause = Boolean Table.
- Final State = $\land$-Subclause = Row.
- Height = Variables = Attributes.

Let $k$, $c$, and $h$ denote any fixed natural numbers.

The following are equivalent:

- Intersection Non-Emptiness for $k$ Balanced Tree Shaped DFA’s with $c$ final states and height $h$.
- $\land_k \lor_c \land$-SAT with at most $h$ variables.
- Boolean Query Join for $k$ tables with $c$ rows and $h$ attributes.
Overview

1. Classic Problem
2. Adding a Stack
3. Restricted Classes of DFA’s
4. Fundamental Connections
5. Complexity of Tree Shaped DFA’s
6. Fine Grained Hardness Results

Michael Wehar
Tree Shaped Finite Automata
Theorem

IE for Tree Shaped DFA’s is NP-complete even when each DFA has at most 3 finals states.

- A witness is any string that is in the intersection.
- The intersection problem is in NP because the witness length is linear in the number of states.
- Hardness follows by a reduction from 3-SAT.
- For a given 3-SAT formula, we can construction a tree shaped DFA for each clause.
Consider a clause \((v_1 \lor \neg v_2 \lor v_4)\) from a 3-SAT instance.

The following DFA is associated with the clause:
Theorem

IE for $k$ Tree Shaped DFA’s is solvable in $O(k \cdot n)$ time with $k \log(n)$ non-deterministic bits.

- Guess a final state for each Tree Shaped DFA.
- This requires $k \log(n)$ non-deterministic bits.
- Each final state determines a branch in the respective DFA.
- For each bit position $i$, make sure that no two branches mismatch on the $i$th bit.
Reduction to $k$-Clique

**Theorem**

IE for $k$ Tree Shaped DFA’s is efficiently reducible to $k$-Clique.

- Let $k$ Tree Shaped DFA’s with $n$ states each be given.
- Form a graph $G$ with $O(n \cdot k)$ vertices such that each tree branch denotes a vertex.
- There is an edge connecting branches $b_i$ and $b_j$ if:
  - $b_i$ and $b_j$ come from different DFA’s
  - $b_i$ and $b_j$ have no bit mismatches.
- A $k$-clique in $G$ represents a valid choice of $k$ branches where there are no mismatches.
Reduction to $k$-Hyperclique

Theorem

IE for $k$ Tree Shaped DFA’s with input alphabet $[c]$ is reducible to $c$-uniform $k$-Hyperclique.

- We construct a $c$-uniform hypergraph $H$.
- The vertices of $H$ are similarly tree branches.
- A group of $c$-many branches forms a hyperedge if:
  - no two branches come from the same DFA
  - at each character position, the $c$-ary intersection of possible characters from each branch is non-empty.
- A $k$-hyperclique in $H$ represents a valid choice of $k$ branches where there are no $c$-ary mismatches.
The intersection problem is \( \text{NP} \)-complete even when each Tree Shaped DFA’s has at most 3 final states.

For \( k \) Tree Shaped DFA’s, we can solve intersection non-emptiness in linear time with limited non-determinism.

There is an efficient reduction to \( k \)-Clique.

Using a known approach for \( k \)-Clique, we can solve the intersection problem in \( O(n^{0.792^k}) \) time.

Further, there is a connection between larger alphabets and higher dimensional graphs.
Overview

1. Classic Problem
2. Adding a Stack
3. Restricted Classes of DFA’s
4. Fundamental Connections
5. Complexity of Tree Shaped DFA’s
6. Fine Grained Hardness Results
Theorem

If IE for 2 Tree Shaped DFA's is solvable in $O(n^{2-\epsilon})$ time, then CNF-SAT is solvable in $O(\text{poly}(n) \cdot 2^{(1-\frac{\epsilon}{2})n})$ time.

- We will define a special kind of reduction from CNF-SAT to Intersection Non-Emptiness.
- Let a CNF formula $\phi$ with $n$ variables and $m$ clauses be given.
- We construct Tree Shaped DFA's $D_1$ and $D_2$ such that $\phi$ is satisfiable if and only if $L(D_1) \cap L(D_2) \neq \emptyset$.
- Each DFA has $O((m + n) \cdot 2^{\frac{n}{2}})$ states.
Theorem

If IE for 2 Tree Shaped DFA’s is solvable in $O(n^{2-\epsilon})$ time, then CNF-SAT is solvable in $O(poly(n) \cdot 2^{(1-\frac{\epsilon}{2})n})$ time.

- A variable assignment for $\phi$ is a bit string of length $n$.
- This string can be broken up into two blocks of $\frac{n}{2}$ bits each.
- Each clause $c_i$ is assigned a clause bit $b_i$.
- A clause bit $b_i$ is valid if either:
  - $b_i = 0$ and block 1 forces $c_i$ to be satisfied
  - $b_i = 1$ and block 2 forces $c_i$ to be satisfied.
Theorem

If $IE$ for 2 Tree Shaped DFA’s is solvable in $O(n^{2-\epsilon})$ time, then CNF-SAT is solvable in $O(poly(n) \cdot 2^{(1-\frac{\epsilon}{2})n})$ time.

- The DFA’s read in block 1 and block 2 of a variable assignment followed by a string of $m$ clause bits.
- $D_1$ branches for block 1 and $D_2$ branches for block 2.
- $D_1$ verifies that for each $i$, if $b_i = 0$, then block 1 satisfies $c_i$.
- $D_2$ verifies that for each $i$, if $b_i = 1$, then block 2 satisfies $c_i$.
- Together, $D_1$ and $D_2$ verify that each clause bit is valid and hence each clause is satisfied by some block.
Hardness for $k$ Tree Shaped DFA’s

Theorem

For any fixed $k$, if IE for $k$ Tree Shaped DFA’s with alphabet $[k]$ is solvable in $O(n^{k-\epsilon})$ time, then CNF-SAT is solvable more quickly.

- Let a CNF formula $\phi$ with $n$ variables and $m$ clauses be given.
- We construct Tree Shaped DFA’s $\{D_i\}_{i \in [k]}$ with input alphabet $[k]$ such that:
  - $\phi$ is satisfiable if and only if $\bigcap_{i \in [k]} L(D_i) \neq \emptyset$.
- Each DFA has $O((m + n) \cdot 2^{\frac{n}{k}})$ states.
Theorem

For any fixed $k$, if IE for $k$ Tree Shaped DFA’s with alphabet $[k]$ is solvable in $O(n^{k-\epsilon})$ time, then CNF-SAT is solvable more quickly.

- Variable assignments are broken up into $k$-many blocks.
- Each DFA is assigned a different block.
- Each clause $c_j$ is assigned a clause character $b_j$ from $[k]$.
- $D_i$ verifies that for each $j$, if $b_j = i$, then block $i$ satisfies $c_j$.
- Notice that the transitions of $D_i$ are labelled with many different alphabet characters from $[k]$. 
If we can beat quadratic time for 2 Tree Shaped DFA’s, then the Strong Exponential Time Hypothesis (SETH) is false.

If for some fixed $k$, we can beat $n^k$ time for $k$ Tree Shaped DFA’s with alphabet $[k]$, then SETH is false.

For each $c$ and $k$, there is a more complex reduction:

- **Input:** A $c$-SAT instance of size $n$.
- **Output:** Roughly, $k \cdot \log(c)$-many Tree DFA’s (binary input) with $2^{n/k}$ states each.

Therefore, if we can solve IE in $n^{o(k)}$ time, then ETH is false.