1 Review of DFAs

We begin with a review of some basic facts about DFA's.

Given $p_1, \ldots, p_n \in$ Prop we will view $2^{\text{Prop}} = \{0,1\}^n$. Let $\Sigma = \{0,1\}$. Then, given $\phi \in$ Form, $models(\phi) \subseteq \Sigma^n$.

Through DFA construction we define a mapping $\phi \rightarrowtail A_\phi$ where $models(\phi) = L(A_\phi)$.

Let's review the desiderata we were looking for in DFAs:

> Emptiness and non-emptiness can be determined in linear time. Recall:
>
> **Lemma 1.** *$L(A) \neq \emptyset$ iff $\exists$ a path from s_0 to F in G_A.*
>
> DFA's are closed under boolean operations:
>
> > Complementation: $A \rightarrowtail A'$ such that $L(A') = \Sigma^* - L(A)$. The complexity of constructing the complement of A is $O(|A|)$.
> >
> > Intersection: $A_1, A_2 \rightarrowtail A$ such that $L(A) = L(A_1 \cap A_2) = L(A_1) \cap L(A_2)$. The complexity of constructing the intersection of A_1 and A_2 is $O(|A_1| \cdot |A_2|)$.
> >
> > Union: $A_1, A_2 \rightarrowtail A$ such that $L(A) = L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$. The complexity of constructing the union of A_1 and A_2 is $O(|A_1| \cdot |A_2|)$.

Conclusion: Boolean Closure is tractable for DFAs.

2 Using Automata for SAT

We wish to create a function DFA $\phi \rightarrowtail A_\phi$, such that $L(A_\phi) = models(\phi) \subseteq 0, 1^{|}AP(\phi)|$.

DFA(ϕ) /* $\phi \in$ Form, $n = |AP(\phi)|$ */
φ *is* p_i *return* Ap_i
φ *is* $(\neg\theta)$ *return* $\overline{A_\psi} \cap A^n$
φ *is* $(\theta \circ \psi)$ *return* $\circ(DFA(\theta), DFA(\psi))$

Definition 1. *The following DFA A_{p_i} accepts models(p_i):*

$$\begin{aligned} \Sigma &: \{0,1\} \\ S &: \{0,1,\ldots,n,n+1\} \\ s_0 &: \mathit{0} \\ F &: \{n\} \end{aligned}$$

where ρ is defined as:

$$\rho(j,a) = \begin{cases} j+1, & j \leq n \text{ and } j \neq i \\ i+1, & j = i \text{ and } a = 1 \\ n+1, & j = i \text{ and } a = 0 \\ j, & j = n+1 \end{cases}$$

Using this function to determine satisfiability is trivial:
$DFA - SAT(\varphi)$
$return\ L(DFA(\varphi)) \neq \emptyset$

Every operation on a DFA can be implemented polynomially, so it might appear that DFA-SAT runs in polynomial time. However, the intersection of m automata of bounded size is exponential in m.

Let's try to build a DFA using our algorithm on the following problem: We are given two binary numbers $p_1 \ldots p_n q_1 \ldots q_n$ and we want to check whether $\overline{p} = \overline{q}$.

In propositional formula, we can express the above problem as $\phi = \bigwedge_{i=1}^{n}(p_i \leftrightarrow q_i)$.

Therefore, using our DFA construction algorithm we will end up with $\bigcap_{i=1}^{n}(A_{p_i} \leftrightarrow A_{q_i})$. Note that since $|A_{p_i}| = |A_{q_i}| = O(n)$, $A_{p_i} \leftrightarrow A_{q_i}$ is of size $O(n^2)$. Therefore, $|\bigcap_{i=1}^{n}(A_{p_i} \leftrightarrow A_{q_i})| = O((n^2)^n) = O(n^{2n}) = O(n^n)$.

We can improve our algorithm to construct the smallest possible DFAs. We rely on canonicity:

Lemma 2. *Given a regular language $L \subseteq \Sigma^*$ there is a unique (up to isomorphism) DFA such that $L(A) = L$.*

Moreover, canonical DFAs are minimal. That is, there is a minimization algorithm such that given a DFA A it finds a minimal DFA B such that $L(A) = L(B)$. The time complexity of this algorithm is $O(n \log(n))$.

To improve our DFA construction algorithm we will add the minimization rule after every case. This will guarantee that that the running time will be $2^{O(n)}$.

Lemma 3. *$DFA - SAT(\varphi)$ uses $EXPTIME$ and $EXPSPACE$.*

Another name for 0/1 decision DFAs is BDDs (Binary Decision Diagrams). Note that the performance of BDDs depends on the order of the input. Considering the above example, it would be much easier if the bits were interleaved; that is, the input was of the form $p_1q_1 \ldots p_nq_n$. In this case, the size of the automaton would be $O(n)$.

Therefore, we conclude that to be efficient, we need the right ordering. However, finding the best ordering is NP-hard. To this purpose, current methods rely heavily on heuristics. First we need a good *initial order*. Second, we can use *Dynamic Reordering*. The idea behind Dynamic Reordering is that we start with a decent initial orderg. If this order blows up the size of the BDD, we try to improve it.

BDDs are not only used for satisfiability. Given $\phi, \psi \in$ Form we may want to prove $\models (\phi \leftrightarrow \psi)$. This problem may be of interest to us since we may want to find out if two given circuits ϕ and ψ compute the same boolean function.

$DFA - SAT(\varphi)$ is useful in comparing for logical equivalence between circuits or formulas.

$\phi \models\dashv \psi$ iff $L(DFA(\phi)) = L(DFA(\psi))$

Further, if in $DFA(\phi)$ we minimize after every step, we can say

$\phi \models\dashv \psi$ iff $DFA(\phi) = DFA(\psi)$

We make sure that we avoid isomorphism when we construct A_ϕ and A_ψ since checking for isomorphism is NP-hard.

3 Living with NP-Completeness

How does one cope with having an NP-Complete problem?

1. Find practical algorithms for the problem.
2. Find how simple you can make the problem and have it remain NP-Complete, i.e. explore the boundary between tractability and intractability.

To explore the boundary, we need to have a good definition of what a boundary is. Let's see this for the satisfiability problem:

> SAT: NP-complete
>
> CNF-SAT: NP-complete
>
> $k - CNF$-SAT: depends on the value of k, where a k-clause is a disjunction of at most k literals.

Let's investigate $k - CNF$-SAT for different values of k:

$3 - CNF$:

$$\bigwedge_{i=1}^{m} c_i,$$

where $c_i = \pm p_{i_1} \pm p_{i_2} \pm p_{i_3}$
$3-CNF$ is NP-Complete, but $1-CNF$ and $2-CNF$ are both in PTIME.

Lemma 4. *A $1-CNF$ formula is satisfiable iff it does not contain conflicting literals. (conflicting literals: $p \wedge \neg p$)*

It is less obvious to see why $2-CNF$ is in PTIME. We will see why later in the course.

Therefore, summarizing our results:

> $0 - CNF$-SAT: PTIME.
>
> $1 - CNF$-SAT: PTIME.

$2 - CNF$-SAT: PTIME.

$3 - CNF$-SAT: NP-complete.

Therefore, for $k - CNF$-SAT the boundary is between $2 - CNF$-SAT and $3 - CNF$-SAT.

Another interesting idea is to count the polarity of clauses. We define a k^+-clause as having at most k positive literals. $2^+ - CNF$ is NP-Complete, while $0^+ - CNF$ and $1^+ - CNF$ are both in PTIME. In fact, $1^+ - CNF$ are the Horn formulas.

4 Horn Formulas

We have the following nomenclature :

- Clause : A clause is a set of literals.
- Literal : A literal is either a proposition (p) or the negation of a proposition ($\neg p$) and is called a *positive* or a *negative literal* respectively.
- Horn Clause : A clause with at most 1 positive literal.
- Horn Formula : A set of Horn Clauses.

A Horn Clause can be of the following type :

1. There are no positive literals in the clause. In *3-CNF* form the clause would look like

   $(\neg p_1 \vee \neg p_2 \vee \neg p_3) \models\!\!\!\dashv \neg(p_1 \wedge p_2 \wedge p_3)$

   This is known as a *constraint*.

2. The clause contains precisely 1 positive literal.

   There can be two cases :

(a) There are no negative literals in the clause (this implies that there is exactly one literal in the clause and that is a positive literal)

p

This is called a *fact.*

(b) There are some negative literals in the clause.

$(\neg p_1 \vee \neg p_2 \vee q) \models\!\!\!\!= \!\!\mid (p_1 \wedge p_2) \rightarrow q$

This is called a *rule.*

Horn Formulas have been used in AI to develop expert systems or formalizations of bodies of knowledge.