Algorithm Theory

Georg Moser Mircea Dan Hernest

Institute of Computer Science @ UIBK

Summer 2007

Savitch's Theorem

Lemma

REACHABILITY $\in$ **SPACE**$(\log^2 n)$

Proof Idea

the deterministic algorithm (for REACHABILITY) we used before needs linear space. To improve one uses a similar divide & conquer approach as in quicksort.

Theorem — Savitch

if f is space constructible, then **NSPACE**$(f(n)) \subseteq$ **SPACE**$(f^2(n))$

Corollary

1. **PSPACE** = **NPSPACE**
2. **NSPACE**$(f(n))$ = **coNSPACE**$(f(n))$ Immerman-Szelepsényi

we suppose f is space constructible

Satisfiability

Definition

a Boolean expression φ is built up from Boolean variables $X = \{x_1, x_2, \dots\}$ and truth values **true**, **false**, by the unary operation $\neg$ and the binary operations $\vee$ and $\wedge$

- ➡ a map $T \colon X' \to \{\textbf{true}, \textbf{false}\}$ ($X' \subseteq X$, X' finite) is a (truth) assignment
- ➡ we call T appropriate for φ if X' contains all variables in φ.
- ➡ we write $T \models \varphi$ if T satisfies φ.
- ➡ we say φ is valid if φ is satisfied by all assignments T appropriate for φ.

Theorem

a Boolean expression φ is unsatisfiable iff its negation $\neg\varphi$ is valid

Conjunctive Normal Form

- ➡ a Boolean expression φ is in conjunctive normal form (CNF) if

$$\varphi \equiv \bigwedge_{i=1}^{n} C_i \ ,$$

  where $n \geqslant 1$, and each C_i is the disjunction of one or more literals
- ➡ C_i is also called a clause

Example

CNF

$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3))$

is an expression in CNFthat is unsatisfiable

Theorem

every Boolean expression is equivalent to one in CNF (or DNF), but the CNF is not unique

Example

non-uniqueness

$$x \equiv (x \vee x) \equiv (x \vee x) \wedge (y \vee \neg y)$$

Representation

a Boolean expression is represented as a string over an alphabet containing $x, 0, 1, (,), \neg, \vee, \wedge$

Problem SAT

SAT

given a Boolean expression φ in CNF, is φ satisfiable?

Complexity

- ➡ we know many ways to decide the language SAT, e.g. truth tables, OBDDs, resolution, etc.
- ➡ still in the worst case all these algorithms are exponential
- ➡ we only know SAT $\in$ **TIME**(2^n)

Lemma

SAT $\in$ **NP**

Proof

- ➡ use characterisation via polynomial verifier
- ➡ use the assignment as certificate

□

Problem HORNSAT

HORNSAT

given a Boolean expression φ that is a Horn formula in CNF, is it satisfiable?

Definition

Horn formulas

- a Horn clause is a clause that has at most one positive literal.

  Example:

  $$(\neg x_2 \vee x_3),\ (\neg x_1 \vee \neg x_2 \vee \neg x_3) \text{ are Horn clauses}$$

- a Boolean expression is a Horn formula if equivalent to a CNF where all clauses are Horn

Complexity

HORNSAT $\in$ **P**

Boolean function

an n-ary Boolean function is a function

$$f\colon \{\textbf{true}, \textbf{false}\}^n \to \{\textbf{true}, \textbf{false}\}$$

Fact

any expression φ can be conceived as an n-ary Boolean function if φ has n (distinct) variables

Proof

suppose $\{x_1, \dots, x_n\}$ occur in φ; let $\vec{t} = (t_1, \dots, t_n)$ be truth values; assume for the assignment T, $T(x_i) = t_i$. Set

$$f(\vec{t}) := \begin{cases} \textbf{true} & \text{if} \quad T \models \varphi \\ \textbf{false} & \text{if} \quad T \not\models \varphi \end{cases}$$

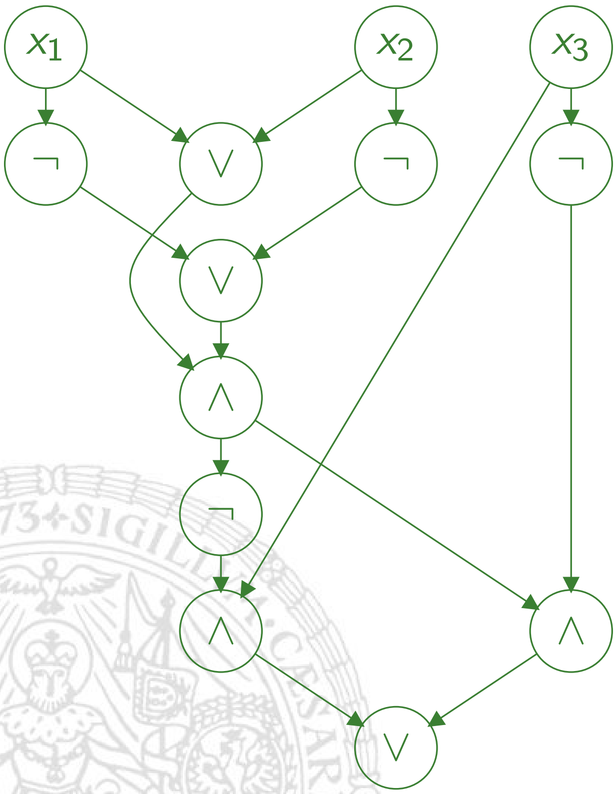
□

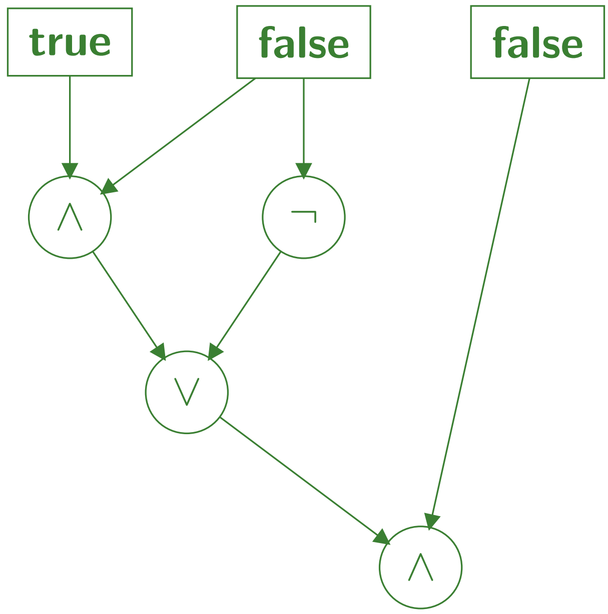
Fact

any n-ary Boolean function f can be expressed as a Boolean expression involving $x_1, \dots, x_n$

Boolean circuit (1)

$(x_3 \wedge \neg((x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))) \vee (\neg x_3 \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2))$





$(((\textbf{true} \wedge \textbf{false}) \vee \neg\textbf{false}) \wedge \textbf{false})$

Boolean circuit (2)

Definition — Boolean circuit

a Boolean circuit is a graph $C = (V, E)$, such that the nodes $V = \{1, \ldots, n\}$ are called gates

1. there are no cycles in C. Hence we can write all edges as (i, j), where $i < j$
2. the indegrees of the gates are 0,1, or 2
3. each gate has a sort from $\{\textbf{true}, \textbf{false}, \neg, \vee, \wedge\} \cup \{x_1, x_2, \ldots\}$:
   - ➡ if the sort of the gate is from $\{\textbf{true}, \textbf{false}\} \cup \{x_1, x_2, \ldots\}$, then it is an input gate and has no incoming edges
   - ➡ if the sort is $\neg$, then the indegree is 1
   - ➡ if the sort is from $\{\vee, \wedge\}$, then the indegree is 2
4. the node n is called output gate.

Value of a Circuit

- we write $s(i)$, for the sort of gate i
- let $X(C)$ be the set of all variables occurring in the circuit C
- an assignment T is appropriate for C, if defined for all variables in $X(C)$
- given T, the truth value of gate j, $T(j)$ is defined as follows:
  1. $T(j) :=$ **true**, if $s(j) =$ **true**
  2. $T(j) :=$ **false**, if $s(j) =$ **false**
  3. $T(j) := T(s(j))$, if $s(j) \in X$
  4. $T(j) =$ not $T(i)$, if $s(j) = \neg$ and $(i, j) \in E$
  5. $T(j) = T(i_1)$ or $T(i_2)$, if $s(j) = \vee$ and $(i_1, j), (i_2, j) \in E$
  6. $T(j) = T(i_1)$ and $T(i_2)$, if $s(j) = \wedge$ and $(i_1, j), (i_2, j) \in E$
- the value of C (written $T(C)$) is defined as $T(n)$, where n is the output gate

CIRCUIT SAT

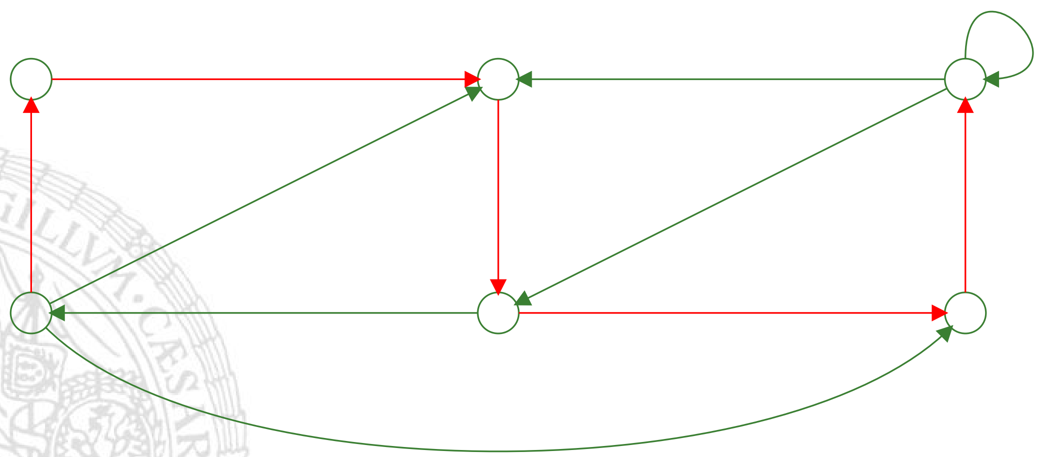
given a circuit C, is there a truth assignment T appropriate for C so that $T(C) =$ **true**?

CIRCUIT VALUE

given a variable-free circuit C, is $T(C) =$ **true**?

HAMILTON PATH

given a (directed) graph. Is there a path that visit every node exactly once?



Reduction (1)

Complexity

the problem HAMILTON PATH is in **NP**

Definition

- ➡ a reduction is a procedures that solves a computational problem A by transforming any instance of A to an equivalent instance of a previously solved problem

Example

we reduced MAX FLOW to REACHABILITY

Definition reduction

- ➡ algorithm A reduces to B if there exists a transformation R which, for every input x of A, produces an equivalent input $R(x)$ of B

Fact

if A reduces to B, then B is not easier than A

Reduction (2)

Definition logspace-reductions

L_1 is reducible to L_2 if

1. exists a function R from strings to strings
2. computable by a deterministic TM in space $\mathcal{O}(\log n)$ such that
3. for all x:

$$x \in \mathrm{L}_1 \quad \text{iff} \quad R(x) \in \mathrm{L}_2 \ .$$

Theorem

if R is a (logspace-) reduction computed by a TM M, then for all inputs x, M halts after a polynomial number of steps

Proof

$\mathbf{L} \subseteq \mathbf{P}$ □

Reduction: HAMILTON PATH $\longrightarrow$ SAT

suppose G has n nodes; the formula $R(G)$ will have n^2 Boolean variables x_{ij}; variable x_{ij} represents that node j is the ith node in the Hamilton path

Clauses of $R(G)$

- $(x_{1j} \vee \cdots \vee x_{nj})$, expressing that node j occurs in the path.
- $\neg(x_{ij} \wedge x_{kj})$ for all i, k, $i \neq k$. This gives the clause $(\neg x_{ij} \vee \neg x_{kj})$
- $(x_{i1} \vee \cdots \vee x_{in})$, expressing that some node occurs at the ith position in the path.
- $\neg(x_{ij} \wedge x_{ik})$ for all j, k, $j \neq k$. Gives $(\neg x_{ij} \vee \neg x_{ik})$
- $\neg(x_{ki} \wedge x_{k+1,j})$ for each $(i,j) \in G$ which is not an edge. Gives: $\neg x_{ki} \vee \neg x_{k+1,j}$

$R(G)$ is the conjunction of all these clauses

Proof (1)

Theorem

R is a reduction from HAMILTON PATH to SAT.

$R(G)$ is satisfied by T implies that G has a Hamilton path

1. for each i there exists a unique j so that $T(x_{ij}) = \textbf{true}$.
2. for each j there exists a unique i so that $T(x_{ij}) = \textbf{true}$.
3. i.e., there exists a permutation $(\pi(1), \dots, \pi(n))$, where $\pi(i) = j$ iff $T(x_{ij}) = \textbf{true}$.
4. by the last group of clauses $(\pi(1), \dots, \pi(n))$ is a path in G

G has a Hamilton path implies that $R(G)$ is satisfiable

1. suppose that $(\pi(1), \dots, \pi(n))$ is a Hamilton path.
2. set $T(x_{ij}) = \textbf{true}$ iff $\pi(i) = j$.

Proof (2)

R is computable in space $\mathcal{O}(\log n)$

1. generate the first four groups of clauses, this depends only on n
2. needs 3 counters i, j, k for the indices of the variables
3. generate all clauses of the form $(\neg x_{ki} \lor \neg x_{k+1,j})$; reusing space
4. test for each clause $(\neg x_{ki} \lor \neg x_{k+1,j})$ whether there exists an edge $(i, j) \in G$ or not

□