A non-examinable sketch proof of the Cook-Levin theorem COMS20010 (Algorithms II)

John Lapinskas, University of Bristol

Lies, damned lies and sketch proofs

Cook-Levin Theorem: Any problem in NP is Cook-reducible to SAT.

Let X be **any** problem in NP, and let $\vec{x}$ be an instance of X.

Then we will construct a CNF formula $F_{\vec{x}}$, of size polynomial in $|\vec{x}|$, which is satisfiable if and only if $\vec{x}$ is a `Yes` instance. Our Cook reduction then just applies the SAT oracle to $F_{\vec{x}}$ and outputs the result.

By definition of NP, there is a polynomial-time algorithm `Verify` such that $\vec{x}$ is a `Yes` instance if and only if $\texttt{Verify}(\vec{x},\vec{w}) = \texttt{Yes}$ for some $\vec{w}$. So we would like to express the statement "$\texttt{Verify}(\vec{x},\vec{w}) = \texttt{Yes}$" as a CNF formula in $\vec{w}$.

Problem: We know **nothing** about how `Verify` works. So to do this, we need to be able to express "A computer running program P on input I for t steps outputs `Yes`" as a CNF formula of polynomial size...!

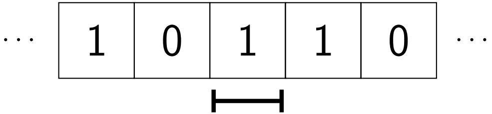
Turing machines

Our goal: Given a program P, we would like a CNF formula $f(P, \vec{I}, t)$ which is satisfiable iff running P on input $\vec{I}$ for t steps outputs `Yes`.

We would like to construct $f(P, \vec{I}, t)$ in time poly$(\vec{I}, t)$.

We could in principle do this for an actual computer architecture, but it would be unspeakably awful. So let's use a Turing machine instead.

Recall from COMS11700: A **Turing machine** is a two-sided infinite string of tape divided into cells containing binary values, plus a tape head. At each time step, based on the current cell and its internal state, the tape head writes a 1 or 0 and moves left or right along the tape, and then the Turing machine changes state or halts. For example:



Machine in state 3, head reads 1 $\longrightarrow$ Write 0, move right, enter state 3.

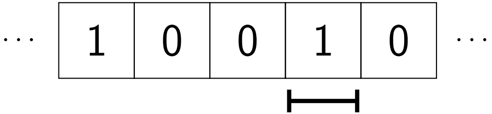
Turing machines

Our goal: Given a program P, we would like a CNF formula $f(P, \vec{I}, t)$ which is satisfiable iff running P on input $\vec{I}$ for t steps outputs `Yes`.

We would like to construct $f(P, \vec{I}, t)$ in time $\text{poly}(\vec{I}, t)$.

We could in principle do this for an actual computer architecture, but it would be unspeakably awful. So let's use a Turing machine instead.

Recall from COMS11700: A **Turing machine** is a two-sided infinite string of tape divided into cells containing binary values, plus a tape head. At each time step, based on the current cell and its internal state, the tape head writes a 1 or 0 and moves left or right along the tape, and then the Turing machine changes state or halts. For example:



Machine in state 3, head reads 1 $\longrightarrow$ Write 0, move right, enter state 3.

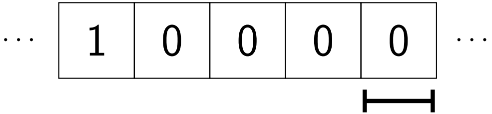
Turing machines

Our goal: Given a program P, we would like a CNF formula $f(P, \vec{I}, t)$ which is satisfiable iff running P on input $\vec{I}$ for t steps outputs `Yes`.

We would like to construct $f(P, \vec{I}, t)$ in time $\mathrm{poly}(\vec{I}, t)$.

We could in principle do this for an actual computer architecture, but it would be unspeakably awful. So let's use a Turing machine instead.

Recall from COMS11700: A **Turing machine** is a two-sided infinite string of tape divided into cells containing binary values, plus a tape head. At each time step, based on the current cell and its internal state, the tape head writes a 1 or 0 and moves left or right along the tape, and then the Turing machine changes state or halts. For example:



Machine in state 3, head reads 0 $\longrightarrow$ Write 1, move left, enter state 7.

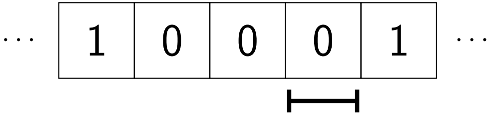
Turing machines

Our goal: Given a program P, we would like a CNF formula $f(P, \vec{I}, t)$ which is satisfiable iff running P on input $\vec{I}$ for t steps outputs `Yes`.

We would like to construct $f(P, \vec{I}, t)$ in time poly$(\vec{I}, t)$.

We could in principle do this for an actual computer architecture, but it would be unspeakably awful. So let's use a Turing machine instead.

Recall from COMS11700: A **Turing machine** is a two-sided infinite string of tape divided into cells containing binary values, plus a tape head. At each time step, based on the current cell and its internal state, the tape head writes a 1 or 0 and moves left or right along the tape, and then the Turing machine changes state or halts. For example:



Machine in state 7, head reads 0 $\longrightarrow$ Write 1, move left, enter state 1.

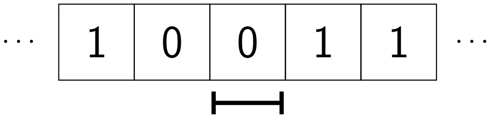
Turing machines

Our goal: Given a program P, we would like a CNF formula $f(P, \vec{I}, t)$ which is satisfiable iff running P on input $\vec{I}$ for t steps outputs `Yes`.

We would like to construct $f(P, \vec{I}, t)$ in time $\text{poly}(\vec{I}, t)$.

We could in principle do this for an actual computer architecture, but it would be unspeakably awful. So let's use a Turing machine instead.

Recall from COMS11700: A **Turing machine** is a two-sided infinite string of tape divided into cells containing binary values, plus a tape head. At each time step, based on the current cell and its internal state, the tape head writes a 1 or 0 and moves left or right along the tape, and then the Turing machine changes state or halts. For example:



Machine in state 1, head reads 0 $\longrightarrow$ Write 0, move right, enter state 10.

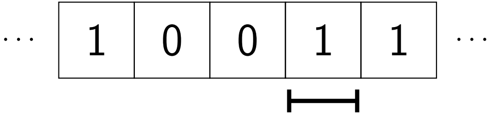
Turing machines

Our goal: Given a program P, we would like a CNF formula $f(P, \vec{I}, t)$ which is satisfiable iff running P on input $\vec{I}$ for t steps outputs `Yes`.

We would like to construct $f(P, \vec{I}, t)$ in time $\text{poly}(\vec{I}, t)$.

We could in principle do this for an actual computer architecture, but it would be unspeakably awful. So let's use a Turing machine instead.

Recall from COMS11700: A **Turing machine** is a two-sided infinite string of tape divided into cells containing binary values, plus a tape head. At each time step, based on the current cell and its internal state, the tape head writes a 1 or 0 and moves left or right along the tape, and then the Turing machine changes state or halts. For example:



Machine in state 10, head reads 1 $\longrightarrow$ Halt.

Expressing computation with logic

Our goal: Given a program P, we would like a CNF formula $f(P, \vec{I}, t)$ which is satisfiable iff running P on input $\vec{I}$ for t steps outputs `Yes`.

We would like to construct $f(P, \vec{I}, t)$ in time poly$(\vec{I}, t)$.

Recall from COMS11700: A **Turing machine** is a two-sided infinite string of tape divided into cells containing binary values, plus a tape head. At each time step, based on the current cell and its internal state, the tape head writes a 1 or 0 and moves left or right along the tape, and then the Turing machine changes state or halts.

The **Church-Turing Thesis** says: Any computable function is computable using a Turing machine.

The **Strong Church-Turing Thesis** says: Any function computable **in polynomial time** is computable **in polynomial time** using a Turing machine. (Ignoring quantum computers, anyway...)

So we can take our program in the form of a Turing machine, which is much easier to simulate!

Expressing computation with logic

Our new goal: Given a **Turing machine** M, we would like a CNF formula $f(M, \vec{I}, t)$ which is satisfiable iff after running M on input $\vec{I}$ for t steps, it halts with output `Yes`.

We would like to construct $f(M, \vec{I}, t)$ in time poly$(\vec{I}, t)$.

Idea: Model the entire computation from start to finish, step by step.

We will have variables:

- $C_{p,\tau}$ to model the cell contents. We want $C_{p,\tau} = 1$ if and only if cell p contains a 1 at time τ.
- $S_{s,\tau}$ to model which state the machine is in. We want $S_{s,\tau} = 1$ if and only if the machine is in state s at time τ.
- $P_{p,\tau}$ to model the position of the tape head. We want $P_{p,\tau} = 1$ if and only if the tape head is at cell p at time τ.

Problem: The tape is infinite, so we need infinitely many variables!

Expressing computation with logic

Our new goal: Given a **Turing machine** M, we would like a CNF formula $f(M, \vec{I}, t)$ which is satisfiable iff after running M on input $\vec{I}$ for t steps, it halts with output `Yes`.

We would like to construct $f(M, \vec{I}, t)$ in time poly$(\vec{I}, t)$.

Idea: Model the entire computation from start to finish, step by step.

We will have variables:

- $C_{p,\tau}$ to model the cell contents. We want $C_{p,\tau} = 1$ if and only if cell p contains a 1 at time τ.
- $S_{s,\tau}$ to model which state the machine is in. We want $S_{s,\tau} = 1$ if and only if the machine is in state s at time τ.
- $P_{p,\tau}$ to model the position of the tape head. We want $P_{p,\tau} = 1$ if and only if the tape head is at cell p at time τ.

Solution: In time t the tape head can only move t spaces, so the state of the Turing machine is determined entirely by cells $-t, -t+1, \ldots, t$.
So actually we only need $O(t^2)$ variables.

Our new goal: Given a **Turing machine** M, we would like a CNF formula $f(M, \vec{I}, t)$ which is satisfiable iff after running M on input $\vec{I}$ for t steps, it halts with output `Yes`.

We would like to construct $f(M, \vec{I}, t)$ in time poly$(\vec{I}, t)$.

Our variables: We want: $C_{p,\tau}$ = contents of cell p at time τ;
$S_{s,\tau} = 1$ iff machine is in state s at time τ;
$P_{p,\tau} = 1$ iff head is at cell p at time τ.

Write $\mathcal{M}_\tau$ for the collection of variables corresponding to the machine's state at time τ, i.e. $\{C_{-t,\tau}, \ldots, C_{t,\tau},\ S_{1,\tau}, \ldots, S_{q,\tau},\ P_{-t,\tau}, \ldots, P_{t,\tau}\}$.

We would like to write:

$$\begin{aligned}
f(M, \vec{I}, t) = [\mathcal{M}_0 \leftrightarrow \text{Tape reads } \vec{I}, \text{ state is 1, head at cell 0}]& \\
\wedge [\mathcal{M}_1 \text{ is derived correctly from } \mathcal{M}_0]& \\
\wedge [\mathcal{M}_2 \text{ is derived correctly from } \mathcal{M}_1]& \\
\vdots \quad & \\
\wedge [\mathcal{M}_t \text{ is derived correctly from } \mathcal{M}_{t-1}]& \\
\wedge [\mathcal{M}_t \leftrightarrow \text{Machine has halted with output } \texttt{Yes}]&
\end{aligned}$$

Our new goal: Given a **Turing machine** M, we would like a CNF formula $f(M, \vec{I}, t)$ which is satisfiable iff after running M on input $\vec{I}$ for t steps, it halts with output `Yes`.

We would like to construct $f(M, \vec{I}, t)$ in time poly($\vec{I}, t$).

Our variables: We want: $C_{p,\tau} =$ contents of cell p at time τ;
$S_{s,\tau} = 1$ iff machine is in state s at time τ;
$P_{p,\tau} = 1$ iff head is at cell p at time τ.

Write $\mathcal{M}_\tau$ for the collection of variables corresponding to the machine's state at time τ, i.e. $\{C_{-t,\tau}, \ldots, C_{t,\tau},\ S_{1,\tau}, \ldots, S_{q,\tau},\ P_{-t,\tau}, \ldots, P_{t,\tau}\}$.

We would like to write:

$$
\begin{aligned}
f(M, \vec{I}, t) = [\mathcal{M}_0 \leftrightarrow \text{Tape reads } \vec{I}, \text{ state is 1, head at cell 0}]& \\
\wedge \bigwedge_{\tau=1}^{t} [\mathcal{M}_\tau \text{ is derived correctly from } \mathcal{M}_{\tau-1}]& \\
\wedge\, [\mathcal{M}_t \leftrightarrow \text{Machine has halted with output } \texttt{Yes}].&
\end{aligned}
$$

If we can express these as CNF statements **of length polynomial in t**, we're done since an AND of CNFs is in CNF.

This is painful but ultimately doable!

Our new goal: Given a **Turing machine** M, we would like a CNF formula $f(M, \vec{I}, t)$ which is satisfiable iff after running M on input $\vec{I}$ for t steps, it halts with output `Yes`.

We would like to construct $f(M, \vec{I}, t)$ in time poly$(\vec{I}, t)$.

Our variables: We want: $C_{p,\tau}$ = contents of cell p at time τ;
$S_{s,\tau} = 1$ iff machine is in state s at time τ;
$P_{p,\tau} = 1$ iff head is at cell p at time τ.

Write $\mathcal{M}_\tau$ for the collection of variables corresponding to the machine's state at time τ, i.e. $\{C_{-t,\tau}, \dots, C_{t,\tau},\ S_{1,\tau}, \dots, S_{q,\tau},\ P_{-t,\tau}, \dots, P_{t,\tau}\}$.

The key point: Computer operations are local — the changes from $\mathcal{M}_\tau$ to $\mathcal{M}_{\tau+1}$ only depend on a few variables in $\mathcal{M}_\tau$!

So we check consistency with an AND of **many** clauses that look like:

$$[P_{i,\tau} = 1 \wedge S_{3,\tau} = 1 \wedge C_{i,\tau} = 0 \Rightarrow C_{i,\tau+1} = 1]$$

Each such clause has only 4 variables, so it can be expressed as an $O(1)$-length CNF formula. We need $\Theta(t)$ such formulae for every variable in $\mathcal{M}_{\tau+1}$, one for each possible (position, state, cell contents) tuple, so in total our consistency check will have length $\Theta(t^2)$ and $|f(M, \vec{I}, t)| \in \Theta(t^3)$. □