Computational Complexity Theory

Lecture 3: Cook-Levin Theorem

Indian Institute of Science

Recap: Complexity Class NP

- Definition. A language $L \subseteq \{0,1\}^*$ is in NP if there's a polynomial function $p: \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial time TM M (called the *verifier*) such that for every x,

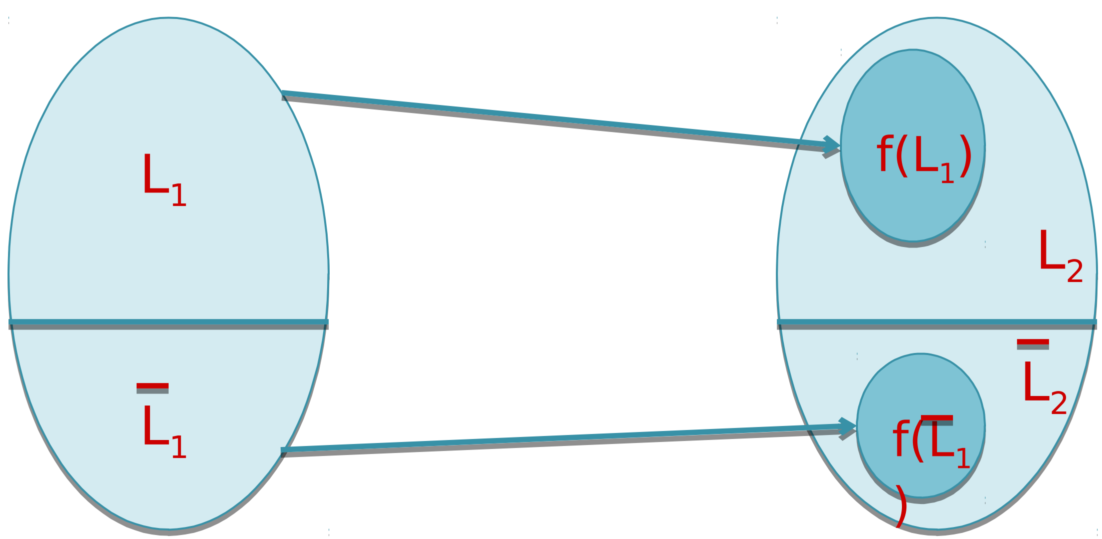
$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

u is called a *certificate or witness* for x (w.r.t L and M) if $x \in L$

Recap: Polynomial time reduction

- Definition. We say a language $L_1 \subseteq \{0,1\}^*$ is <u>*polynomial time (Karp) reducible*</u> to a language $L_2 \subseteq \{0,1\}^*$ if there's a polynomial time computable function f s.t.

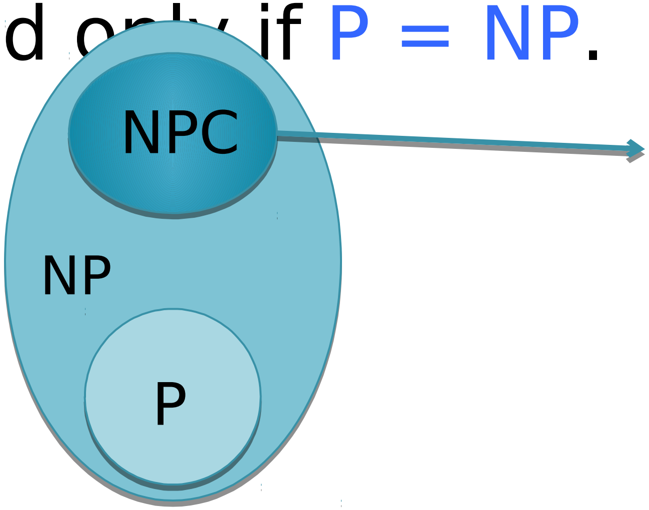
$$x \in L_1 \Leftrightarrow f(x) \in L_2$$



Recap: NP-completeness

- **Definition.** A language L’ is *NP-hard* if for every L in NP, $L \leq_p L'$. Further, L’ is *NP-complete* if L’ is in NP and is NP-hard.

- **Observe.** If L’ is NP-hard and L’ is in P then P = NP. If L’ is NP-complete then L’ in P if and only if P = NP.



Hardest problems inside NP in the sense that if one NPC problem is in P then all problems in NP is in P.

Recap: A natural NP-complete problem

- Definition. A boolean formula is in *Conjunctive Normal Form* (*CNF*) if it is an AND of OR of literals.

  e.g. $\phi = (x_1 \lor x_2) \land (x_3 \lor \neg x_2)$

- Definition. Let SAT be the language consisting of all *satisfiable CNF formulae.*

- Theorem. *(Cook-Levin)* SAT is NP-complete.

  Easy to see that SAT is in NP.

  Need to show that SAT is NP-hard.

Proof of Cook-Levin Theorem

Cook-Levin theorem: Proof

- Main idea: Computation is *local*; i.e. every step of computation *looks at* and *changes* only constantly many bits; and this step can be implemented by a small CNF formula.

- Let $L \in NP$. We intend to come up with a polynomial time computable function $f\colon x \longmapsto \phi_x$ s.t.,
  - $x \in L \iff \phi_x \in SAT$

Notation: $|\phi_x|$:= size of ϕ_x

= number of $\vee$ or $\wedge$ in ϕ_x

Cook-Levin theorem: Proof

- Language L has a poly-time verifier M such that

  $$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

- Idea: Capture the computation of $M(x, ..)$ by a CNF ϕ_x such that

  $$\exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1 \iff \phi_x \text{ is satisfiable}$$

- For any fixed x, $M(x, ..)$ is a deterministic TM that takes u as input and runs in time polynomial in $|u|$.

Cook-Levin theorem: Proof

- Main Theorem. Let N be a deterministic TM that runs in time T(n) on every input u of length n, and outputs 0/1. Then,
  1. There's a CNF ϕ of size *poly*(T(n)) such that ϕ(u, *"auxiliary variables"*) is satisfiable as a function of the *"auxiliary variables"* if and only if N(u) =1.
  2. ϕ is computable in time *poly(T(n))*.

Cook-Levin theorem: Proof

- Main Theorem. Let N be a deterministic TM that runs in time T(n) on every input u of length n, and outputs 0/1. Then,
  1. There's a CNF ϕ of size *poly*(T(n)) such that ϕ(u, *"auxiliary variables"*) is satisfiable <u>as a function of the *"auxiliary variables"*</u> if and only if N(u) =1.
  2. ϕ is computable in time *poly(T(n))*.

- ϕ(u, *"auxiliary variables"*) is satisfiable <u>as a function of all variables</u> if and only if ∃u s.t N(u) =1.

Cook-Levin theorem: Proof

- Main Theorem. Let N be a deterministic TM that runs in time T(n) on every input u of length n, and outputs 0/1. Then,
  1. There's a CNF ϕ of size *poly*(T(n)) such that ϕ(u, *"auxiliary variables"*) is satisfiable as a function of the *"auxiliary variables"* if and only if N(u) =1.
  2. ϕ is computable in time *poly(T(n))*.

- Cook-Levin theorem follows from above!

Proof of Main Theorem

Main theorem: Proof

- Step 1. Let N be a deterministic TM that runs in time $T(n)$ on every input u of length n, and outputs $0/1$. Then,
  1. There's a boolean circuit ψ of size *poly*$(T(n))$ such that $\psi(u) = 1$ if and only if $N(u) = 1$.
  2. ψ is computable in time *poly(T(n))*.

- Step 2. "Convert" circuit ψ to a CNF ϕ efficiently by introducing <u>auxiliary variables</u>.

Main theorem: Step 1

- Assume (w.l.o.g) that N has a single tape and it writes its output on the first cell at the end of computation.

Main theorem: Step 1

- Assume (w.l.o.g) that N has a single tape and it writes its output on the first cell at the end of computation.

- A step of computation of N consists of
  - Changing the content of the current cell
  - Changing state
  - Changing head position

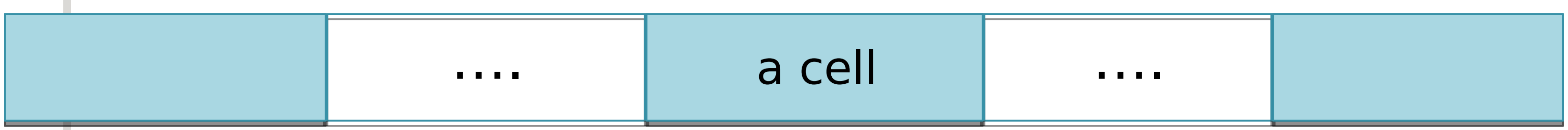
Main theorem: Step 1

- Assume (w.l.o.g) that N has a single tape and it writes its output on the first cell at the end of computation.

- A step of computation of N consists of
  - Changing the content of the current cell
  - Changing state
  - Changing head position

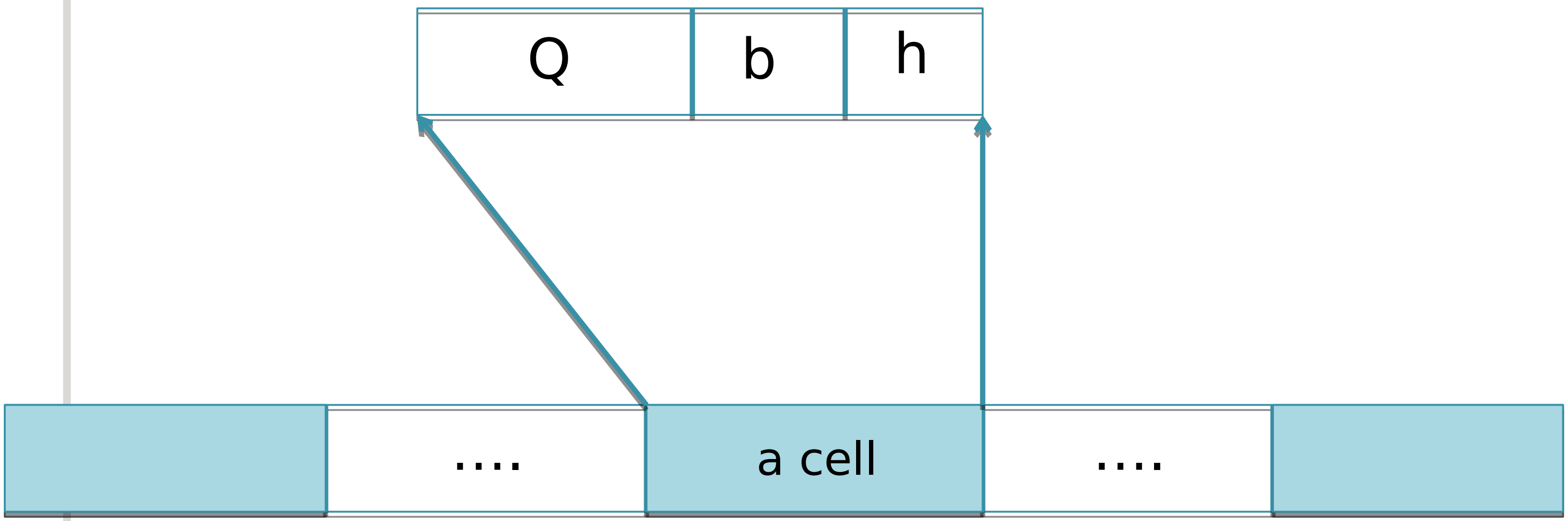
- Think of a ‘compound’ tape: every cell stores the current state, a bit content and head indicator.

Main theorem: Step 1



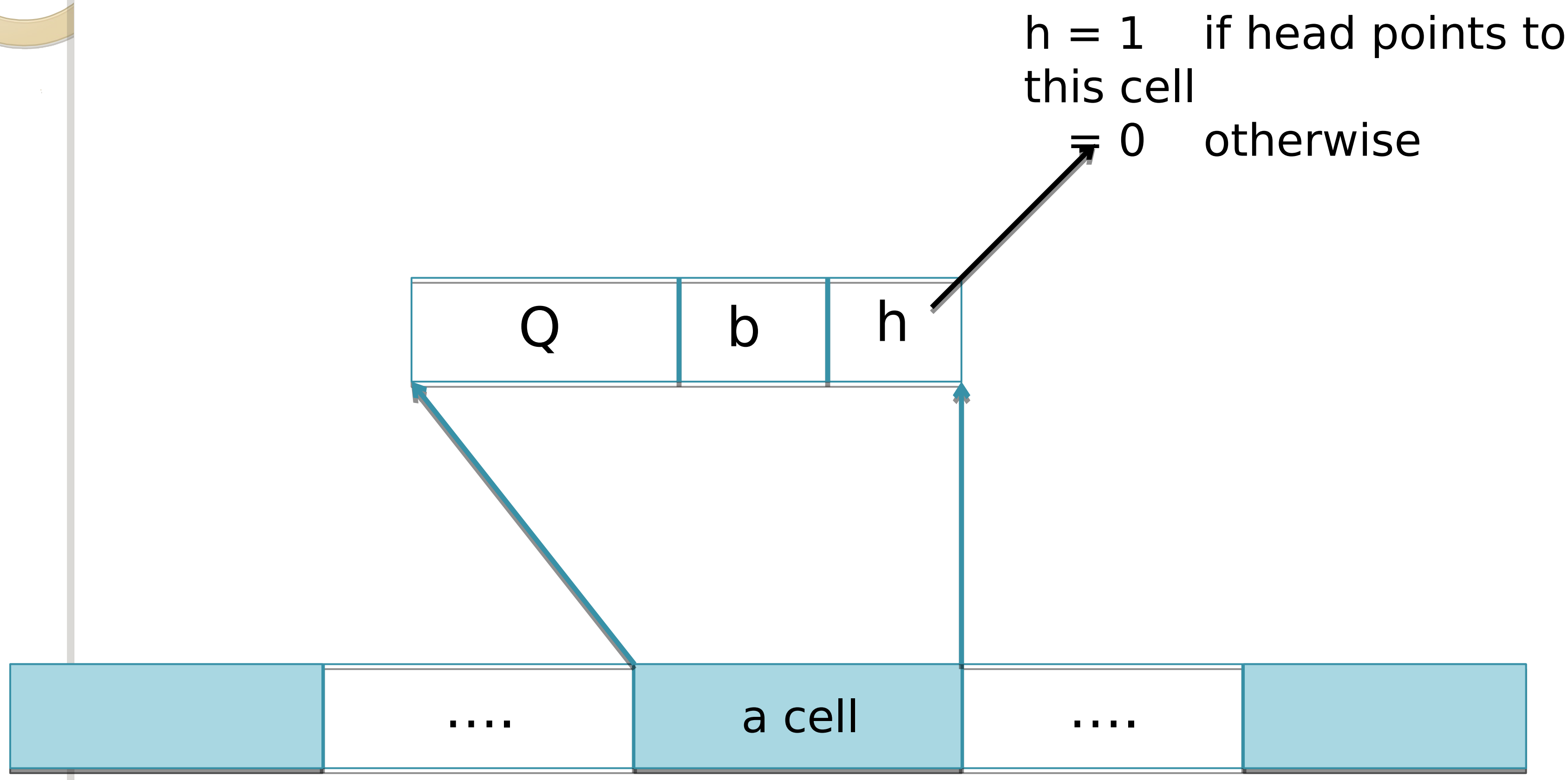
A compound tape

Main theorem: Step 1



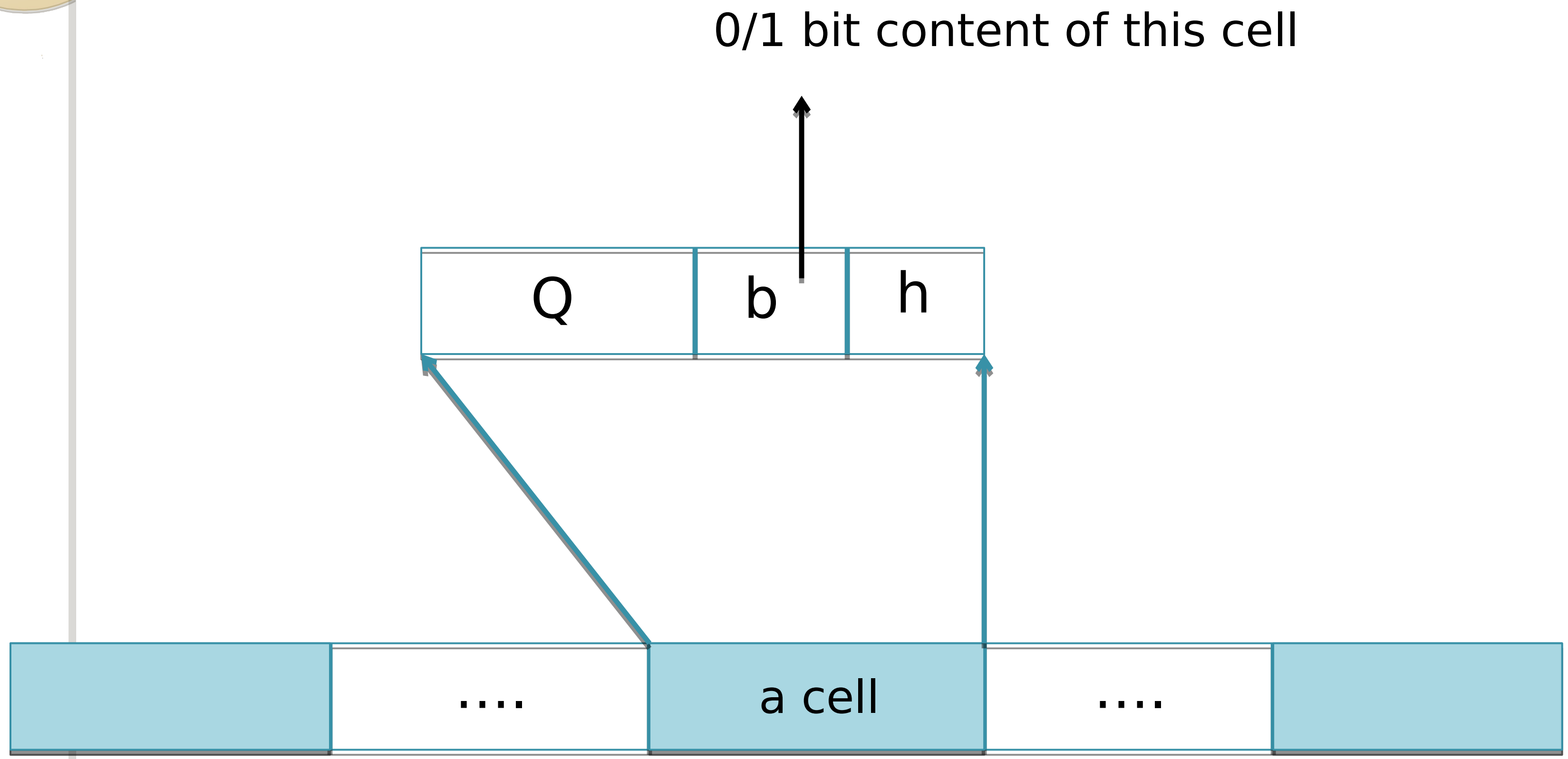
A compound tape

Main theorem: Step 1



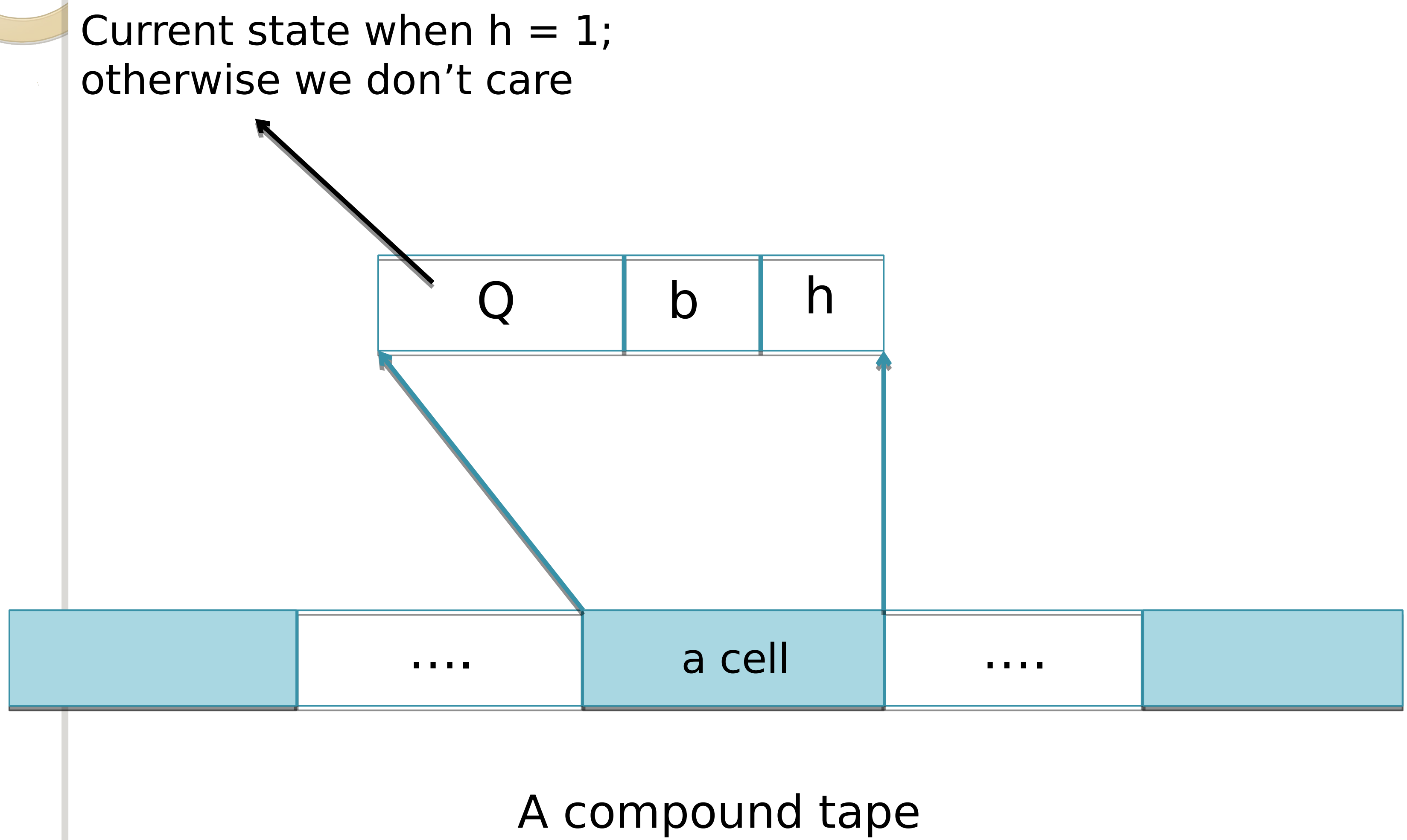
A compound tape

Main theorem: Step 1



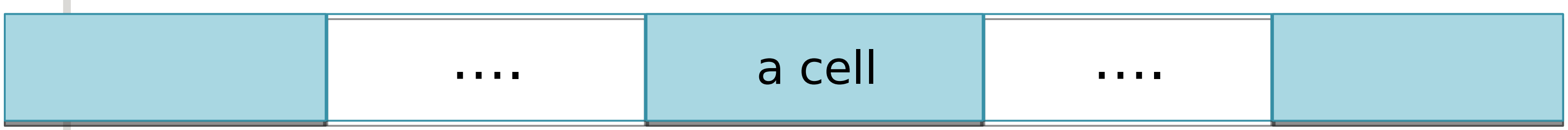
A compound tape

Main theorem: Step 1



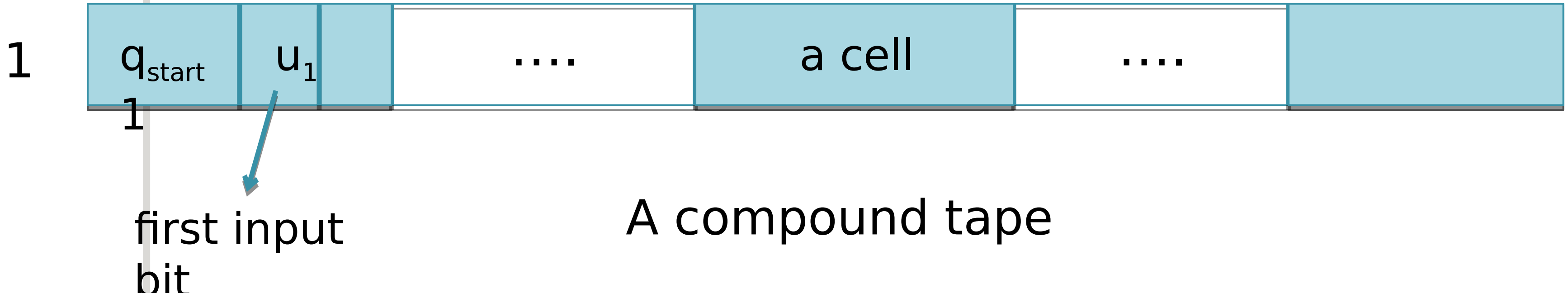
Main theorem: Step 1

- Computation of N can be completely described by a sequence of T(n) compound tapes, the i-th of which captures a `*snapshot*' of N's computation at the i-th step.

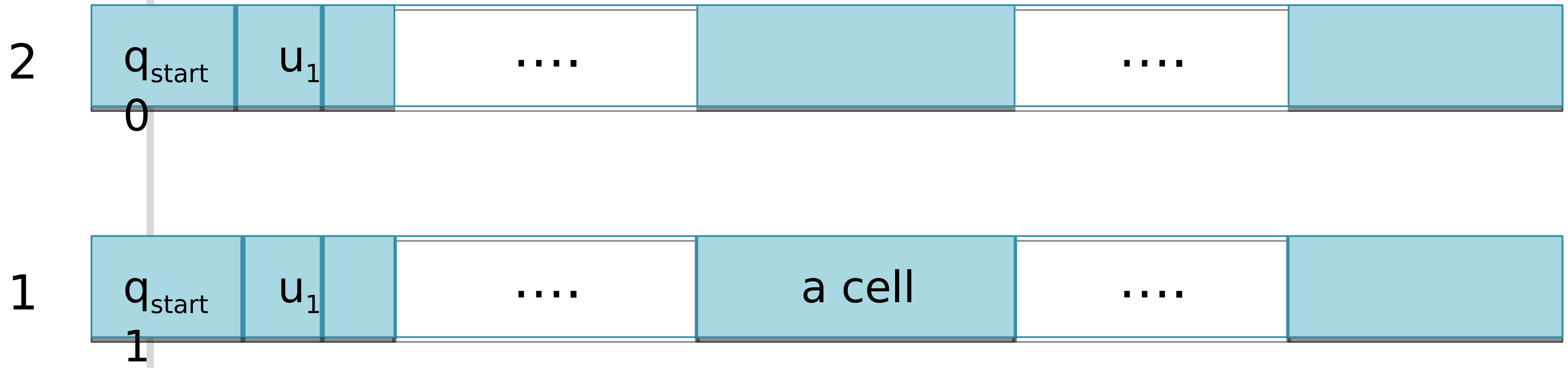


A compound tape

Main theorem: Step 1

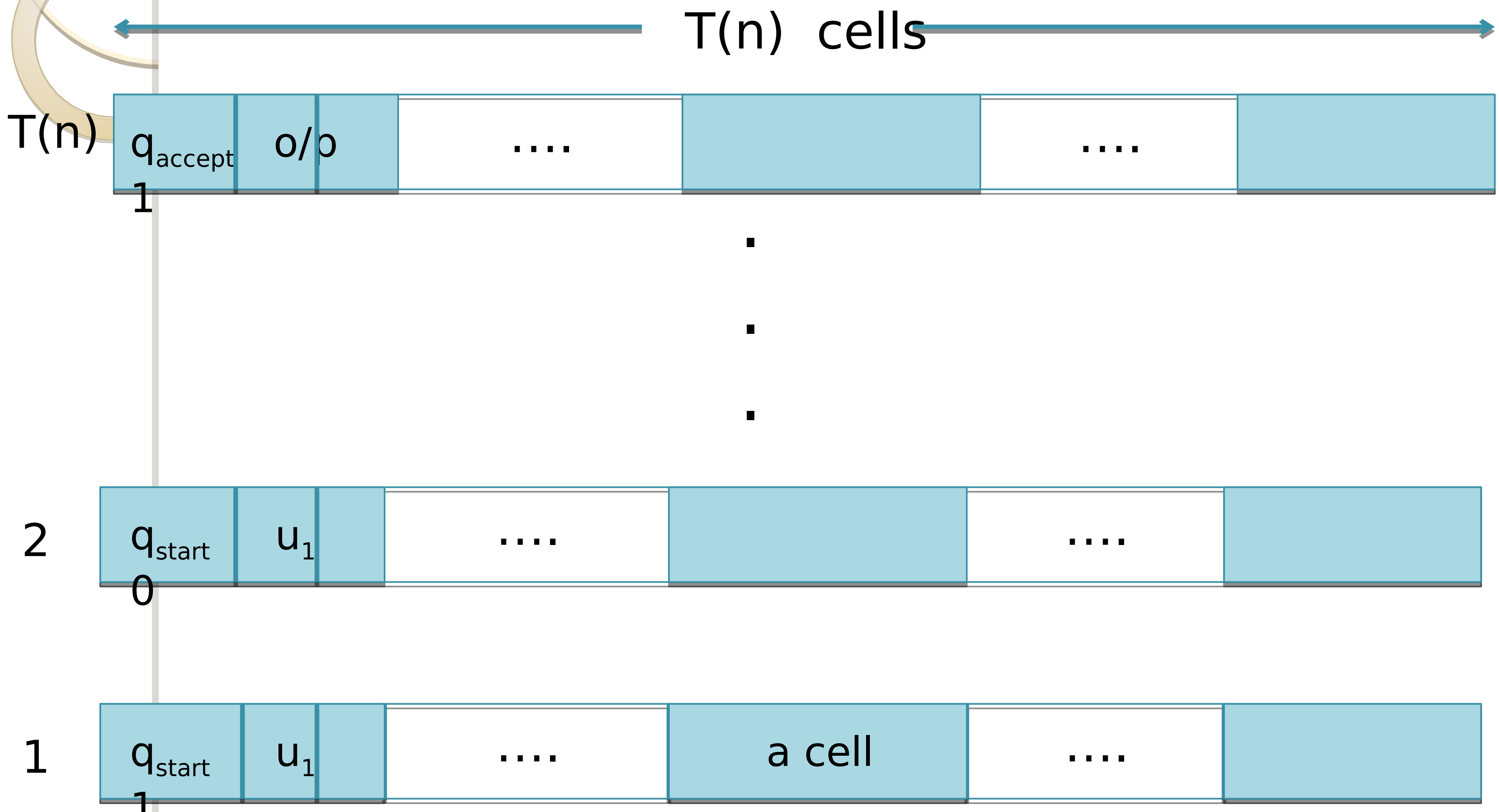


Main theorem: Step 1



A compound tape

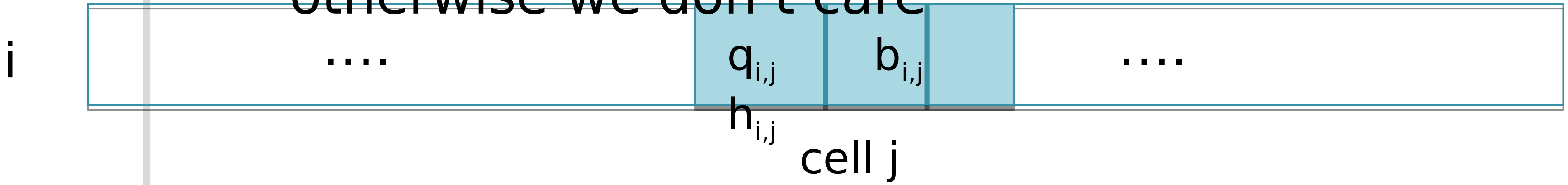
Main theorem: Step 1



A compound tape

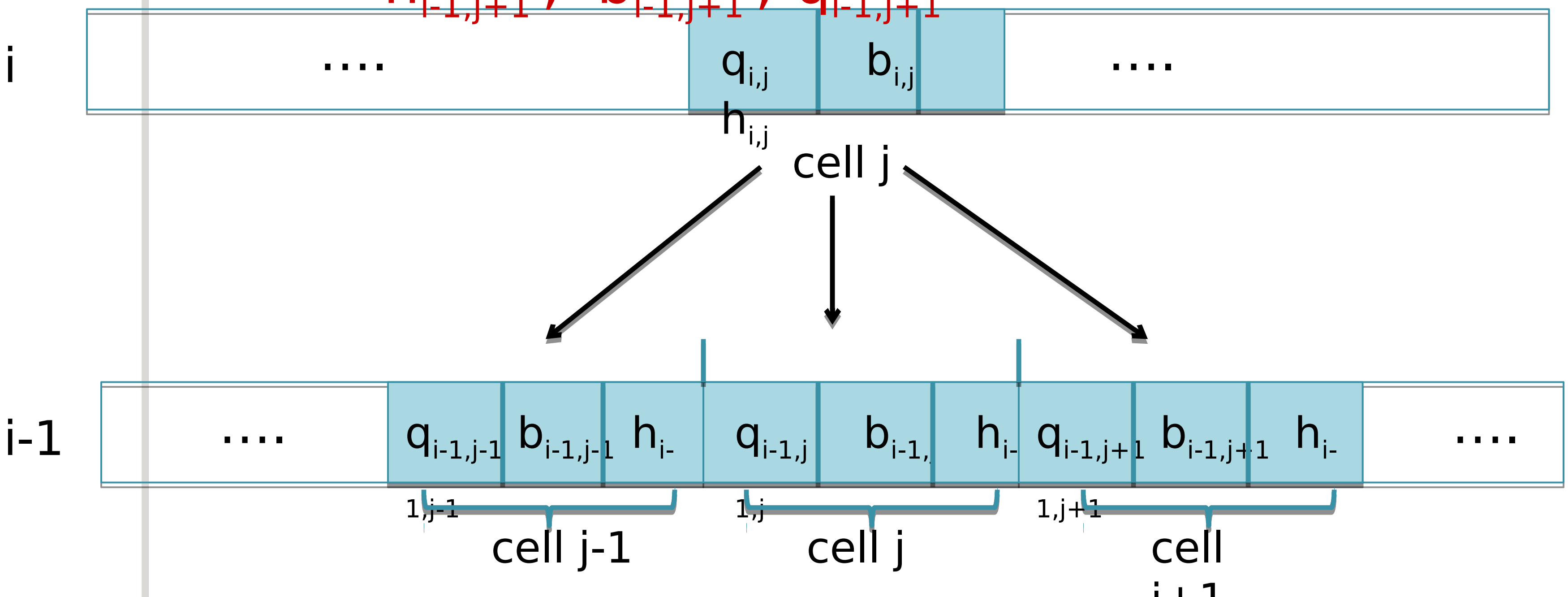
Main theorem: Step 1

- $h_{i,j}$ = 1 iff head points to cell j at i-th step
- $b_{i,j}$ = bit content of cell j at i-th step
- $q_{i,j}$ = a sequence of $\log |Q|$ bits which contains the current state info if $h_{i,j} = 1$; otherwise we don't care



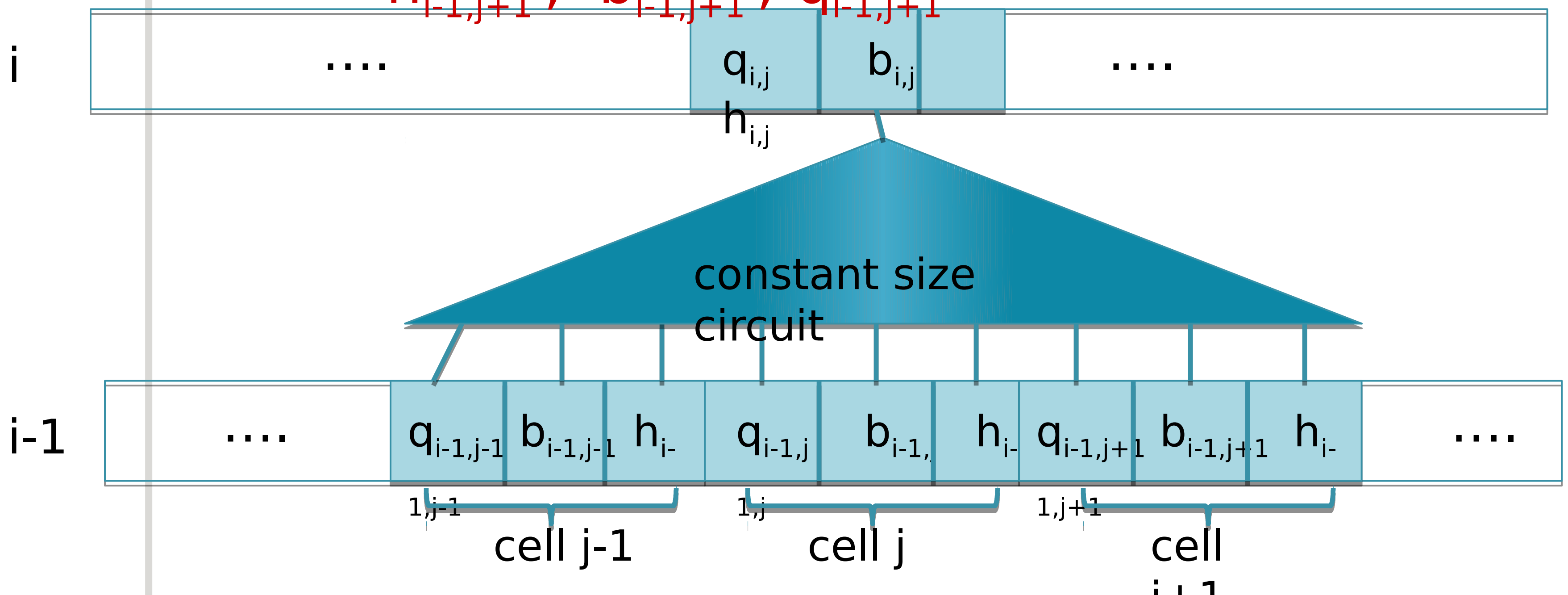
Main theorem: Step 1

- Locality of computation: The bits in $h_{i,j}$, $b_{i,j}$ and $q_{i,j}$ depend **<u>only on</u>** the bits in
  - $h_{i-1,j-1}$, $b_{i-1,j-1}$, $q_{i-1,j-1}$,
  - $h_{i-1,j}$, $b_{i-1,j}$, $q_{i-1,j}$, and
  - $h_{i-1,j+1}$, $b_{i-1,j+1}$, $q_{i-1,j+1}$

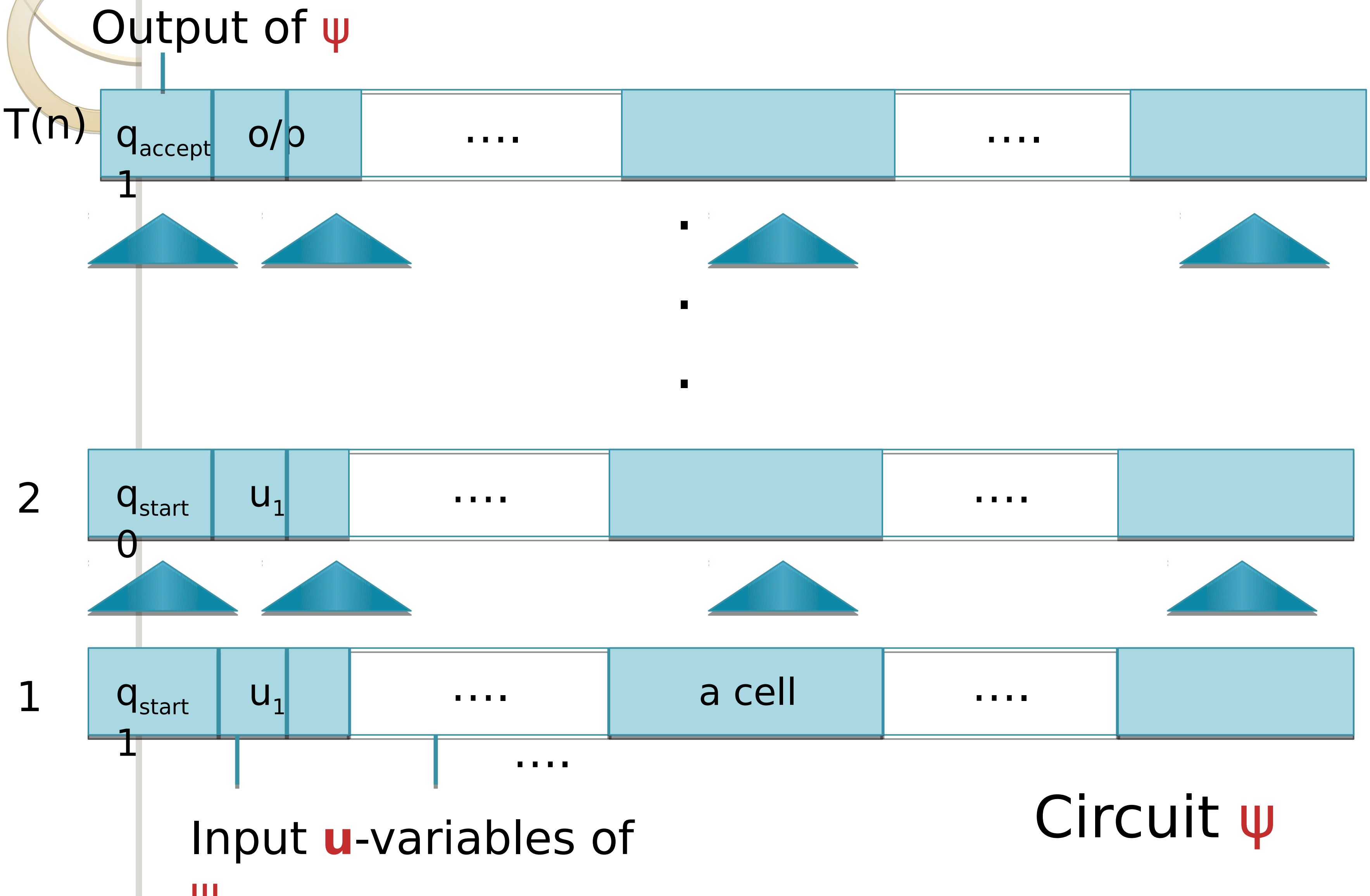


Main theorem: Step 1

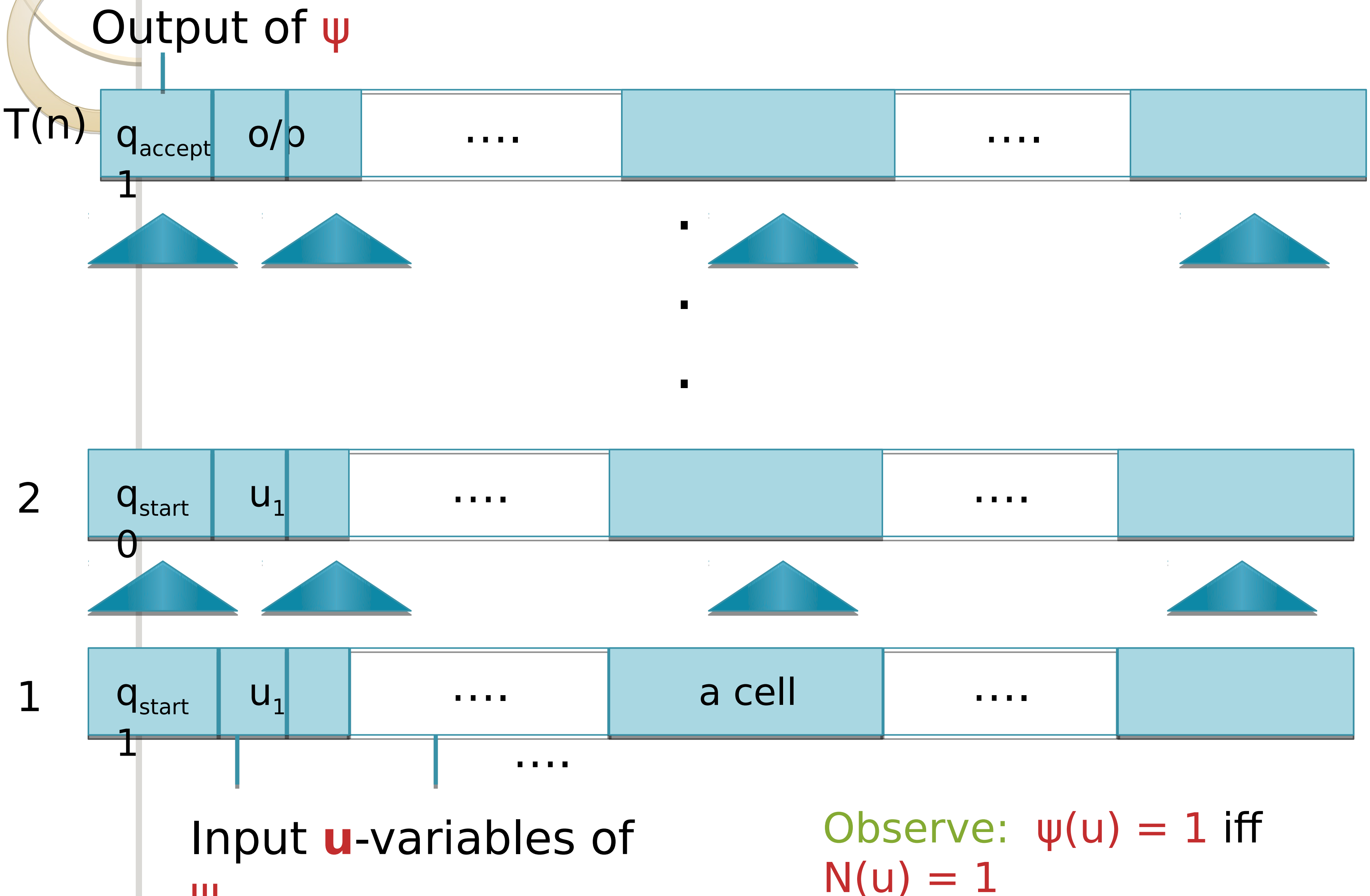
- Locality of computation: The bits in $h_{i,j}$, $b_{i,j}$ and $q_{i,j}$ depend **only on** the bits in
  - $h_{i-1,j-1}$, $b_{i-1,j-1}$, $q_{i-1,j-1}$,
  - $h_{i-1,j}$, $b_{i-1,j}$, $q_{i-1,j}$, and
  - $h_{i-1,j+1}$, $b_{i-1,j+1}$, $q_{i-1,j+1}$



Main theorem: Step 1



Main theorem: Step 1

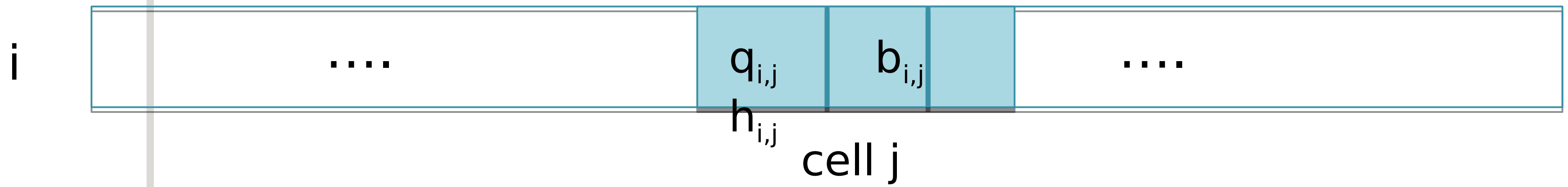


Observe: $\psi(u) = 1$ iff $N(u) = 1$

Main theorem: Step 2

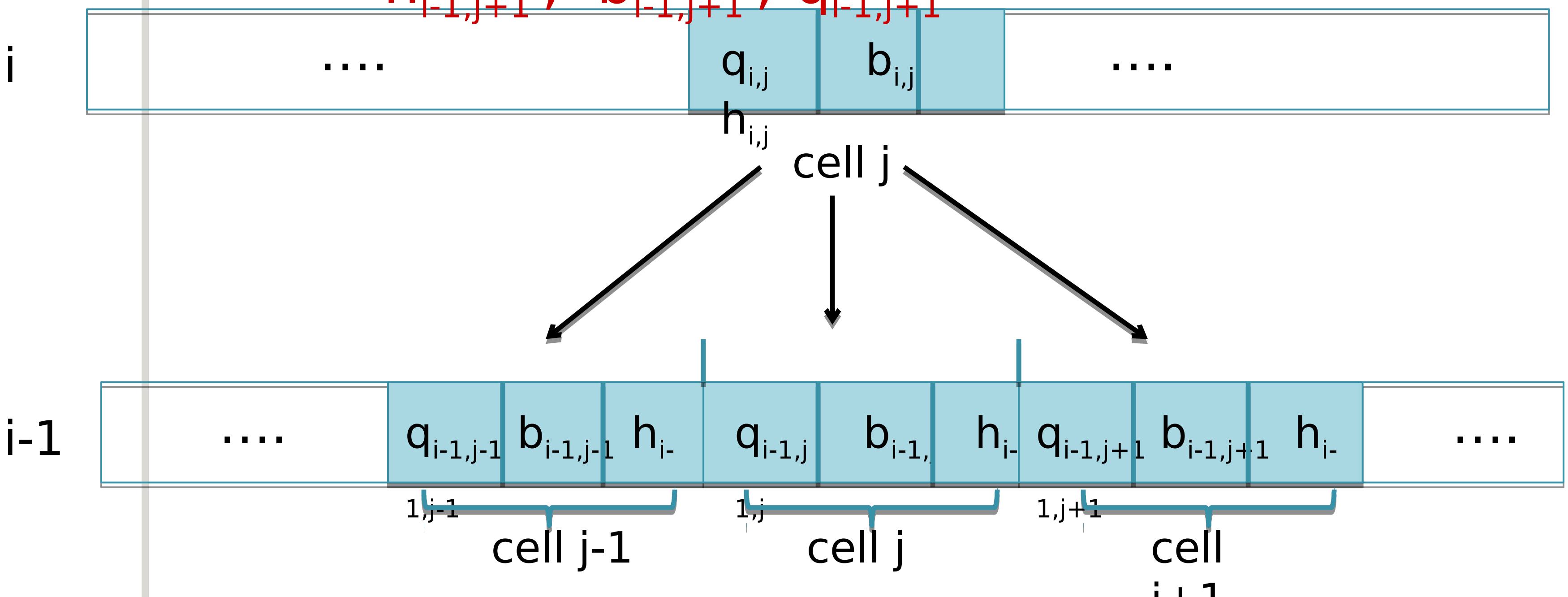
- Think of $h_{i,j}$, $b_{i,j}$ and the bits of $q_{i,j}$ as formal boolean variables.

auxiliary variables



Main theorem: Step 2

- Locality of computation: The variables $h_{i,j}$, $b_{i,j}$ and $q_{i,j}$ depend only on the variables
  - $h_{i-1,j-1}$, $b_{i-1,j-1}$, $q_{i-1,j-1}$,
  - $h_{i-1,j}$, $b_{i-1,j}$, $q_{i-1,j}$, and
  - $h_{i-1,j+1}$, $b_{i-1,j+1}$, $q_{i-1,j+1}$



Main theorem: Step 2

- Hence,

$b_{ij} = B_{ij}(h_{i-1,j-1},\ b_{i-1,j-1},\ q_{i-1,j-1},\ h_{i-1,j},\ b_{i-1,j},\ q_{i-1,j},\ h_{i-1,j+1},\ b_{i-1,j+1},\ q_{i-1,j+1})$

= a fixed function of the arguments depending only

on N's transition function δ.

- The above equality can be captured by a constant size CNF Ψ_{ij} . Also, Ψ_{ij} is easily computable from δ.

Main theorem: Step 2

- Similarly,

$h_{ij} = H_{ij}(h_{i-1,j-1}, b_{i-1,j-1}, q_{i-1,j-1}, h_{i-1,j}, b_{i-1,j}, q_{i-1,j}, h_{i-1,j+1}, b_{i-1,j+1}, q_{i-1,j+1})$

= a fixed function of the arguments depending only

on N's transition function δ.

- The above equality can be captured by a constant size CNF Φ_{ij} . Also, Φ_{ij} is easily computable from δ.

Main theorem: Step 2

- Similarly,

k-th bit of q_{ij} where $1 \le k \le \log |Q|$

$q_{ijk} = C_{ijk}(h_{i-1,j-1},\ b_{i-1,j-1},\ q_{i-1,j-1},\ h_{i-1,j},\ b_{i-1,j},\ q_{i-1,j},\ h_{i-1,j+1},\ b_{i-1,j+1},\ q_{i-1,j+1})$

= a fixed function of the arguments depending only

on N's transition function δ.

- The above equality can be captured by a constant size CNF θ_{ijk} . Also, θ_{ijk} is easily computable from δ.

Main theorem: Step 2

- Let λ be the conjunction of Ψ_{ij} , Φ_{ij} and θ_{ijk} for all i, j, k.
  - $i \in [1, T(n)]$,
  - $j \in [1, T(n)]$, and
  - $k \in [1, \log |Q|]$

- λ is a CNF in the u-variables and the auxiliary variables. Size of λ is $O(T(n)^2)$.

Main theorem: Step 2

- Let λ be the conjunction of Ψ_{ij} , Φ_{ij} and θ_{ijk} for all i, j, k.
    - $i \in [1, T(n)]$,
    - $j \in [1, T(n)]$, and
    - $k \in [1, \log |Q|]$

- λ is a CNF in the u-variables and the auxiliary variables. Size of λ is $O(T(n)^2)$.

- Define $\phi = \lambda \wedge (b_{T(n),1} = 1)$

Convert to CNF

Main theorem: Step 2

- Observe: An assignment to u and the auxiliary variables satisfies λ if and only if it "captures" computation of N on the assigned input u.

Main theorem: Step 2

- Observe: An assignment to u and the auxiliary variables satisfies λ if and only if it “captures” computation of N on the assigned input u.

- Hence, an assignment to u and the auxiliary variables satisfies ϕ if and only if N outputs 1 on the assigned input u.

Main theorem: Step 2

- Observe: An assignment to u and the auxiliary variables satisfies λ if and only if it "captures" computation of N on the assigned input u.

- Hence, an assignment to u and the auxiliary variables satisfies ϕ if and only if N outputs 1 on the assigned input u, i.e.

  ϕ(u, *"auxiliary variables"*) is satisfiable iff N(u) =1.

Main theorem: Comments

- ϕ is a CNF of size $O(T(n)^2)$ and is also computable from N in $O(T(n)^2)$ time.

- ϕ is a function of u (the input) and some "auxiliary variables" (the b_{ij}, h_{ij} and q_{ijk} variables).

- $\phi(u,$ *"auxiliary variables"*$)$ is satisfiable iff $N(u) = 1$.

Q.E.D

Main theorem: Comments

- With some more effort, size ϕ can be brought down to $O(T(n).\log T(n))$.

Main theorem: Comments

- With some more effort, size ϕ can be brought down to $O(T(n). \log T(n))$.

- The reduction from N, u to $\phi(u, \ldots)$ is not just a poly-time reduction, it is actually a *log-space reduction* (we'll define this later).

Main theorem: Comments

- With some more effort, size ϕ can be brought down to $O(T(n).\log T(n))$.

- The reduction from N, u to $\phi(u, \ldots)$ is not just a poly-time reduction, it is actually a *log-space reduction* (we'll define this later).

- Observe that once u is fixed the values of the "auxiliary variables" are also determined in any satisfying assignment for ϕ.

Main theorem: Comments

- With some more effort, size ϕ can be brought down to $O(T(n).\log T(n))$.

- The reduction from N, u to $\phi(u, \ldots)$ is not just a poly-time reduction, it is actually a *log-space reduction* (we'll define this later).

- Each clause of ϕ has only constantly many literals!

3SAT is NP-complete

- Definition. A CNF is a called a kCNF if every clause has at most k literals.

  e.g. a 2CNF $\phi = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_2)$

- Definition. kSAT is the language consisting of all *satisfiable kCNFs.*

3SAT is NP-complete

- Definition. A CNF is a called a kCNF if every clause has at most k literals.

  e.g. a 2CNF $\phi = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_2)$

- Definition. kSAT is the language consisting of all *satisfiable kCNFs.*

- Cook-Levin. There's some constant k such that kSAT is NP-complete.

3SAT is NP-complete

- Definition. A CNF is a called a kCNF if every clause has at most k literals.

  e.g. a 2CNF $\phi = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_2)$

- Definition. kSAT is the language consisting of all *satisfiable kCNFs.*

- Theorem. 3SAT is NP-complete.

Proof sketch: $(x_1 \vee x_2 \vee x_3 \vee \neg x_4)$ is satisfiable iff $(x_1 \vee x_2 \vee z) \wedge (x_3 \vee \neg x_4 \vee \neg z)$ is satisfiable.