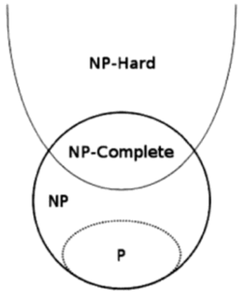
1: NP Completeness

P & NP



השוואה בין בעיות חישוב – לפי סדר גודל של סיבוכיות זמן ומקום. אנחנו יודעים לתת ניתוח כזה לאלגוריתם ספציפי.

אבל נרצה לאפיין את הבעיה עצמה. להוכיח שעבור בעיה מסויימת, כל פתרון ידרוש סיבוכיות זמן או מקום מסויימים.

לדוגמה אנחנו יודעים לנתח את אלגוריתם *Dijkstra*. אבל נרצה לאפיין את בעיית *shortest-path* עצמה.

בהינתן 2 בעיות, נרצה לקבוע מי יותר מסובכת. לדוגמה בעיות *max-flow, shortest-path*.

אנחנו יודעים שהאלגוריתמים הידועים ל-*max-flow* מסובכים יותר מאלה של *shortest-path*. העובדה הזו חסרת משמעות.

הגדרות גסות:

***P* (*polynomial*)** – קבוצת כל ה**בעיות שיש להן** אלגוריתם פולינומי. כמו בעיית *max-flow, shortest-path*.

***NP* (*nondeterministic polynomial*)** – קבוצת כל הבעיות שיש להן אלגוריתם פולינומי **אם** משתמשים באי-דטרמיניזם.

***NPC* (*NP-complete*)** – קבוצת כל הבעיות שנמצאות ב-NP, ו**קשות לפחות כמו** כל הבעיות ב-*NP*.

אם נמצא אלגוריתם פולינומי לאחת הבעיות שם, אז זה יוכיח ש- $P = NP$.

***NPH* (*NP-hard*)** – קבוצת כל השפות שקשות לפחות כמו כל הבעיות ב-*NP*.

***Self-reductions* – רדוקציות עצמיות**

בעיות הכרעה ובעיות חיפוש: **בעיית חיפוש** – מחפשים פיתרון כלשהו. **בעיית הכרעה** – בהינתן פיתרון, האם הוא נכון.

באופן כללי – בנושא של שלמות-*NP*, מספיק לדבר על בעיות הכרעה ולא צריך לדבר על בעיות חיפוש.

נדגים באמצעות בעיית *k-clique*: בהינתן גרף *G*, נגדיר $\omega(G)$ את גודל הקליקה הכי גדולה שלו. אז השפה:

$$\text{k-CLIQUE} \coloneqq \{G : \omega(G) \geq k\}$$

היא קבוצת הגרפים שיש בהם קליקה בגודל *k*.

בעיית החיפוש היא הבעיה של מציאת הקליקה K_k בגרף (אם היא קיימת). **בעיית ההכרעה** היא הבעיה של בדיקה האם $G \in \text{k-CLIQUE}$.

בעיית החיפוש שייכת ל-*P*: נעבור על כל ה- $\binom{n}{k} \leq n^k$ קבוצות של k קודקודים. לכל אחת נבדוק אם היא קליקה. זה זמן פולינומיאלי.

ובאופן טריוויאלי, זה גם פתרון פולינומי לבעיית ההכרעה: אם מצאנו קליקה, אז היא קיימת.

טריוויאלי שבעיות הכרעה לא קשות יותר מבעיות חיפוש. במקרה הזה גם ההפך נכון: בעיית החיפוש *search k-clique* לא קשה יותר מבעיית ההכרעה:

נניח שיש לנו אלגוריתם פולינומי לבעיית ההכרעה – *A*. הוא מחזיר 0 או 1. נגדיר בעזרתו אלגוריתם פולינומי לבעיית החיפוש:

קלט: גרף *G*. **פלט:** קבוצת קודקודים $V' \subseteq V(G)$ שהיא K_k.

1. אם $A(G) = 0$, החזר ϕ.
2. כל עוד $|V(G)| \geq k$ (יש בגרף לפחות *k* קודקודים):
   - 2.1 נבחר קודקוד שרירותי $v \in V(G)$.
   - 2.2 אם $A(G - v) = 1$ (כלומר, יש *k-clique* ב-*G* בלי *v*) אז נגדיר $G \coloneqq G - v$.
3. נחזיר את *G*.

אם בשלב 1 לא החזרנו ϕ, אז זה אומר שיש קליקה. בשלב 2, כל פעם נבדוק קודקוד. אם בלעדיו עדיין יש קליקה, נוריד אותו.

אם בלעדיו אין קליקה, נשאיר אותו ונבדוק את הקודקוד הבא. ככה עד שנגיע ל-*k* קודקודים. ואז מצאנו קליקה בגודל *k*.

***Deterministic Turing Machines (DTMs)* – מכונת טיורינג דטרמיניסטית**

מודל חישובי (אוטומט) המורכב מ:

- א"ב סופי Σ.
- סרט אינסופי חד צדדי, שיש לו "מיקומים".
  - כל מיקום יכול להכיל אות אחת מתוך הא"ב.

- ראש קריאה/כתיבה.
- קבוצה סופית של מצבים Q.
  - מצב התחלתי ייחודי q_{start},
  - מצב עצירה ייחודי q_{halt}.
- פונקציית מעברים $\delta: Q \times \Sigma \to \{\text{left}, \Delta, \text{right}\} \times Q \times \Sigma$.

תהליך החישוב של DTM:

- החישוב מתחיל עם הקלט כתוב על הסרט, המכונה במצב q_{start}, הראש מעל המיקום הראשון בסרט.
- כל עוד המכונה לא הגיעה ל- q_{halt}, בצע:
  - יהי $q \in Q$ המצב הנוכחי של המכונה.
  - נקרא אות $\sigma \in \Sigma$ מהמיקום הנוכחי.
  - לפי הפונקציה δ:
    - נכתוב אות σ' במיקום,
    - נישאר במקום או נזוז ימינה או שמאלה,
    - המכונה תעבור למצב q'.
- אם הגענו ל- q_{halt}: אם המילה $accept$ כתובה על הסרט, הקלט התקבל ע"י ה-DTM. אחרת, נדחה.
  - אם ה-DTM לעולם לא עוצרת, נאמר הקלט נדחה.

$L(M)$ – שפה של DTM. יהי DTM כלשהו M. אזי השפה של M היא קבוצת כל המילים שהמכונה מקבלת, ומוגדרת:

$$L(M) \coloneqq \{\omega \in \Sigma^* : M \text{ accepts } \omega\}$$

חישוב פונקציה ב-DTM: יהי DTM כלשהו M. אזי נאמר ש-M **מחשבת** פונקציה $f: \{0,1\}^* \to \{0,1\}^*$ אם לכל $x \in \{0,1\}^*$,

המכונה עוצרת ובסוף הריצה המילה $f(x)$ כתובה על הסרט. כלומר בהינתן הקלט x על הסרט, הפלט בסוף יהיה y.

מעבר אחד לפי δ נקרא **צעד חישובי** של ה-DTM. מספר הצעדים ש-DTM מבצעת בהינתן קלט הוא הזמן הנדרש לחישוב הקלט. הוא יכול להיות אינסופי.

חישוב ב-T זמן: יהיו פונקציות $T: \mathbb{N} \to \mathbb{N}$, $f: \{0,1\}^* \to \{0,1\}^*$.

נאמר ש-DTM כלשהי M מחשבת את f בזמן T אם לכל $n \in \mathbb{N}$ ולכל $x \in \{0,1\}^n$, בהינתן קלט x, M עוצרת עם $f(x)$ כתוב על הסרט, ב-$T(n)$ צעדים.

עבור פונקציה $T: \mathbb{N} \to \mathbb{N}$, נגדיר $DTIME\big(T(n)\big)$ את קבוצת כל הפונקציות הבוליאניות $f: \{0,1\}^* \to \{0,1\}$ (כלומר שפות),

שעבורן קיימת DTM שמחשבת את f בזמן $O\big(T(n)\big)$. הפונקציות האלה נקראות $O\big(T(n)\big)$-DTM-$computable$.

נגדיר את P באופן פורמלי: המחלקה P מכילה את כל השפות L שעבורן קיים DTM כלשהי M כך ש: $L = L(M)$. כלומר:

$$P \coloneqq \bigcup_{c \geq 1} DTIME(n^c)$$

כל השפות שהן $O\big(T(n)\big)$-DTM-$computable$ עבור $T(n) = n^c$ כלשהו, שזה כל הזמנים הפולינומיאלים.

מכונת טיורינג אי-דטרמיניסטית (להלן $NDTM$), בנויה באופן זהה למכונת טיורינג, אבל המעברים לא נקבעים ע"י פונקציה, אלא ע"י היחס:

$$\delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{\text{left}, \Delta, \text{right}\})$$

כלומר, δ היא בעצם זוגות של: קלט ומצב, וקלט ומצב ותזוזה. כלומר לכל זוג $(q, \sigma) \in Q \times \Sigma$, יש מספר מעברים תקינים שהמכונה יכולה לבצע.

לכן המכונה אי-דטרמיניסטית.

מילה x מעל Σ מקיימת $x \in L(M)$ אם **קיימת** סדרת מעברים שהמכונה יכולה לבצע בהינתן x, כך שמגיעים ל- q_{halt} והמילה $accept$ כתובה על הסרט.

אם אין סדרה כזו, אז $x \notin L(M)$.

אם נתעלם מהגבלות זמן, DTM שקולים ל-$NDTM$ בכך שלכל $NDTM$ יש DTM שמדמה אותה.

נאמר ש-$NDTM$ כלשהי M רצה בזמן $T(n)$ אם לכל קלט $x \in \{0,1\}^n$ ולכל סדרת מעברים, M מגיעה ל- q_{halt} (בין אם מתקבלת או לא) תוך $T(n)$ צעדים.

עבור פונקציה $T : \mathbb{N} \to \mathbb{N}$, נגדיר: $NTIME\big(T(n)\big)$ מכילה את כל השפות $L \subseteq \{0,1\}^*$ שעבורן יש $NDTM$ שרצה בזמן $O\big(T(n)\big)$,

כך ש- $L(M) = L$.

1: NP Completeness

המחלקה NP מכילה את כל השפות שעבורן קיים $NDTM$ כלשהי M כך ש $L(M)=L$, כלומר:

$$NP \coloneqq \bigcup_{c\geq 1} NTIME(n^c)$$

Verification Algorithms – אלגוריתם אימות

תהי $L\subseteq\{0,1\}^*$ שפה שמקיימת $L=L(M)$ עבור $NDTM$ **פולינומיאלי** כלשהו M. כלומר, $L\in NTIME\big(p(n)\big)$ עבור פולינום כלשהו p.

אזי, לכל $x\in L$ קיימת **תעודה כלשהי** y, (שמייצגת את הבחירות עבור M) כך שבהינתן x, אם נכריח את M ללכת לפי y, M תעצור עם הפלט 1 על הסרט.

בנוסף, מתקיים $|y|\leq p(|x|)$. כלומר זה מגביל את מספר הצעדים של-M (כי M היא מכונה בזמן פולינומי).

אם $x\notin L$, אז לא קיימת תעודה y שיכולה לגרום ל-M לעצור מ-x עם 1 על הסרט.

זה מוביל אותנו להגדרה אלטרנטיבית ל- NP. שפה $L\subseteq\{0,1\}^*$ היא ב-NP אם קיימים:

פולינום $p:\mathbb{N}\to\mathbb{N}$, ו-$NDTM$ כלשהו M (אלגוריתם), כך ש: לכל $x\in\{0,1\}^*$,

מתקיים $x\in L$ **אם ורק אם** קיים $y\in\{0,1\}^{\leq p(|x|)}$ כך ש-M בהינתן x ו-y עוצר עם $accept$, תוך $p(|x|+|y|)$ צעדים.

כלומר התנאים:

1. האלגוריתם מחזיר אמת עבור y אמ"מ x שייך לשפה,
2. הגודל של y פולינומי בגודל של x,
3. האלגוריתם רץ בזמן פולינומי ב- $|x|+|y|$.

ה-$NDTM$ מההגדרה הזו נקרא **אלגוריתם אימות** עבור L.

טענה: $\text{k-CLIQUE}\in NP$. **הוכחה** – נגדיר אלגוריתם אימות:

קלט: גרף G, וקבוצת קודקודים X. הקבוצה X היא **העד.** אם $G[X]\cong K_k$ (k-קליקה) נחזיר 1. אחרת, 0.

האלגוריתם עומד בתנאים: אם $G\in\text{k-CLIQUE}$, אז אכן קיים X מתאים. והגודל שלו פולינומי בגודל הגרף. אחרת, לא קיימת קבוצה X כזו.

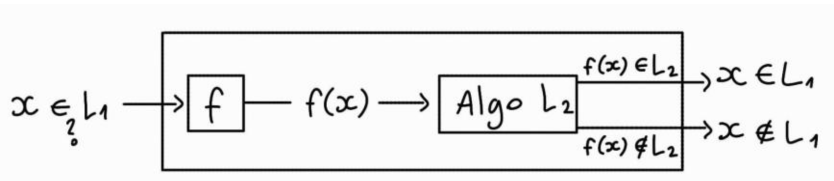
Polynomial Time Reductions

יהיו שפות $L_1,L_2\subseteq\{0,1\}^*$. נסמן $L_1\leq_p L_2$ אם קיימות: פונקציה $f:\{0,1\}^*\to\{0,1\}^*$ ופולינום $p:\mathbb{N}\to\mathbb{N}$ כך ש:

1. מתקיים $x\in L_1 \Leftrightarrow f(x)\in L_2$,
2. לכל $x\in L_1$, אפשר לחשב את $f(x)$ בזמן $p(|x|)$.

אם $L_1\leq_p L_2$, נאמר שניתן לעשות רדוקציה מ- L_1 ל- L_2 בזמן פולינומי. הפונקציה f נקראת **רדוקציה פולינומית**.

במקרה זה נאמר ש L_1 <u>קשה לכל היותר כמו</u> L_2. למה? כי אם ידוע ש $L_1\leq_p L_2$, ונתון אלגוריתם פולינומי עבור L_2, אז יש לנו אלגוריתם פולינומי ל- L_1:



שפה $L\subseteq\{0,1\}^*$ תיקרא NP*-hard* אם $L'\leq_p L$ לכל $L'\in NP$.

כלומר, אם מכל שפה ב-NP אפשר לעשות רדוקציה בזמן פולינומי ל-L. זה אומר שהבעיה של L קשה **לפחות כמו** L'.

בדרך כלל כדי להוכיח ששפה L היא NPH, ניקח שפה L^* שאנחנו יודעים שהיא NPH, ונעשה רדוקציה ממנה ל-L.

NPC זה כל השפות ב- $NP\cap NPH$. אם נוכיח $P\cap NPC\neq\phi$, זה יוכיח ש $P=NP$.

The Cook-Levine Theorem

הזיהוי הראשון של שפה NPC. הקדמה – CNF-SAT:

יהיו משתנים בוליאניים $x_1,\dots,x_n$. לכל משתנה יש את הליטרלים שלו: החיובי x_i, השלילי $\bar{x}_i$.

1: NP Completeness

אם x_i קיבל את הערך אמת, אז $\bar{x}_i$ יקבל שקר. אם x_i קיבל שקר, אז $\bar{x}_i$ מקבל אמת.

נוסחה בוליאנית φ תיקרא *CNF* (*conjunctive normal form*) אם היא מורכבת מפסוקיות שביניהן יש רק $\vee$ (*OR*),

ובתוך כל פסוקית הליטרלים מופרדים על ידי $\wedge$ (*AND*). "מכפלה של סכומים":

$$\varphi = (x_1 \vee \bar{x}_4 \vee x_3 \vee \bar{x}_5) \wedge x_2 \wedge (\bar{x}_2 \vee \bar{x}_5)$$

נוסחה כזו תיקרא **ספיקה** אם קיימת השמה למשתנים שלו כך שהנוסחה יוצאת אמת. השמה כזו תיקרא **השמה מספקת.**

לדוגמה, ההשמה: $x_1 = x_2 = T, x_3 = x_4 = x_5 = F$. אז:

$$\varphi = (T \vee \bar{F} \vee F \vee \bar{F}) \wedge T \wedge (\bar{T} \vee \bar{F}) = (T \vee T \vee F \vee T) \wedge T \wedge (F \vee T) = T \wedge T \wedge T = T$$

נגדיר את השפה:

$$\text{CNF-SAT} := \{\varphi : \varphi \text{ is a satisfiable Boolean formula in CNF}\}$$

כלומר, כל הנוסחאות הבוליאניות שהן נוסחת *CNF* ספיקה. משפט *Cook-Levine* – השפה *CNF-SAT* היא ב-*NPC*.

עבור מספר טבעי k, נגדיר:

$$\text{k-CNF-SAT} \coloneqq \{\varphi\text{: satisfiable CNF formulas with } k \text{ literals in each clause}\}$$

כלומר ביטויי *CNF* עם k ליטרלים בכל פסוקית.

טענה: *3-CNF-SAT* היא *NPC*. קשה לפחות כמו כל בעיה ב-*NP*. המעבר מ-2 ל-3 העביר אותנו מ-*P* ל-*NPC*.

הוכחה: נוכיח שהיא ב-*NP* וגם *NPH*. כדי להוכיח שהיא ב-*NP*, נגדיר אלגוריתם אימות לבעיה:

קלט: φ, נוסחת *3-CNF*. a, השמה למשתנים של φ. אם a מספקת את φ, נחזיר אמת. אחרת, שקר.

האלגוריתם מקיים את 3 התנאים הבאים באופן טריוויאלי:

1. האלגוריתם מחזיר אמת **אם ורק אם** $\varphi \in \text{3-CNF-SAT}$.
2. הגודל של a פולינומי בגודל של φ.
3. האלגוריתם רץ בזמן $|\varphi| + |a|$.

כדי להוכיח שהשפה ב-*NPH*, צריך להראות ש $\forall L \in NP,\ L \leq_p \text{3-CNF-SAT}$. למעשה, נוכיח ש $L^* \leq_p \text{3-CNF-SAT}$ עבור $L^* \in NPH$ כלשהי.

נשתמש ב *CNF-SAT* כי אנחנו יודעים שהיא *NPH*. (אמרנו שהיא *NPC*, וזה גם *NPH*). נצטרך להראות ש $\text{CNF-SAT} \leq_p \text{3-CNF-SAT}$. נמצא פונקציה f כך שבהינתן נוסחה φ שהיא *CNF*, יתקיים:

1. $f(\varphi)$ היא נוסחת *3-CNF*.
2. מתקיים: $f(\varphi)$ ספיקה **אם ורק אם** φ ספיקה.
3. ניתן לחשב את $f(\varphi)$ בזמן פולינומי בגודל של φ.

הבהרה: הפונקציה הזו לא פותרת את בעיית *CNF-SAT* או *3-CNF-SAT*. היא רק מתרגמת ביניהן. אם נמצא אלגוריתם פולינומי ל-*3-CNF-SAT*, אז יהיה אלגוריתם פולינומי ל-*CNF-SAT*.

נגדיר את הפונקציה: נחליף כל פסוקית $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_m$ **בגאדג'ט הפסוקית** (*clause gadget*) לפי המקרים:

אם $m = 3$, נעתיק את הפסוקית כמו שהיא.

אם $m < 3$, נחזור על ליטרלים כדי להשלים ל-3:

$$C = \ell_1 \longrightarrow C' = \ell_1 \vee \ell_1 \vee \ell_1, \qquad C = \ell_1 \vee \ell_2 \longrightarrow C' = \ell_1 \vee \ell_2 \vee \ell_2$$

אם $m > 3$, נגדיר $m - 3$ משתנים חדשים: $y_1^C, y_2^C, \dots, y_{m-3}^C$.

לכל C כזו, הפונקציה f תגדיר גאדג'ט לאותו C ומשתנים אלו יופיעו רק בגאדג'ט של C. הגאדג'ט עבור הפסוקית C יוגדר:

$$(\ell_1 \vee \ell_2 \vee y_1) \wedge (\overline{y_1} \vee \ell_3 \vee y_2) \wedge (\overline{y_2} \vee \ell_4 \vee y_3) \wedge \cdots \wedge (\overline{y_{m-3}} \vee \ell_{m-1} \vee \ell_m)$$

דוגמאות:

$$\varphi = \underbrace{(x_1 \vee \overline{x_2})}_{C_1, m=2} \wedge \underbrace{(x_3 \vee x_4 \vee \overline{x_2} \vee x_1)}_{C_2, m=4} \longrightarrow f(\varphi) = \underbrace{(x_1 \vee \overline{x_2} \vee \overline{x_2})}_{G_1} \wedge \underbrace{\left(x_3 \vee x_4 \vee y_1^{C_2}\right) \wedge \left(\overline{y_1^{C_2}} \vee \overline{x_2} \vee x_1\right)}_{G_2}$$

1: NP Completeness

$$\varphi=(x_1\vee x_2\vee\overline{x_3}\vee x_4\vee x_5\vee\overline{x_6}\vee x_7)$$
$$\longrightarrow f(\varphi)=\ (x_1\vee x_2\vee y_1)\wedge(\overline{y_1}\vee\overline{x_3}\vee y_2)\wedge(\overline{y_2}\vee x_4\vee y_3)\wedge(\overline{y_3}\vee x_5\vee y_4)\wedge(\overline{y_4}\vee x_6\vee x_7)$$

$$\varphi=\underbrace{(x_1\vee\overline{x_2}\vee x_3\vee\overline{x_4})}_{C_1}\wedge\underbrace{(\overline{x_1}\vee\overline{x_3}\vee\overline{x_5}\vee x_4)}_{C_2}$$
$$\longrightarrow f(\varphi)=\ \underbrace{\left(x_1\vee\overline{x_2}\vee y_1^{C_1}\right)\wedge\left(\overline{y_1^{C_1}}\vee x_3\vee\overline{x_4}\right)}_{G_1}\wedge\underbrace{\left(\overline{x_1}\vee\overline{x_3}\vee y_1^{C_2}\right)\wedge\left(\overline{y_1^{C_2}}\vee\overline{x_5}\vee y_4\right)}_{G_2}$$

נוכיח ש-f מקיימת את 3 התנאים לכל φ שהיא נוסחת CNF:

1. $f(\varphi)$ היא נוסחת 3-CNF. (טריוויאלי, כי לפי הגדרת הבנייה כל G תהיה באורך 3).

3. ניתן לחשב את $f(\varphi)$ בזמן פולינומי – כי מספר המשתנים החדשים פולינומי ב- $|\varphi|$, ומספר הפסוקיות החדשות פולינומי ב-$|\varphi|$.

הוכחת תנאי 2:

כיוון ראשון: נניח ש φ ספיקה. תהי a השמה המספקת את φ.

נזכור ש $f(\varphi)$ מכילה משתנים חדשים שאינם ב-φ המקורית, ולכן יכול להיות ש-a אינה השמה תקינה עבור $f(\varphi)$.

נגדיר a' השמה מספקת עבור $f(\varphi)$: לכל x שהוא משתנה מקורי של φ, $a'(x)=a(x)$.

לכל פסוקית $C=\ell_1\vee\ell_2\vee\cdots\vee\ell_m$ באורך $m\geq 4$, יש משתנים לוקאליים y. למשתנים אלו אין השמה ב-a. ניתן להם השמה:

מכיוון ש-a מספקת את φ, קיים $i\in[m]$ כך ש- $\ell_i=1$ (יש ליטרל חיובי כלשהו ב-C). ואותו ליטרל הוא משתנה מקורי של φ.

לכל $j\in[m-3]$: אם $j<i$ זה משתנים שמופיעים לפני ℓ_i בגאדג'ט, אם $j\geq i$ זה משתנים שמופיעים אחרי. נגדיר:

$$a'\left(y_j^C\right)=\begin{cases}1, & j<i\\ 0, & j\geq i\end{cases}$$

נוכיח שזו השמה מספקת:

כל הפסוקיות בגאדג'ט שאינן מכילות את ℓ_i ומופיעות לפניו, מסופקות כי $a\left(y_j^C\right)=1$. הפסוקית שמכילה את ℓ_i מסופקת כי $\ell_i=1$.

כל הפסוקיות אחרי ℓ_i מסופקות בגלל ה- $\bar{y}_j$.

כיוון שני: נניח ש $f(\varphi)$ ספיקה, ותהי a' השמה מספקת עבורה. נגדיר השמה a המספקת את φ: לכל x שהוא משתנה מקורי של φ, $a'(x)=a(x)$.

נוכיח שזו השמה מספקת: נב"ש שלא, כלומר קיימת פסוקית $C=\ell_1\vee\ell_2\vee\cdots\vee\ell_m$ של φ כך ש: $\ell_1=\ell_2=\cdots=\ell_m=0$.

עבור $m\leq 3$, זה אותם משתנים אז ההשמה לא מספקת גם את $f(\varphi)$, סתירה.

עבור $m\geq 4$, נבחר את הגאדג'ט המתאים ל-C ב- $f(\varphi)$: בגלל שאנחנו יודעים ש-a' מספקת את $f(\varphi)$, ו-a לא מספקת את φ חייב להתקיים:

$$(\ell_1\vee\ell_2\vee y_1)\wedge(\overline{y_1}\vee\ell_3\vee y_2)\wedge(\overline{y_2}\vee\ell_4\vee y_3)\wedge\cdots\wedge(\overline{y_{m-3}}\vee\ell_{m-1}\vee\ell_m)$$

(המסומנים בירוק הם 1, כל השאר 0). כי כל ה-ℓ הם 0, אז y_1 חייב להיות 1. אז $\overline{y_1}$ הוא 0, אז y_2 חייב להיות 1...

כל פעם, זה מכריח את ה-y הבא להיות 1. ובפסוקית האחרונה, נקבל שהכל 0.

כלומר בכל מקרה הפסוקית האחרונה תהיה לא מסופקת, בסתירה לכך ש-a' מספקת את $f(\varphi)$.

בסך הכל, נקבל ש הפונקציה f שלנו תקינה, כלומר $\text{CNF-SAT}\leq_p\text{3-CNF-SAT}$ אז *3-CNF-SAT* היא ב-*NPH*.

ומכיוון שהראנו שהיא *NP*, בסך הכל קיבלנו שהיא *NPC*, כנדרש.